Parker

## Programmer's Guide

# ACR Series Controllers

Effective: June, 2021
Document Number: 88-028698-01E

ENGINEERING YOUR SUCCESS.

# User Information

**Warning:** **ACR7000 and IPA products are used to control electrical and mechanical components of motion control systems. You should test your motion system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.**

**Warning:** **Risk of damage and/or personal injury.**
**The ACR7000 and IPA described in this guide contain no user-serviceable parts. Attempting to open the case of any unit, or to replace any internal component, may result in damage to the unit and/or personal injury. This may also void the warranty.**



## Contact Information for Technical Assistance

Contact your local automation technology center (ATC) or distributor.

**North America and Asia**
Parker Hannifin
Electronic Motion and Controls Division
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (707) 584-7558
Fax: (707) 584-8029
Email: emn.service@support.parker.com
Internet: http://www.parkermotion.com

# Important Safety Information

It is important that motion control equipment is installed and operated in such a way that all applicable safety requirements are met. It is your responsibility as an installer to ensure that you identify the relevant safety standards and comply with them; failure to do so may result in damage to equipment and personal injury. In particular, you should study the contents of this user guide carefully before installing or operating the equipment.

The installation, set up, test and maintenance procedures given in this user guide should only be carried out by competent personnel trained in the installation of electronic equipment. Such personnel should be aware of the potential electrical and mechanical hazards associated with mains-powered motion control equipment—please see the safety warnings below. The individual or group having overall responsibility for this equipment must ensure that operators are adequately trained.

Under no circumstances will the suppliers of the equipment be liable for any incidental, consequential or special damages of any kind whatsoever, including but not limited to lost profits arising from or in any way connected with the use of the equipment or this guide.

**Warning:** **High-performance motion control equipment is capable of producing rapid movement and very high forces. Unexpected motion may occur especially during the development of controller programs. KEEP WELL CLEAR of any machinery driven by stepper or servo motors. Never touch any part of the equipment while it is in operation.**

**This product is sold as a motion control component to be installed in a complete system using good engineering practice. Care must be taken to ensure that the product is installed and used in a safe manner according to local safety laws and regulations. In particular, the product must be positioned such that no part is accessible while power may be applied.**

**This and other information from Parker Hannifin Corporation, its subsidiaries, and authorized distributors provides product or system options for further investigation by users having technical expertise. Before you select or use any product or system, it is important that you analyze all aspects of your application and review the information concerning the product in the current product catalog. The user, through its own analysis and testing, is solely responsible for making the final selection of the system and components and assuring that all performance, safety, and warning requirements of the application are met.**

**If the equipment is used in any manner that does not conform to the instructions given in this user guide, then the protection provided by the equipment may be impaired.**

# Contents

# Change Summary

The change summary below lists the latest additions, changes, and corrections to the ACR Programmer's Guide and the corresponding section of Parker Motion Manager Online Help.

## *Revision E Changes*

Document 88-028698-01E (ACR Programmer's Guide) supersedes document 88-028698-01D. Changes associated with this document are noted in this section.

- Updated for ACR7000 series and IPA, adding Parker Motion Manager. For prior ACR products, see previous revision D.

# Before We Begin

This document is intended to accompany the printed and online documents listed below, as part of the ACR product user documentation.

| Reference Document | Description |
|---|---|
| PMM Quick Start Guide | Walkthrough of Parker Motion Manager for first time users |
| ACR Command Reference | Provides detailed descriptions of all AcroBASIC language commands with examples |
| ACR Parameter & Bit Reference | Provides list of all ACR and IPA Parameters and Bits with explanations |
| EtherNet/IP User Guide Feb 2015 | How to setup ACR7000 or IPA as master for Wago 750 series expansion I/O |
| ComACRServer6 User Guide | Provides information about ComACRserver6 and detailed descriptions of its properties and methods for PC interface via Visual Basic .NET, Visual C++, Visual C#, Wonderware or LabView |
| ACR7000 Stepper Controller Hardware Guide | Hardware-related information specific to the ACR7000 Stepper |
| ACR7000 Stepper Connection Guide | IO connection document |
| ACR7000 Servo Controller User Guide | Hardware-related information specific to the ACR7000 servo |
| ACR7000 Servo Connection Guide | IO Connection document |
| ACR7000 Controller Hardware Guide | Hardware-related information specific to the ACR7000 controller |
| ACR7000 Controller Connection Guide | IO connection document |
| IPA Hardware Installation Guide | Hardware-related information specific to the IPA |
| IPA Quick Reference Guide | IO connection document |

# *Assumptions of Technical Experience*

To effectively use the information in this manual, you should have a fundamental understanding of the following:

- Electronics concepts such as voltage, switches, current, etc.
- Motion control concepts such as motion profiles, torque, velocity, distance, force, etc.
- Programming skills in a high-level language such as C or Python is helpful.
- Ethernet communication and networking.
- Safety requirements, standards and best practices for automation machinery.

If you are new to the AcroBASIC Programming Language, read the Quick Start and Chapter 1 thoroughly.

# *Before You Begin*

Before you begin to implement the ACR or IPA controller's features in your application you should complete the items listed below.

- Complete all the installation provided in Hardware Installation Guide.
- For linear actuators, precision stages, linear motor systems and systems with limited travel, install end-of-travel sensors and enable and test end-of-travel sensors.
- If you are controlling any servo axes, complete the servo tuning procedures. Be sure to use Parker Motion Manager's built-in tuning utility to easily tune the axis and integrate the gains into your motion program.
- If you are new to the AcroBASIC Programming Language, begin with the Parker Motion Manager Quick Start and read Chapter 1 (Programming Basics) thoroughly.

Keep in mind that this Programmer's Guide covers most of what programmers need, but it is ultimately the responsibility of the programmer to consider the requirements of the machine and develop their application accordingly.

# CHAPTER 1
# Parker Motion Manager

# Parker Motion Manager

The ACR7000 series controller and IPA are configured and programmed with Parker Motion Manager (PMM), a Windows-based programming tool designed to simplify and speed up your ACR programming efforts.

PMM's Configuration Wizard has been streamlined to help you quickly set the controller's:

- Units for each Master.
- Motor parameters for each axis.
- Scaling for each axis.
- Inputs for Limit and Home sensors.

PMM is an updated code development tool enabling programmers to:

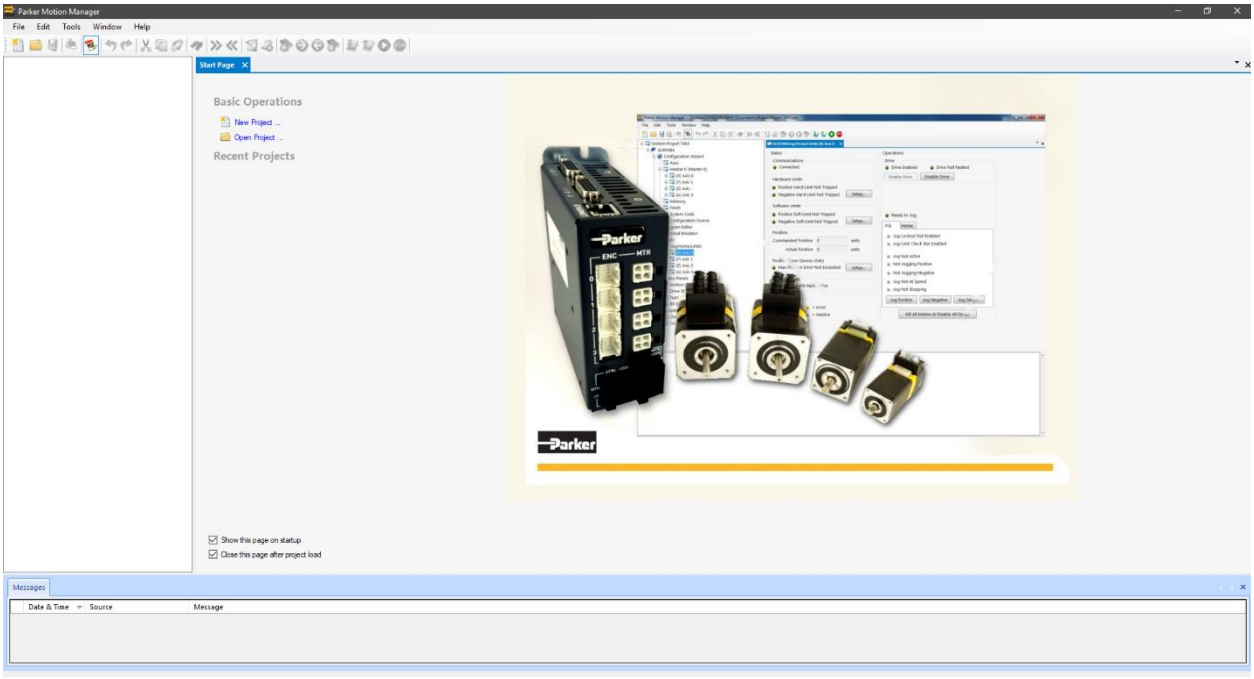
- Create, edit, download and upload AcroBASIC programs.
- Test and debug programs and controller operation.
- Test motion and tune your system to optimize performance.
- Monitor controller, integrated drive, bit and parameter status.
- Use high-performance software oscilloscopes for advanced programming.
- Use an improved Servo Tuner screen featuring auto-scaling graphs.

Ease of use improvements:

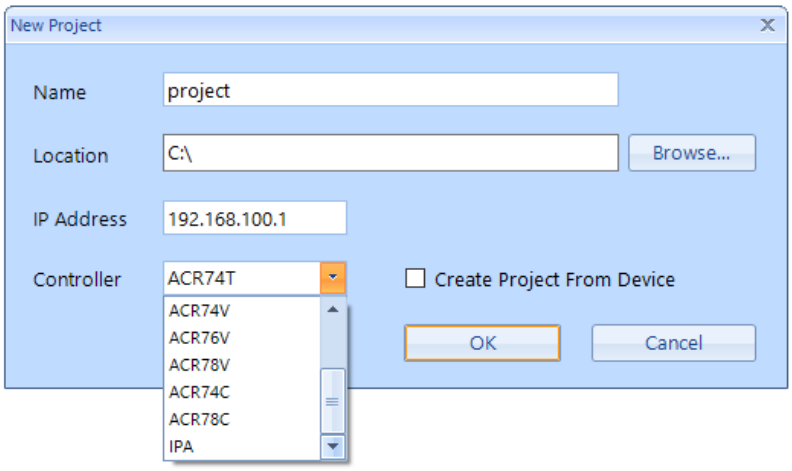
- Start Page showing recent projects.
- Projects stored as single files for easy sharing and archiving.
- Improved Terminal Emulator with user buttons, preset buttons for common commands and a command repeat feature that can be accessed using the arrow keys.
- Product-specific status panels.
- Copy Axis feature to save time when configuring similar axes.

As program development is done within PMM, let's first learn the main parts of PMM.

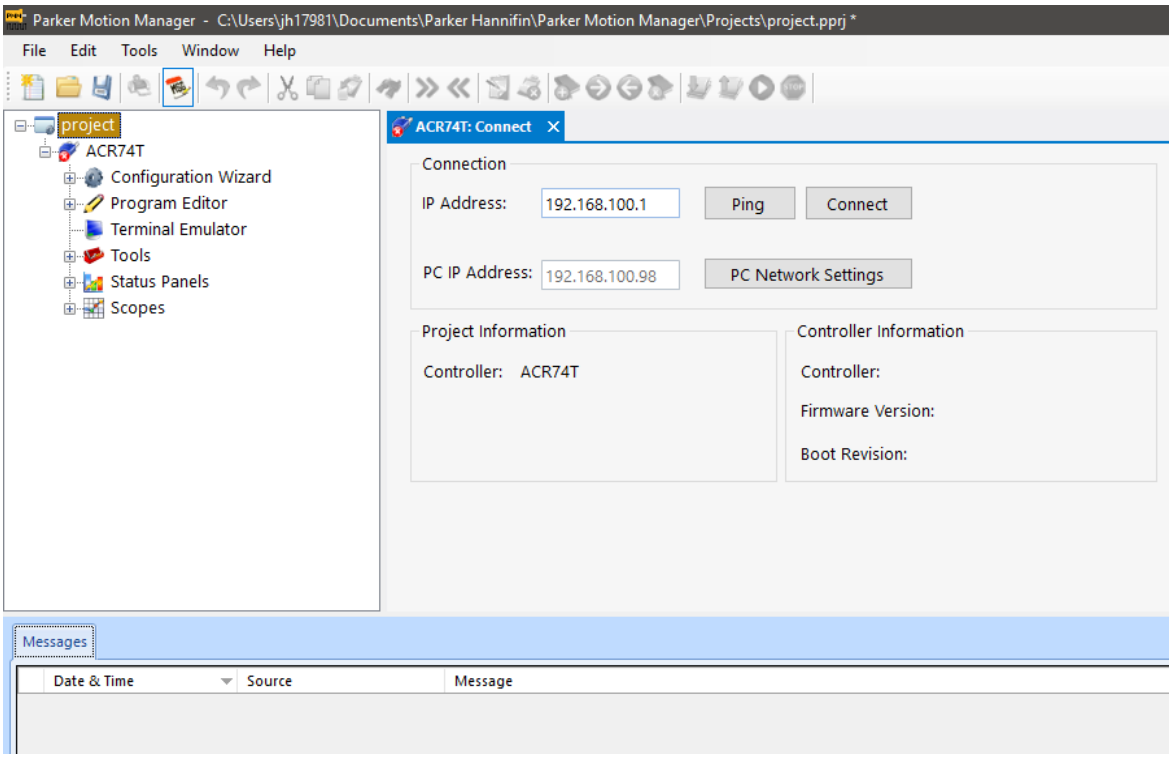
# Getting Started with PMM




When first starting PMM, the Start screen will appear. A new project can be initiated or an existing project can be opened. As projects are created, they will appear under Recent Projects. The Start screen can be disabled or re-enabled under *Tools → Options*. That menu also allows clearing the list of Recent Projects.




When creating a new project, give it a name, a location on the hard drive and a type (model number on the side of controller). The IP address has been set as the controller default of 192.168.100.1. Users can also upload from the controller for existing machines using the *Create Project From Device* check box.




# *Connection*

The Connect window can be opened by clicking the controller name in the Explorer (left-hand side of PMM). The connection status is shown on the Explorer. The red circle with white X will appear when not connected to the controller, making it easy to determine if you are not connected when in the Program Editor, Status Panels, Scopes, etc.

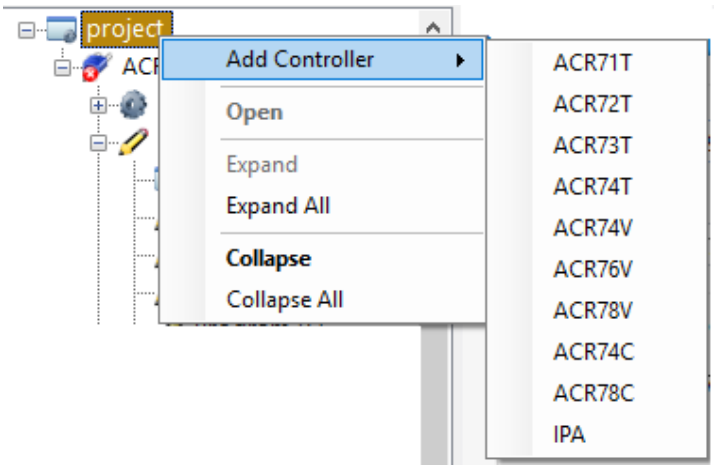
The *Ping* button performs a quick check to see if your PC can see the controller's IP address successfully.

The *Connect* button attempts to connect to the controller specified by the *IP Address* field.

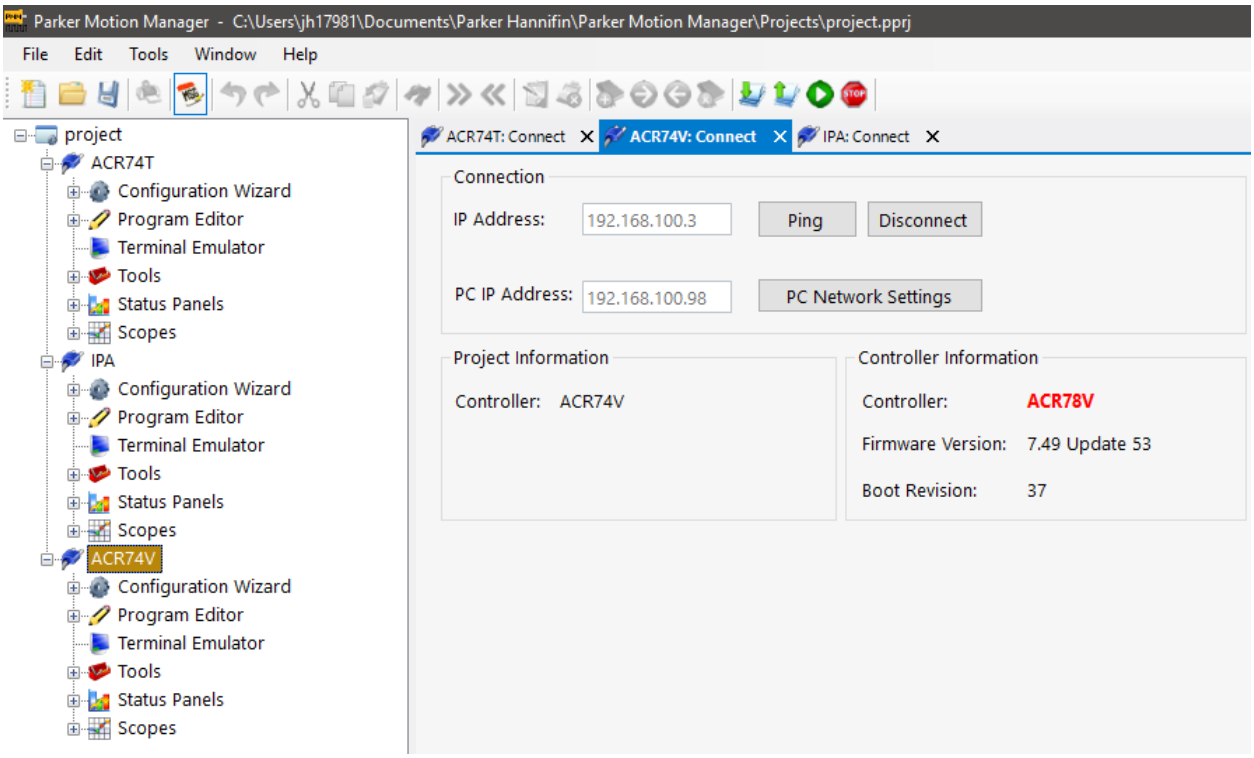
The *PC IP Address* field displays your PC's IP address. The first three octets (numbers) of this address will need to match the first three octets of the controller's IP address (192.168.100.x). The last number will be between 2-255 and unique on your network. Use the *PC Network Settings* button to change your PC's IP address.

Need to change PC's IP address? See Setting the IP Address and Subnet Mask—PC.

A project can have multiple controllers. Right-click on the project name to add additional controllers. This can be for machines with more than one controller that store all of their configurations in one project file.




Multiple connections are supported but each controller will need its own unique IP address. For the ACR7000, use the `IP` command in the Terminal Emulator to change the controller's IP address. Issue the `ESAVE` command and cycle power to make it take effect. Be sure to label and note the controller's IP address!




For the IPA, use dial the switches (S1 and S10) to set the IP address, or set the dials to 99 and use the `IP` command like with the ACR7000.

Troubleshooting a connection? See Connecting to the Controller.

# *Uploading a Project from the Controller to PMM*

One of the most common tasks with any motion controller is to upload a project from an installed controller so that it can be downloaded to a replacement controller. There are two ways to upload a project in PMM:

- If there is no pre-existing project file on the PC, the user can upload the entire project from the controller using the *New Project* dialog.
- If a project is already loaded in PMM, it can be updated to match the project on the controller by using the upload button in the Toolbar.

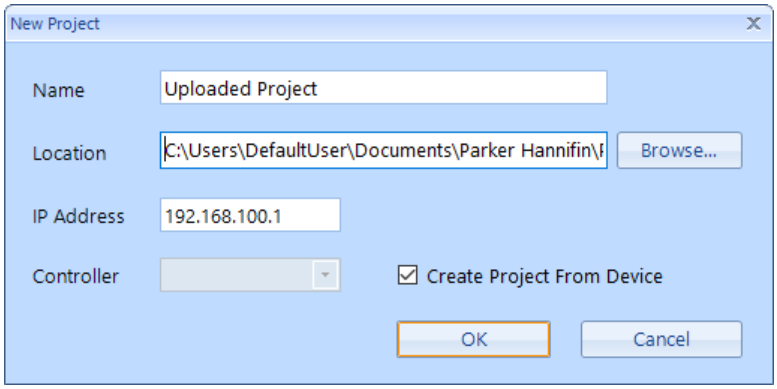
This guide will show uploading from the New Project dialog as it is the better option for quickly replacing a controller.

## Procedure

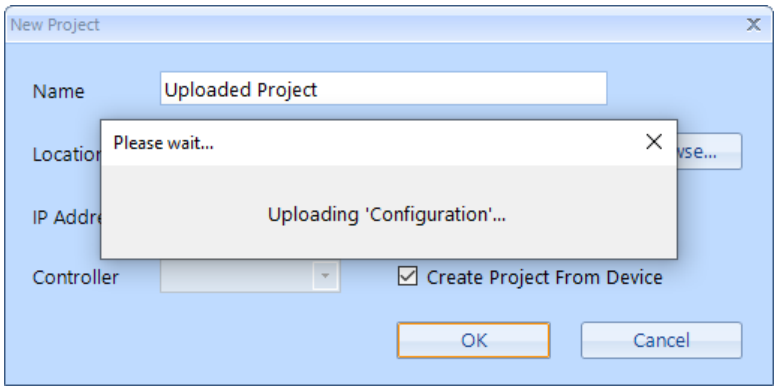
Step 1: Open Parker Motion Manager.

Step 2: Click *File* → *New Project* (or the equivalent Toolbar button).

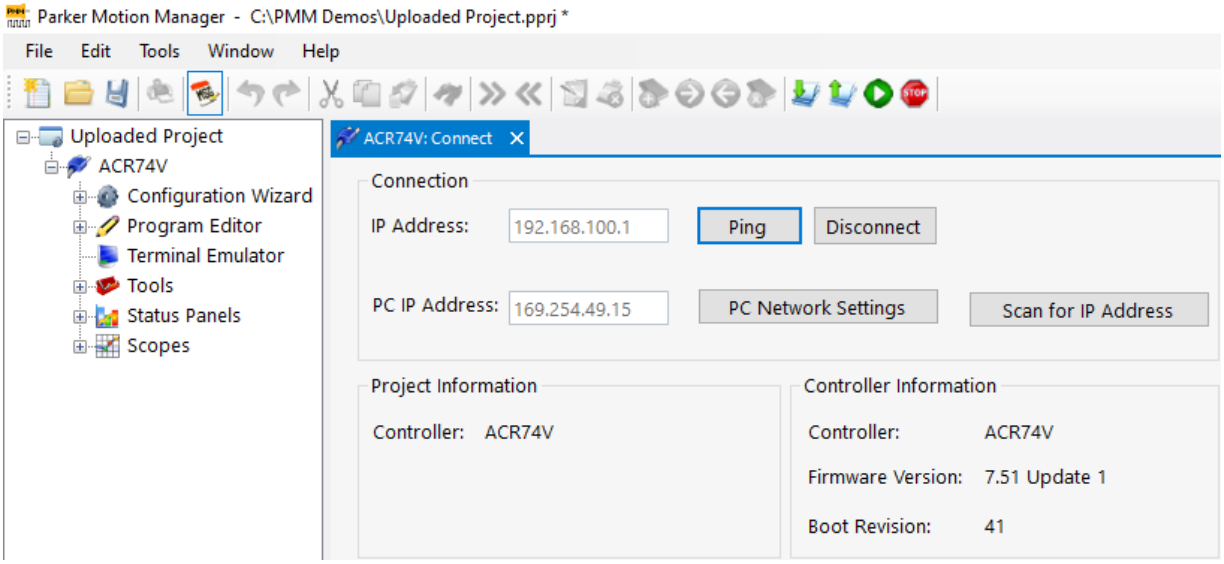
Step 3: Give the project a name and choose a location to store the uploaded file. The defaults are fine. Put in the IP address of the controller from which you want to upload. Check the *Create Project From Device* check box. Click *OK*.




Step 4: Wait for the upload to complete.



Step 5: Once the upload is complete, the project will be loaded and PMM will connect to the controller automatically.




Step 6: Save the uploaded project to the PC by clicking *File* →*Save Project* or *File* → *Save Project As*. There is an equivalent Toolbar button for this as well.

# *Downloading a Project from PMM to the Controller*

There are two ways to download a project from PMM to an ACR controller:

- The Configuration Wizard has a check box on the Finish screen that initiates a download when the user clicks *Finish*.
- The project can be downloaded using a button in the Toolbar.

This procedure will be explained assuming the user is downloading via the button in the Toolbar.

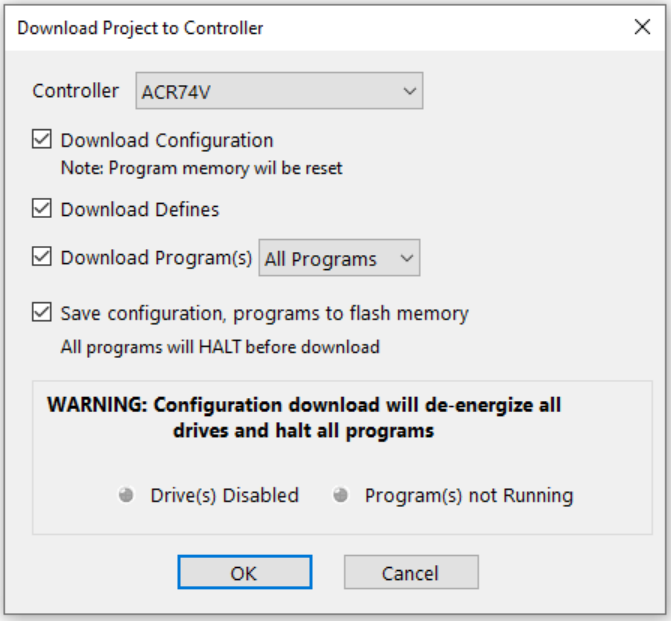
## Procedure

Step 1: Open the Parker Motion Manager project that is going to be downloaded to the controller.

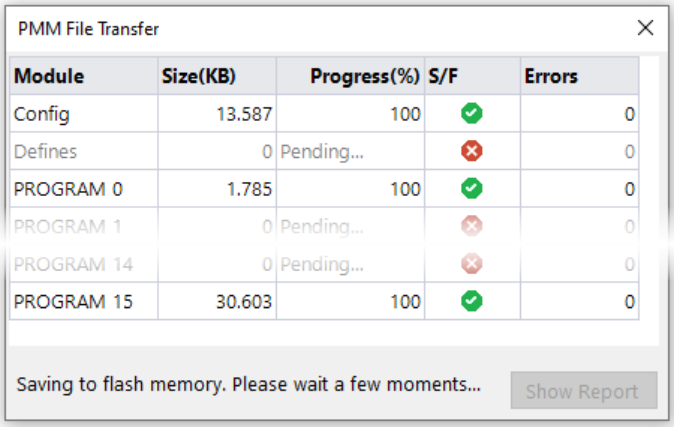
Step 2: Connect to the controller. See the previous section for details on establishing a connection.

Step 3: Click the download button in the Toolbar.

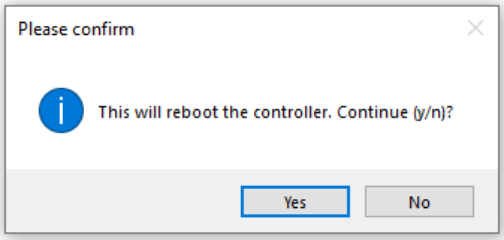
Step 4: The *Download Project to Controller* dialog box will appear. This dialog box allows the user to configure what parts of the project are downloaded to save time during development and troubleshooting. Users who are installing a replacement controller should configure the options like they are set below (all boxes checked, *Download Program(s)* pull-down set to *All Programs*). Click *OK*.

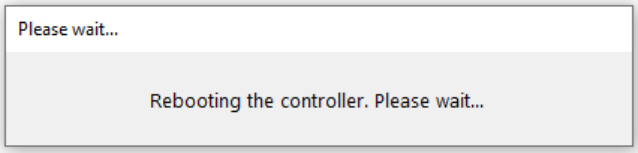



Step 5: Wait for the download to finish.

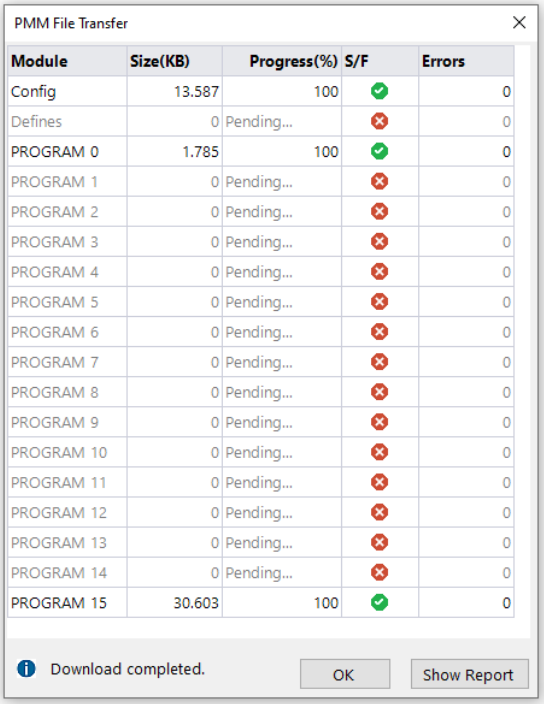


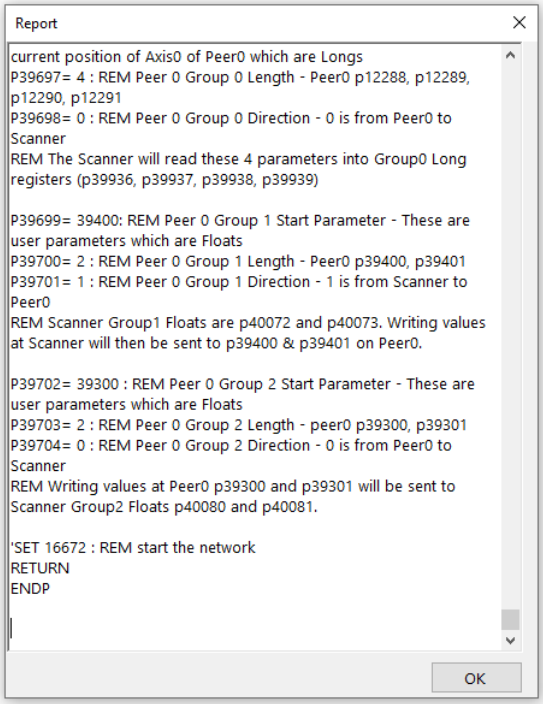

Step 7: A dialog will appear requesting that the controller be rebooted. Click *Yes* to reboot it, which will allow the new motor configurations (if applicable) to take effect.



Step 8: After the reboot is done, the PMM File Transfer dialog will reappear. Click *Show Report* to see what was downloaded. Errors will be highlighted in red.



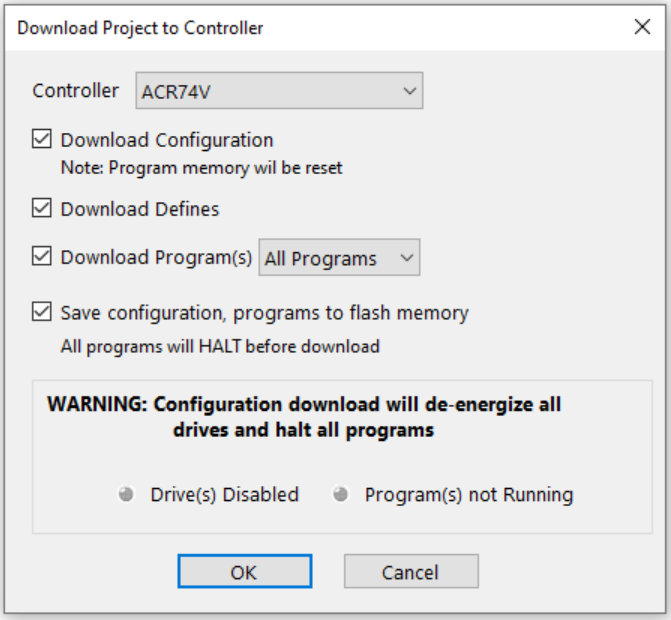

## Reference

Several dialogs shown in the Procedure section above deserve more explanation. This information is shown separately to keep the procedure light and easy to follow.

The Download Project to Controller dialog has several options for choosing what gets downloaded.

The *Controller* pull-down allows the user to select which controller is being targeted for download. This only affects projects with multiple controllers.

The *Download Configuration* check box selects whether the Configuration Wizard data (System Code) will be sent down during the download. It is sometimes helpful to uncheck this if minor tweaks are being made to a program. If the box is checked, existing programs and defines will be deleted, meaning new ones will need to be downloaded to take their place at some point.



The *Download Defines* check box selects whether defines are sent down.

The *Download Program(s)* check box selects whether programs are sent down. The associated pull-down selects which program to send down. The user can also select *All Programs*.

There are also indicator lights to show that the drives have been disabled and the programs have been halted.

**NOTE: Downloading to the controller will disable all drives and halt all programs.**

The PMM File Transfer Dialog also has several useful features.

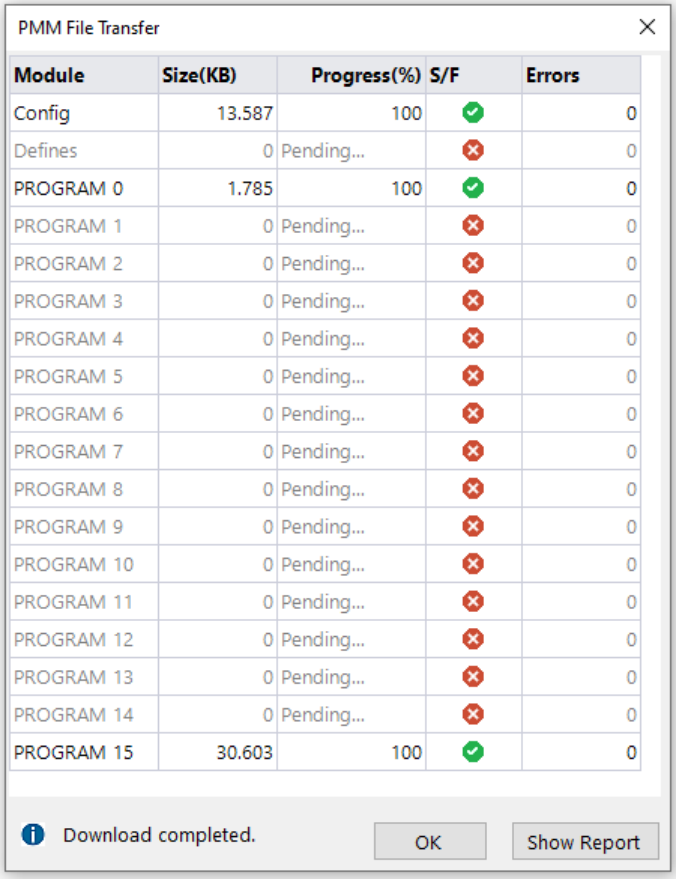
The *Module* column shows the various items scheduled for download, most of which are programs. Defines and configuration data are listed as well.

The *Size(KB)* column shows the size of the data.

The *Progress(%)* column will show *Pending…* for sections whose download has not started (or are not scheduled). It will otherwise show how much of a section has been download.

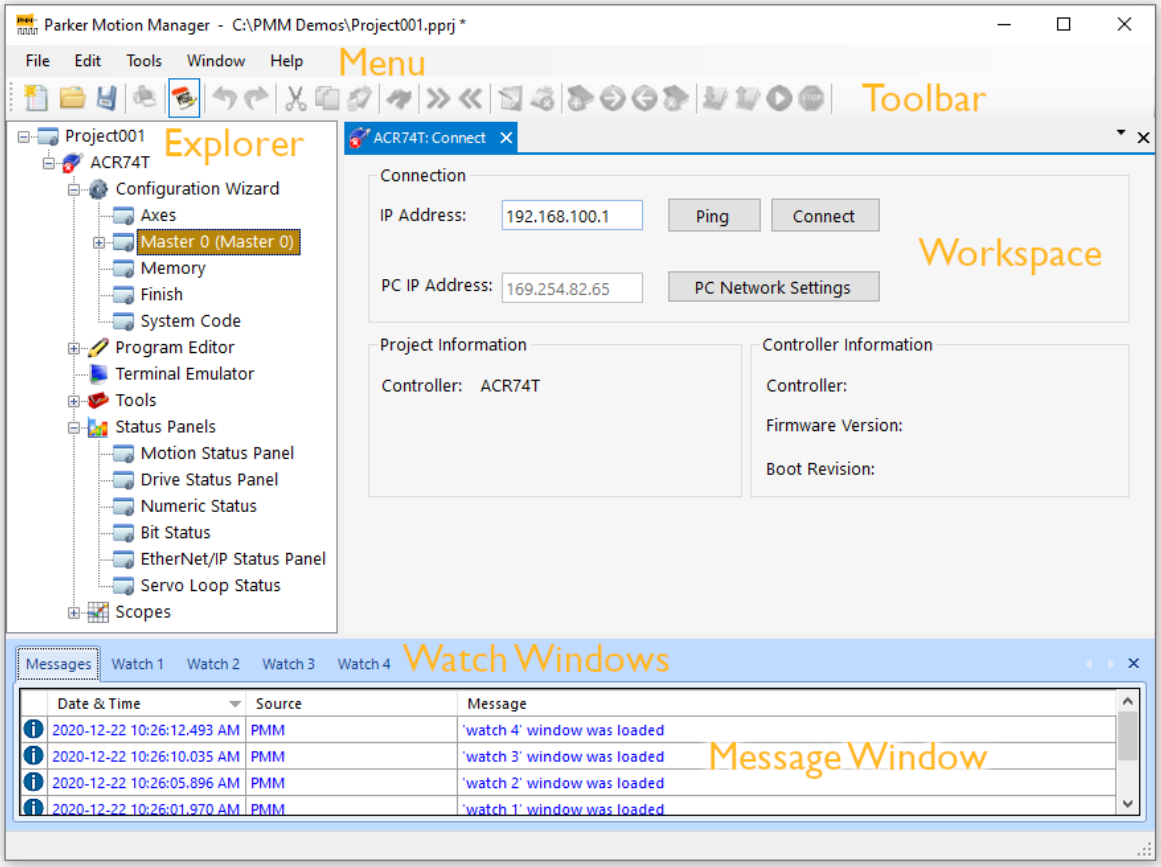
The *S/F* (success/failure) column shows a red "x" for sections that failed to download properly (or were never started). It shows a green check mark for sections that completed without errors.

The *Errors* column shows how many errors were present during download of a section. These can be AcroBASIC syntax errors, parameter range violations invalid options.




The *Show Report* button displays a full report of what was downloaded and highlights errors in red.

# Parker Motion Manager Parts




The main screen of PMM is divided the into seven different sections shown above: Menu, Toolbar, Explorer, Workspace, Message Window and Watch Window(s).

Each section is further explained in the following pages.

# Menu

The Menu bar provides quick access to common project management tools and options. Many of them are familiar to Windows users.

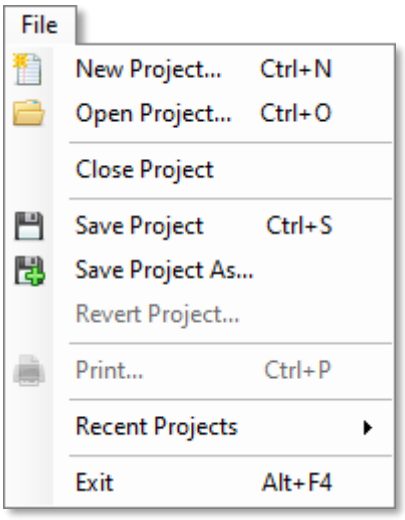
## File Menu

Manage project files. Most of these are standard Windows file management tools and are self-explanatory.

*Revert Project* reloads the project from the saved copy on the hard drive.

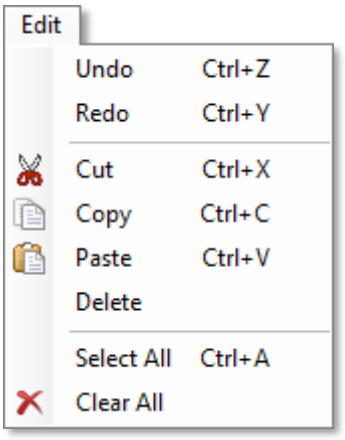
*Print* allows the user to print any text-based editor (like the Program Editor) to PDF or a printer.

*Recent Projects* provides a list of recently edited projects so that they can be quickly reopened.




## Edit Menu

Provides a few standard text editing tools, such as *Copy* and *Paste*. The tools in this menu are only usable in text-based editors like the Program Editor and textual fields like the ones in the Memory Configuration screen.



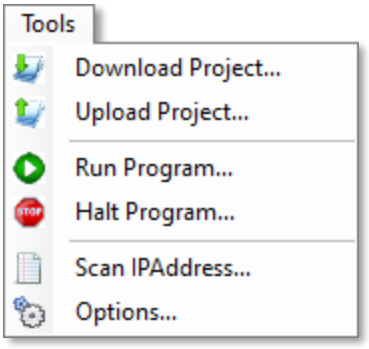

## Tools Menu

Provides certain tools specific to PMM.

*Download Project* can be used to quickly download an entire project to the controller.

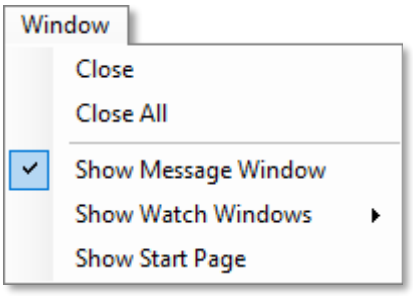
*Upload Project* can be used to upload a project from the controller into PMM, overwriting the current project data.

*Run Program* and *Halt Program* can be used to start or stop one or more programs.




Scan IP Address is useful for finding a controller whose IP address is unknown. More information about using this tool can be found in Finding an ACR with the Scan Tool.

## Window Menu

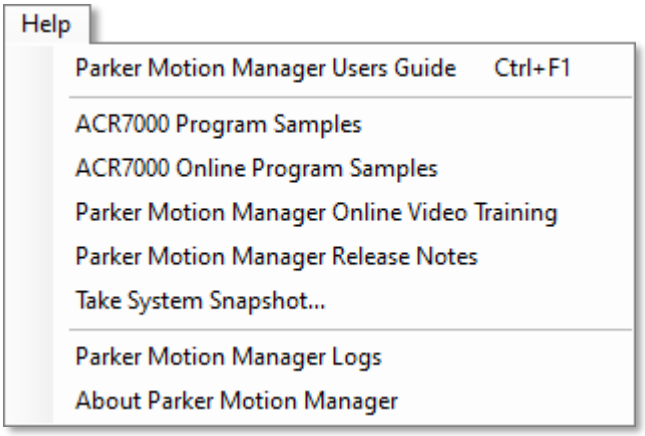



Provides options for managing internal PMM windows.

Close All is useful for quickly decluttering the screen.

*Show Message Window*, *Show Watch Windows* and *Show Start Page* all display their respective windows, which are covered later.

## Help Menu




Holds various help-related topics.

*Parker Motion Manager User's Guide* will open the help file, which is an indexed and searchable CHM file designed to allow for rapid look-up.

*Parker Motion Manager Release Notes* can help in case you believe you have encountered a software bug.

*Take System Snapshot* can help if you need to send data about your PC configuration to Parker engineers.

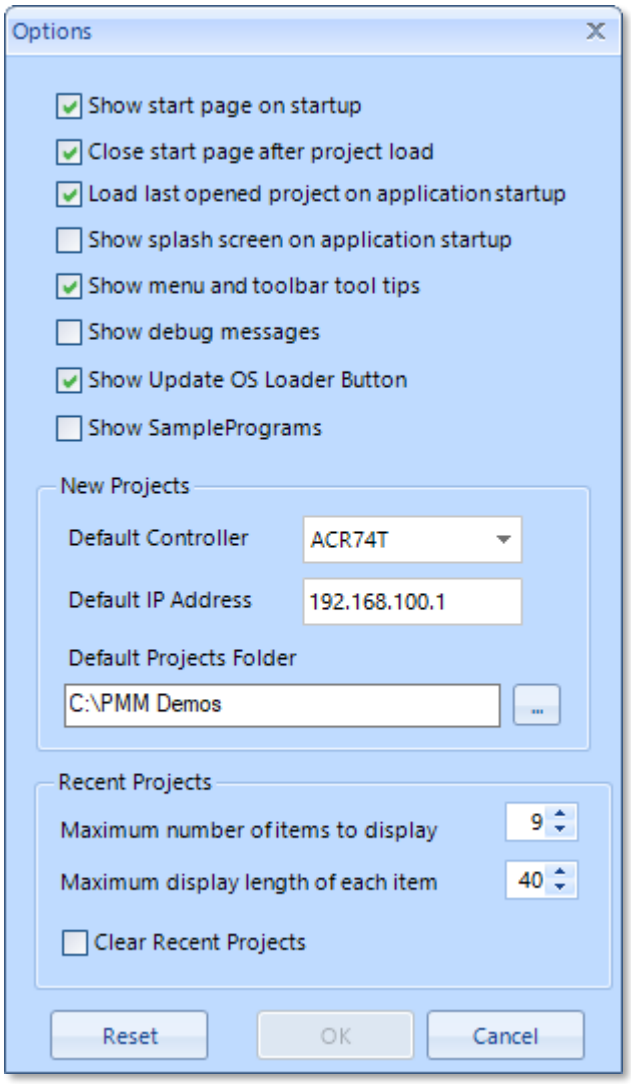
*About Parker Motion Manager* will display the version of PMM.

### Tools → Options

This dialog contains settings for PMM. Most of them are self-explanatory.

The *Show menu and toolbar tool tips* option is useful for new programmers who are not familiar with PMM yet.

The *Show debug messages* option can be helpful if you are using a pre-release build (unusual) or need to supply application crash information to Parker support.




# *Toolbar*

The Toolbar is where the most commonly used tools are kept. Each tool gets an icon and a tool-tip ("hover-over" text) that helps identify and describe it.

Icon availability changes automatically and depends upon the Workspace.

 Opens a dialog for creating a new project.

 Opens a dialog for selecting a previously saved project.


Saves the changes to the current project.


Opens a print dialog for sending the current text to a printer (including print-to-PDF).


Displays/hides the Message Window.


Undoes the last change made to the text in the active control.


Redoes the last change made to the text in the active control.


Cuts the selected text to the Clipboard.


Copies the selected text to the Clipboard.


Pastes the selected text from the Clipboard.


Opens the Find/Replace dialog for finding and/or replacing the current text.


Increases the indentation of the selected line in the active program editor.


Decreases the indentation of the selected line in the active program editor.


Comments the selected block of text in the active program editor.


Uncomments the selected block of text in the active program editor.


Adds a bookmark to selected line in the active program editor. Bookmarks are useful for quickly navigating very large programs.


Goes to the next bookmark in the active program editor.


Goes to the previous bookmark in the active program editor.


Removes a bookmark from the selected line in the active program editor.


Opens the Download Project to Controller dialog.


Opens the Upload Project from Controller dialog.


Opens the Starts Program(s) dialog.

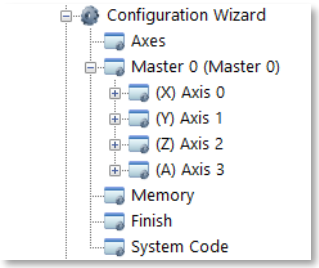
Opens the Halt Program(s) dialog.

# *Explorer*

The Explorer is divided into several sections.

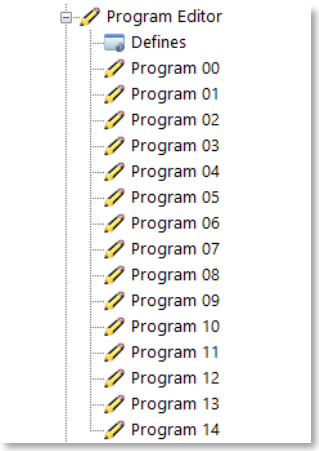

## Connection

These tools allow the user to rename the controller, add another controller to the project, delete a controller, connect to a controller or disconnect from a controller. The Connect screen also provides useful information about the controller, such as firmware level and model number.




## Configuration Wizard

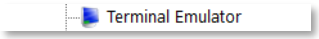
This wizard is a comprehensive start-to-finish configuration tool that sets up axis names, motor data, engineering units, I/O configuration, default move characteristics and more.




## Program Editor

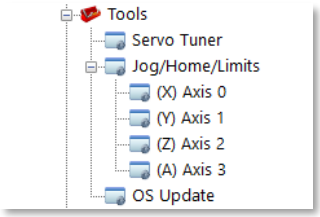
The various program editors are where programs are written.

The Defines editor is a tabular editor where a user can set up named aliases (known as "defines") for commonly used parameters. PMM's Defines editor provides a user-friendly experience by checking user input for validity.




## Terminal Emulator

This is one of the most useful tools in PMM and will receive detailed coverage later. The Terminal Emulator is used to send ASCII commands directly to the controller.

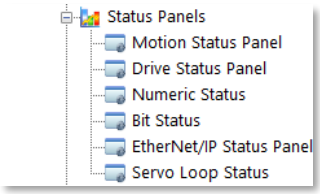



### Tools

The Servo Tuner is a graphical tuning tool that helps the user run test moves, evaluate performance and update gains. Not available for stepper axes.

The Jog/Home/Limits screen is used to test jog motion on each axis, allowing the user to quickly verify that the hardware and configuration are working correctly.

The OS Update tool is used to update to load a new OS onto the controller.



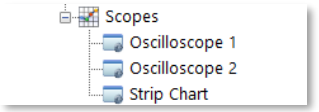

### Status Panels

The Motion Status Panel and Drive Status Panel are used to investigate faults and view position/speed data.

The Numeric Status and Bit Status panels are used to check the status of any bit or parameter in the controller.

The EtherNet/IP Status Panel displays connection and fault data specific to EtherNet/IP.

The Servo Loop Status panel visually connects the various position command registers on each axis with the output being generated to meet that position.

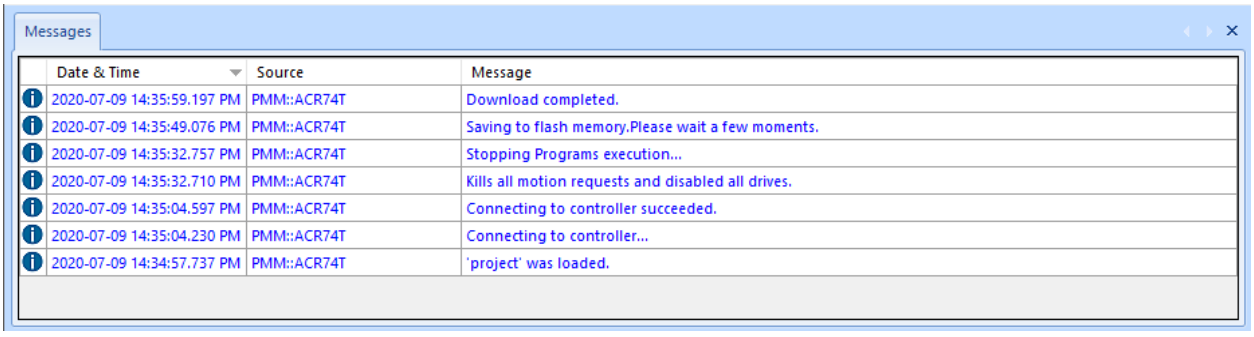



### Scopes

The scopes allow the user to graph any parameter in the controller to help evaluate performance or track down process issues.

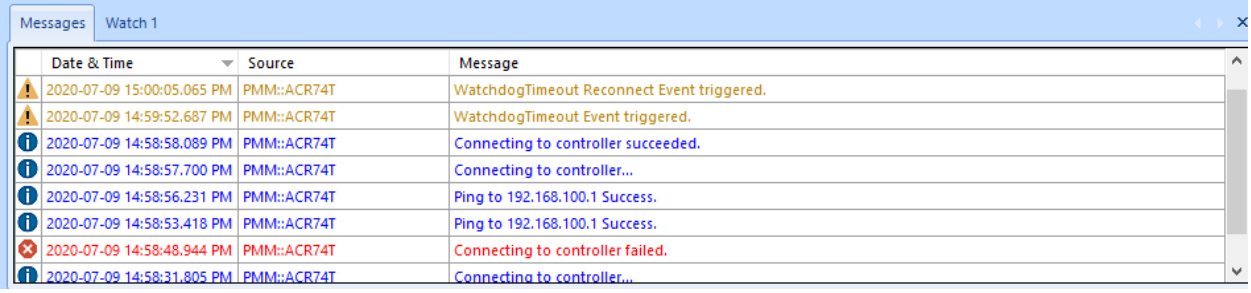
# *Message Window*

The Message Window provides status and error information while online with the controller. It also displays "housekeeping" messages from PMM when not online. It is recommended that it be kept open when online.

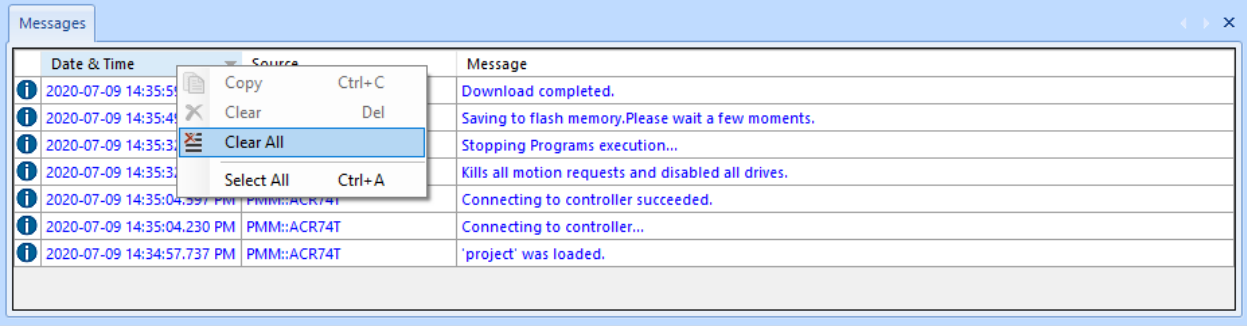



Informational messages appear in blue, warnings appear in yellow and errors appear in red. The Messages Window is particularly useful when troubleshooting connection issues with the controller. Notice the two warnings shown at the top of the image below:

- WatchdogTimeout Reconnect Event triggered. This means that PMM has previously lost its connection to the controller but is attempting to reestablish it. This happens when the controller is rebooted after a download.
- WatchdogTimeout Event triggered. This is the warning that occurs immediately after PMM loses its connection to the controller. Again, this will occur during a reboot.



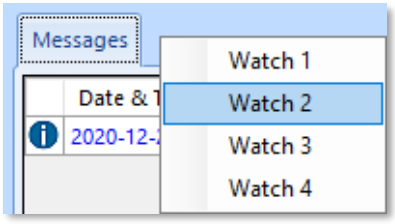

Click on column headers to sort messages by that column's data. Right-clicking the header presents an option to clear messages. The messages can also be selected and copied to a text file, email or spreadsheet program.

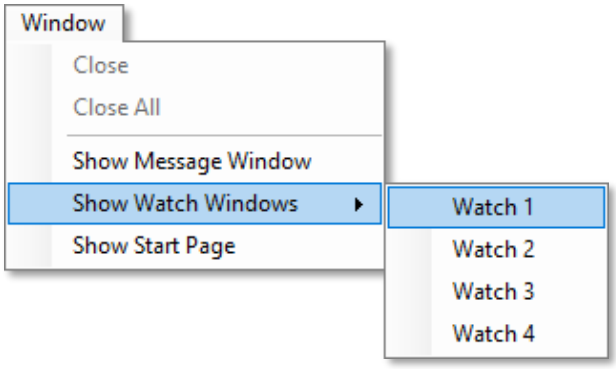



# *Watch Windows*

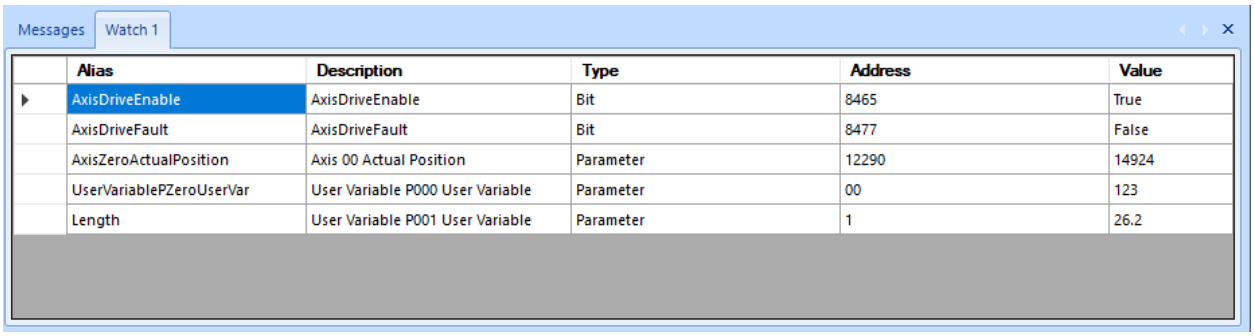
Watch windows let the user monitor bits and parameters in real time when PMM in online with the controller. Four watch windows are available. These are saved within the project. If the user closes the project and loads the project again, the watch windows are also loaded.

Right-click to the right of *Messages* to display watch windows or go to *Menu* → *Window* → *Show Watch Windows*.



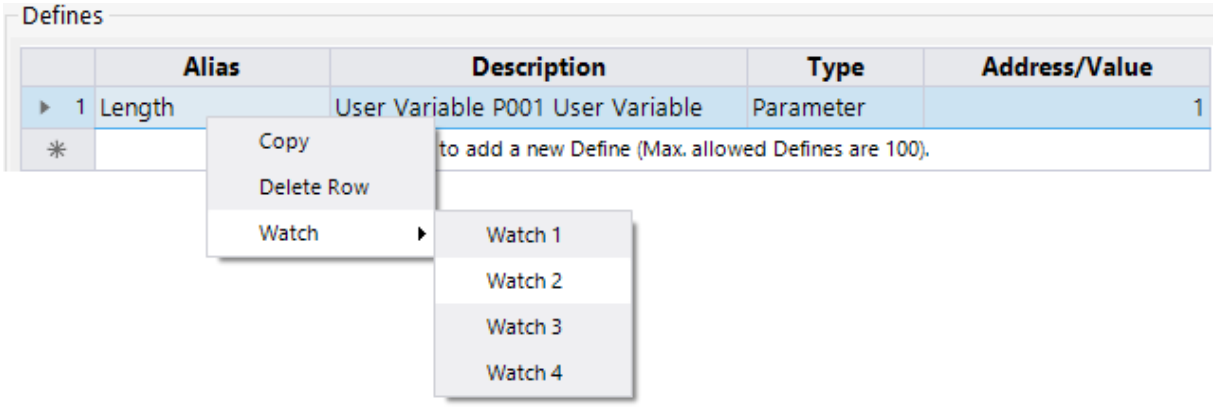



Each watch window can hold 20 rows of bits or parameters.

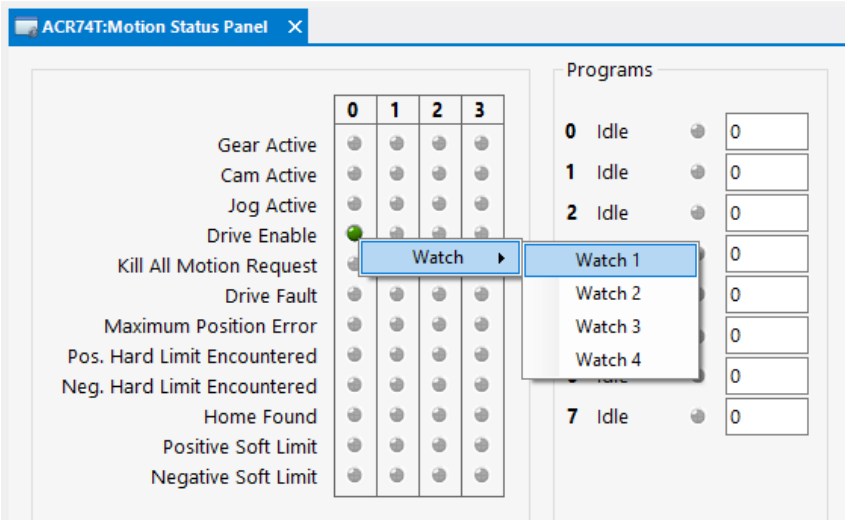



| Alias | Description | Type | Address | Value |
|---|---|---|---|---|
| AxisDriveEnable | AxisDriveEnable | Bit | 8465 | True |
| AxisDriveFault | AxisDriveFault | Bit | 8477 | False |
| AxisZeroActualPosition | Axis 00 Actual Position | Parameter | 12290 | 14924 |
| UserVariablePZeroUserVar | User Variable P000 User Variable | Parameter | 00 | 123 |
| Length | User Variable P001 User Variable | Parameter | 1 | 26.2 |

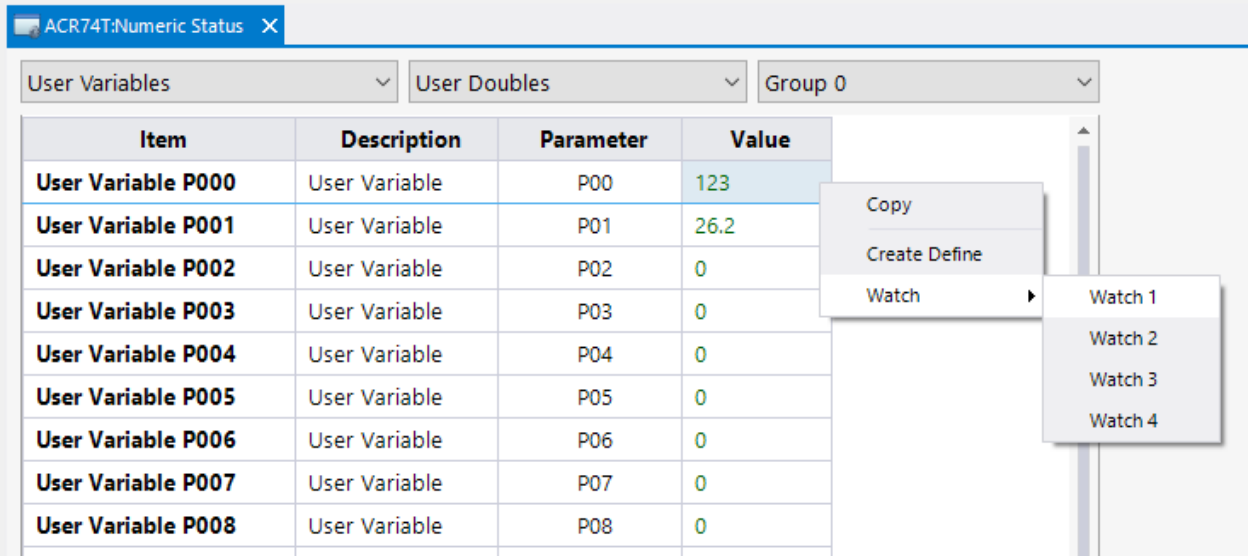
To add defines to a watch window, go to *Program Editor* → *Defines* and right-click the define you want to add. Then, click *Watch* and select the watch window that should display the define.



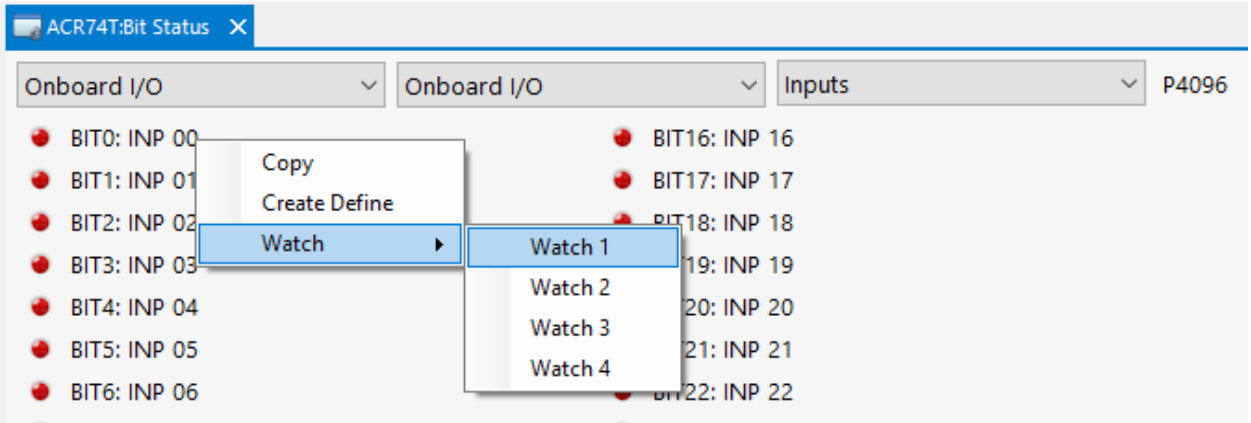

Bits and parameters can also be added to a watch window from the Motion Status Panel or Drive Status Panel. To add a bit or parameter, right-click on the indicator of interest and select a watch window:




This feature is also supported on the Numeric Status panel.




The Bit Status panel can also be used to populate watch windows.



# Configuration Wizard

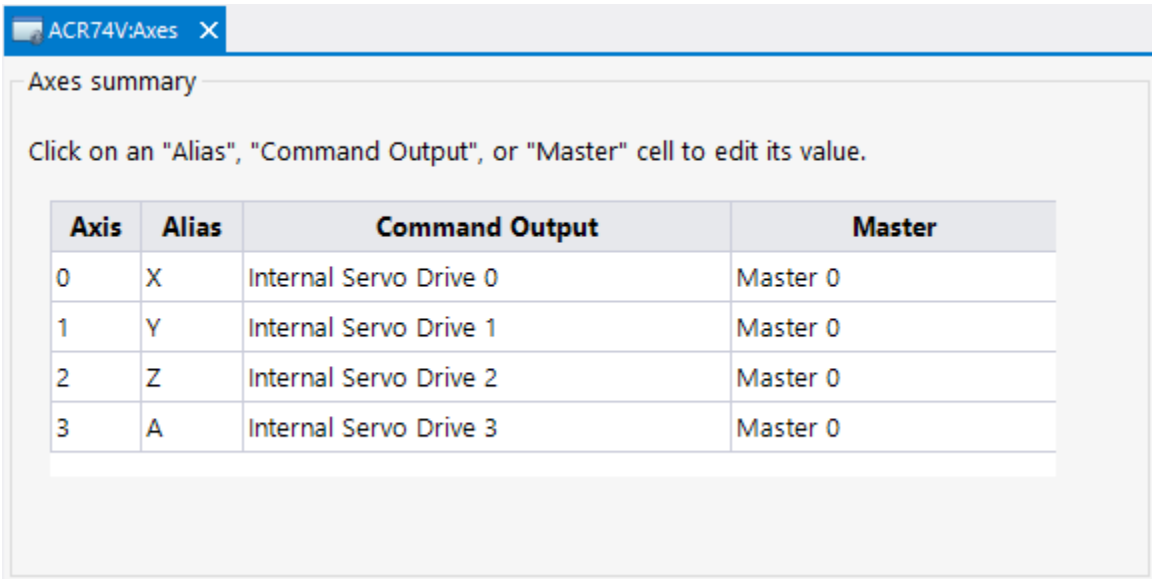
The Configuration Wizard steps users through setting the controller for the motor or drive type, scaling, limit/home sensor assignment and fault settings for each axis. These will be specific to the controller type. An ACR74T integrated stepper will have four axes and need to know the step motor part number, whereas an ACR78C controller will have eight axes and need to know the drive types (servo or stepper) and feedback resolution for servos or closed-loop steppers. The ACR74V integrated servo will need to know the servo motor type for each axis while the IPA is a single-axis servo. All screens in the Configuration Wizard have three common buttons located at the bottom:

- *Previous* and *Next* allow the user to navigate back and forth through the Configuration Wizard.
- *Reset to Default* sets all parameters on the current screen back to their default values.

**NOTE: The information presented here is referential in nature. For an example of setting up an ACR for the first time, see the Parker Motion Manager Quick Start Guide.**

## *Axes*

The first item in the Configuration Wizard is the Axes screen, used to create basic functional groupings of axes.

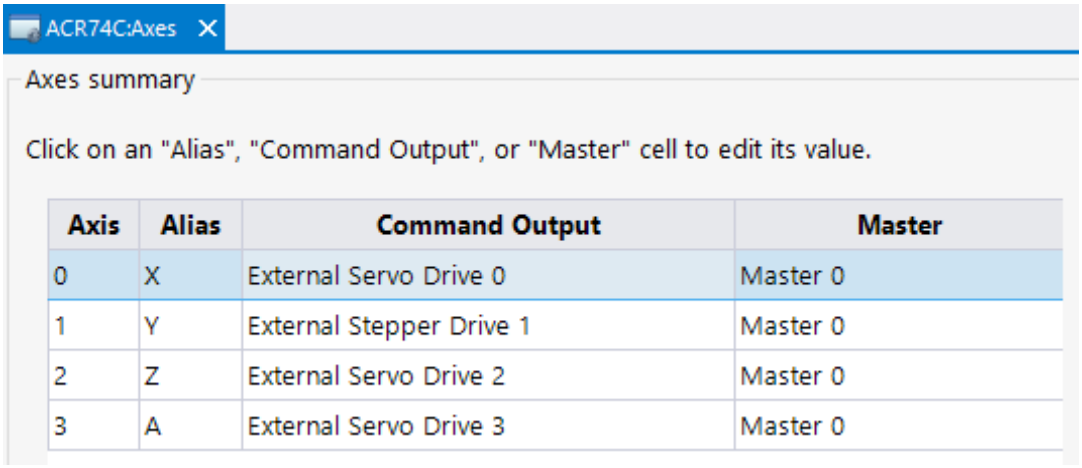



Each axis has an alias, a short alphabetical name up to four letters long that can be used to command it to move (`X25 Y4`), enable (`DRIVE ON X`), reset (`RES X`) or perform other tasks. Numbers and special characters are not permitted in axis aliases—only letters are allowed. The defaults are:

- Axis 0 is X.
- Axis 2 is Y.
- Axis 2 is Z.
- Axis 3 is A.
- Axis 4 is B.
- Axis 5 is C.
- Axis 6 is U.
- Axis 7 is V.

Users can rename the axes if they prefer. The names are only used for identifying the axes and to not ascribe any specific motion properties. For instance, an axis called U is not required to be a rotary axis.

The Command Output specifies the output hardware that will be used for an axis. This feature is mainly included for the ACR7xC so that the user can select ±10 VDC output or step-and-direction output. For the ACR7xT, ACR7xV and IPA, the Command Output is fixed as the drive hardware is included in the product. Technically, the user is also permitted to change the order of axes—Axis 0 can be made to use the hardware typically reserved for Axis 3, for instance. This practice is discouraged and is permitted mainly to provide contingencies in the event of hardware failure (swapping an axis to get a machine running).




| Axis | Alias | Command Output | Master |
|---|---|---|---|
| 0 | X | External Servo Drive 0 | Master 0 |
| 1 | Y | External Stepper Drive 1 | Master 0 |
| 2 | Z | External Servo Drive 2 | Master 0 |
| 3 | A | External Servo Drive 3 | Master 0 |

Each axis is assigned to a master which is a motion trajectory calculator. A master is attached to a program (Program 0 for Master 0, Program 1 for Master 1, etc.). Each axis must be assigned to a master. By default, they are all assigned to Master 0. The number of masters available is equal to the number of axes available on the product and is never more than eight.

Interpolated motion between multiple axes requires that they be attached to the same master. Interpolated motion refers to path-based motion that requires more than simple sequencing. The chart below helps explain what types of motion are considered "interpolated" in the ACR.

Interpolated:

- Diagonal lines.
- Circles/arcs.
- Splines.
- Smoothed paths (look-ahead).
- Modulo motion.

Not Interpolated:

- Jogging.
- Gearing.
- Camming.
- Single-axis moves (via jog profiler).
- Homing (via jog profiler).

Each axis can also be commanded to be moved separately, as is the case with jog moves.

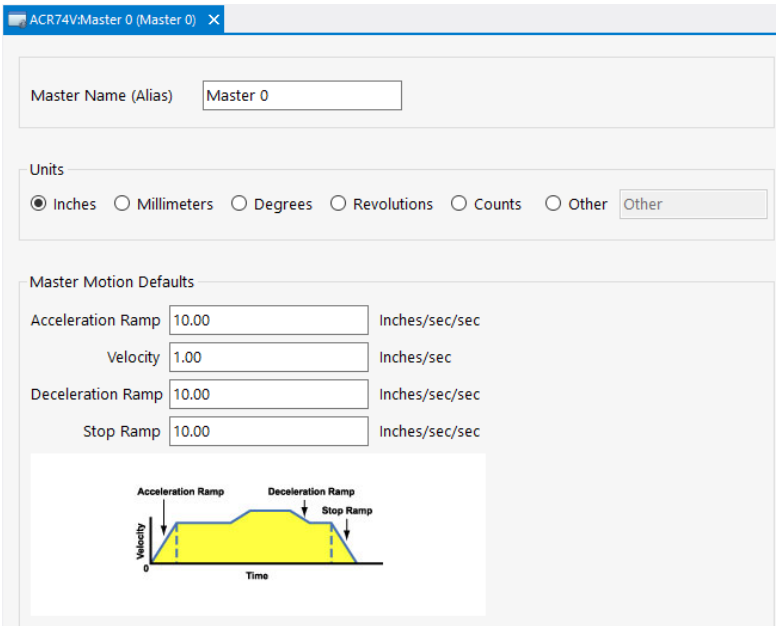
Some machines have multiple functional axis groups. For example, a machine might have one cartesian system that performs a stacking process and another that handles an inspection process. Multiaxis ACR controllers are capable of handling separate axis groups, a task best accomplished using multiple masters. Assigning a set of axes to Master 0 allows for interpolated motion on those axes in Program 0—axes attached to Master 1 are similarly coordinated using Program 1 and so on.

In the system shown above, the two subsystems would be attached to different masters, like Master 0 and Master 1. This would allow the controller to tightly coordinate moves within each subsystem, but would also allow each subsystem to maintain its independence from the other.

The advantage to using interpolated motion on a master is that it is easy to make a move on several axes start and stop at the same time, following exactly the path that the user needs. This can save process time and fulfill certain machine goals (e.g. "move the product in a circle").

# *Master (Units)*

The second screen in the Configuration Wizard is the Master screen. Here the units for the master can be selected. Users can use inches, millimeters, degrees, revolutions, encoder counts or specify their own. The units selected here are used throughout the rest of the configuration and in AcroBASIC programs. Selecting *Inches*, *Millimeters*, *Degrees* or *Revolutions* offers advantages on the Scaling screen later in the configuration. Selecting *Counts* amounts to no scaling, as all moves will use encoder or stepper counts as their unit of measure. Selecting *Other* provides no special features—this selection should only be used in special circumstances.




Users can also set default values for velocity, acceleration, deceleration and stop ramps for interpolated motion. The stop ramp is used when stopping motion with interpolated moves. The provided diagram explains how the

selected dynamics affect the motion profile. Note that the motion defaults can be changed at any time during program execution, so getting them right here is not critical.

The Master alias can be changed to name the group of axes. This name is only used within PMM for documentation purposes and is not used in the AcroBASIC program, nor is it stored in the controller.

# Drive/Motor

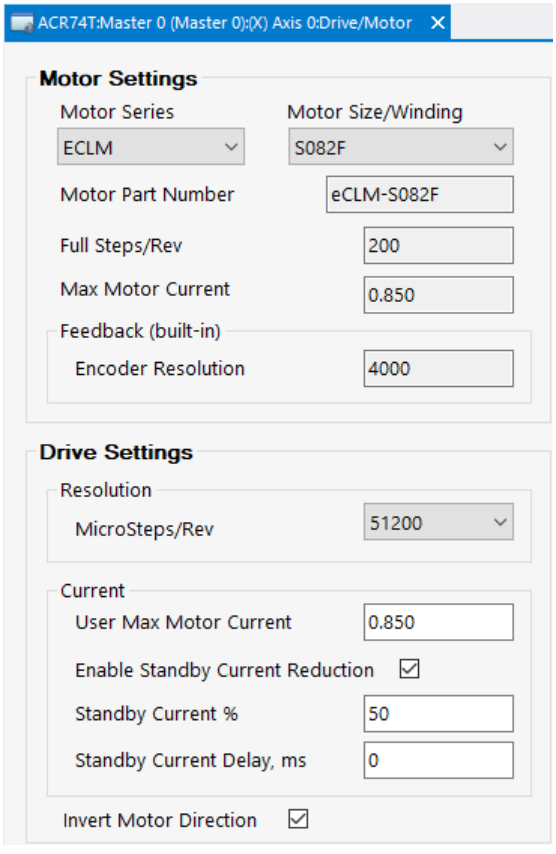
The Drive/Motor screen lets the user configure what motor is connected to the AC. In the case of the ACR7xC, the user instead configures parameters for the attached servo or stepper drive. This screen's display is highly dependent on the model of ACR in use, so it will be covered separately for each one.

New for PMM!

The Motor/Drive screen includes an Invert Motor Direction checkbox. This makes it easier to reverse the direction the user wants to be positive for the application. By default, clockwise looking at the motor shaft is positive for all rotary motors, except the P Series drives with P Series motors which are counterclockwise positive by default. For linear motors, the default positive direction is away from the cable exit on the coil.

# Drive/Motor (ACR7xT Stepper)

This screen allows the user to configure the type of stepper motor connected to this axis.




## Motor Settings

For Parker motors, use the *Motor Series* and *Motor Size/Winding* pull-downs to select the type of motor in use. The motor's model number can be found on its product label. This will populate all the other data in the Motor Setting subpanel.

For third-party motors, select *Other* from *Motor Series*. Three fields are available for editing:

- The *Motor Part Number* field is for documentation purposes only.
- The *Full Steps/Rev* field should usually be set to 200 which corresponds to a 1.8° step motor (50 cycle/rev). Set this field to 400 for motors with 0.9° steps (100 cycle/rev).
- *Max Motor Current* is the published current rating for the motor in amps peak-of-sine.

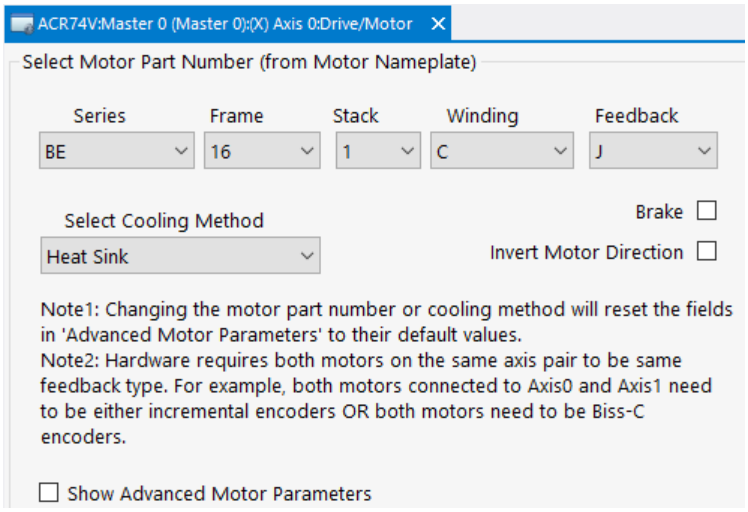
## Drive Settings

There are several fields here. The defaults are fine for most applications, but they are explained below for clarity.

- The *MicroSteps/Rev* pull-down allows the user to configure how fine the smallest possible step is for this axis. There is usually never a reason to set it to any value other than the default of 51,200, which is the highest setting.
- The *User Max Motor Current* field allows the user to reduce current to the motor. Lowering this value can help keep the motor cool but will also limit the maximum available torque.
- The *Enable Standby Current Reduction* check box allows the drive to reduce the current it delivers to the motor when it is not moving. Using this check box reduces motor heating during standstill periods, but also limits the available holding torque.
- The *Standby Current %* field allows the user to configure how much current will be delivered when standby current reduction is enabled. The default is 50%, which is typically sufficient.
- The *Standby Current Delay, ms* field is the length of time between the end of motion and the start of standby current reduction. The default is 0 ms. It can help to increase this time if the stepper motor appears to "slip" a little at the tail end of a move.
- The *Invert Motor Direction* checkbox switches the direction of positive motion for the axis, useful if the default positive direction is found to be going the wrong way.

# *Drive/Motor (ACR7xV Servo or IPA)*

This screen allows the user to configure the type of servo motor connected to this axis.




Use the *Series*, *Frame*, *Stack*, *Winding* and *Feedback* pull-downs to configure the model of motor used on this axis. The motor model number should be printed on the label on the side of the motor. This will set motor parameters such as rated current, encoder resolution, torque constant and many others. For non-Parker motors, set the *Series* to *Other* (more on this below).

The *Invert Motor Direction* checkbox switches the direction of positive motion for the axis, useful if the default positive direction is found to be going the wrong way.

The *Brake* checkbox should be checked for motors with a built-in failsafe brake. The ACR7xV has built-in brake supplies tapped from the internal 24 VDC control power. The IPA has a dry contact brake relay that requires an external 24 VDC supply.

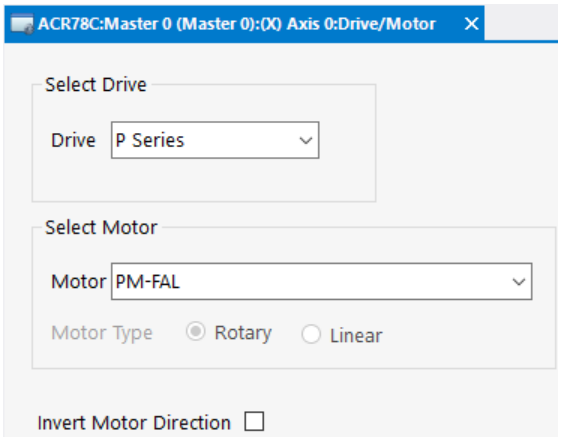
The *Select Cooling Method* pull-down can be set to *Heat Sink* or *No Heat Sink*. For motors that will mount to a metal actuator, like an electric cylinder or ballscrew table, this should be set to *Heat Sink*. For motors that will be left in open air, it is better to select *No Heat Sink*. For motors that will be connected to gearheads run at a high duty cycle, *No Heat Sink* is also a more appropriate option since gearheads can generate significant heat on their own. The *No Heat Sink* option reduces the torque rating of the motor slightly to help prevent it from overheating.

The *Show Advanced Motor Parameters* checkbox activates a hidden screen that is not normally needed when using a Parker motor—it is unchecked by default. However, it should be checked whenever a third-party motor or

Parker kit motor is being used. It should also be checked for Parker linear motors purchased without mechanics (i.e. Parker did not provide the encoder and bearings).

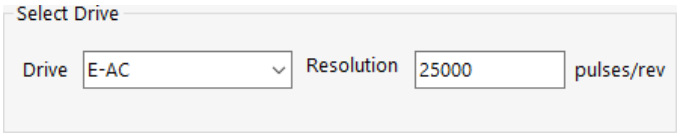
There are two notes on this screen and the second one bears clarification. For the ACR7xV, the axes are broken into pairs. Axis 0 and Axis 1 are on the first power board, Axis 2 and Axis 3 are on the second power board and so on. Each power board can only support one type of feedback. If a motor with BiSS-C feedback is selected for Axis 0, so must one be selected for Axis 1. If incremental feedback is needed on Axis 3, then Axis 2 will need to use a motor with incremental feedback. Users should plan accordingly when selecting motors.

# *Drive/Motor (ACR7xC)*




This screen allows the user configure the type of servo/stepper drive and motor connected to this axis.

The *Drive* pull-down is used to select the model of Parker servo or stepper drive connected to this axis. Available drives will be filtered depending on whether the user configured this axis for stepper or servo output. All of Parker's currently offered compatible drives are listed as well as several legacy models. For third-party drives, select *Other*. Note that any axis set up for servo output must be connected to a servo drive configured for ±10 VDC analog torque control. Some Parker servo drives are listed for both servo and stepper output because they support ±10 VDC torque control as well as step-and-direction.



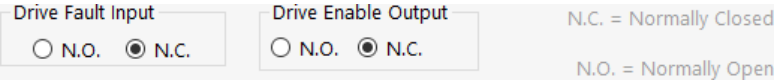

If the axis is configured as a stepper axis, a *Resolution* field will be made available next to the drive selection. This must be filled out with the drive's command resolution. Selecting a Parker drive populates the resolution with the default for that drive. It is important to double-check the resolution setting on drive to make sure the correct value is entered in this field.

The *Motor* pull-down is used to select the motor for this axis. The main purpose for doing this is to configure the encoder resolution, used in scaling calculations. Both rotary and linear motors from Parker are listed. These motors are filtered based on their compatibility with the selected drive. For third-party motors, select *Other*.

The *Motor Type* radio selector can be set to *Rotary* for rotary motors or *Linear* for linear motors. This helps with automatic scaling calculations later. If a Parker motor and drive have been selected, this option is grayed out. If *Motor* is set to *Other*, the user needs to select *Rotary* or *Linear* as appropriate.

The *Invert Motor Direction* checkbox switches the direction of positive motion for the axis, useful if the default positive direction is found to be going the wrong way.

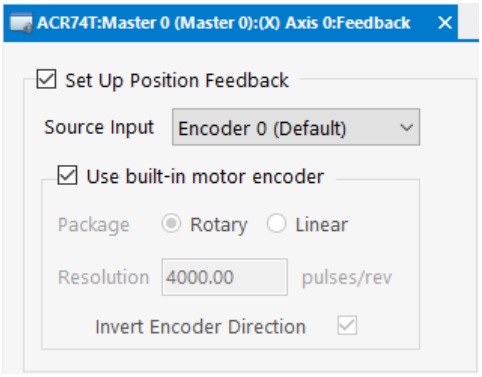
If *Drive* is set to *Other*, two additional radio selectors will become visible. These configure axis I/O for normally open or normally closed operation. The *Drive Fault Input* radio selector configures the input used by the drive to report a fault. On ACR7xC axis connectors, this input is on pins 16 and 17. The *Drive Enable Output* configures the output used by the controller to enable the drive. On ACR7xC axis connectors, this output is on pins 20 and 21.



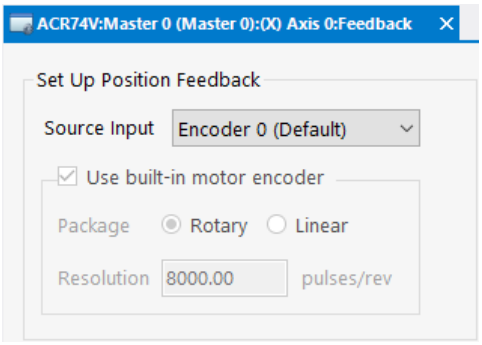

Consult the drive's documentation to know which is appropriate.

# Feedback

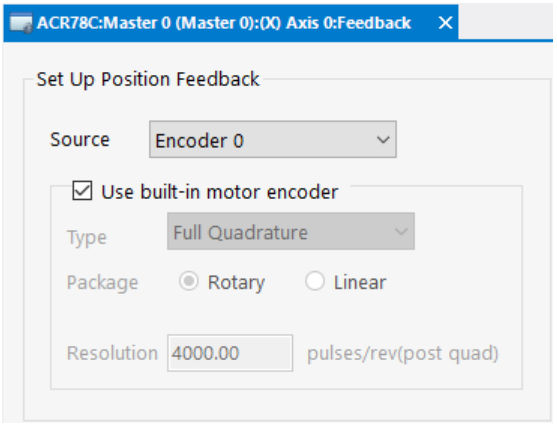
The ACR7000 and IPA controllers support several standard feedback types. The Feedback screen allows the user to fine-tune their configuration. Each controller type has its own version of the screen, but there are a few common tools:




- Set Up Position Feedback. This option is only present for stepper axes. Stepper axes do not require position feedback, but support it as an option. Unchecking this box turns feedback off.
- Source Input. This selects which encoder will be used for feedback on this axis. ACR74T and ACR7xC units allow the user to select any encoder except the auxiliary encoder. ACR7xV units only allow the user to select one of the two encoders on the same power board as this axis. So, Axis 0 can use either Encoder 0 or Encoder 1 for feedback.
- Use built-in motor encoder. If this option is selected, all options below it are grayed out and default values are used from the motor file.
- Type. This option is only available for the ACR7xC. It allows the user to select either quadrature (default) encoder feedback or SSI (Serial Synchronous Interface). The SSI option is useful for interfacing with certain kinds of position sensors, including devices that are not traditional encoders. It is also useful when interfacing with an Aries AE/SE servo drive for legacy machine upgrades.

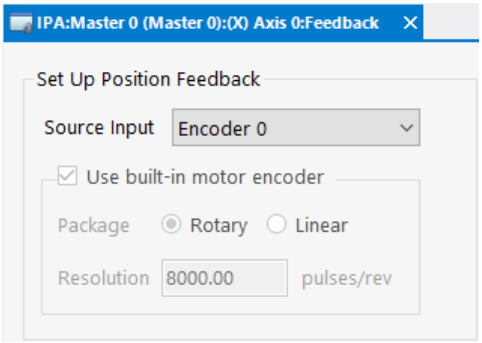



- Package. The user can select Rotary or Linear. This helps PMM work out scaling by determining which set of units apply.
- Resolution. Encoder resolution in counts/rev for rotary encoders or counts/mm for linear encoders.
- Invert Encoder Direction. This option is only available for the ACR7xC. It allows the user to change the positive direction (polarity) of the encoder. This is useful in cases where the command and feedback signal polarities do not match due to system design.




The ACR74T integrated stepper has optional quadrature encoder inputs for each axis. Open loop steppers (step motors without encoders) are supported as well as closed-loop steppers. The eCL series closed-loop step motors' encoder extension cables connect to these inputs, resulting in a plug-and-play solution. Selecting an eCL motor sets the motor settings on the previous screen and

the encoder resolution under Feedback. Linear quadrature encoders, such as with Parker precision stages, are also supported.



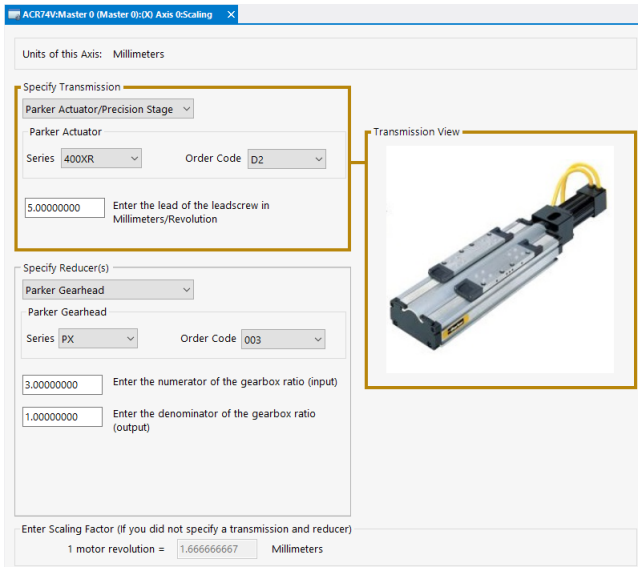

The ACR7xV integrated servo requires encoder feedback for closed-loop servo control. If a Parker servo motor was selected on the Drive/Motor screen, the feedback screen will have already been set based on the motor specifications.

The ACR7xC standalone controller supports stepper and servo axes. Servo axes will produce a ±10 VDC analog signal to an external servo amplifier and read its encoder feedback. The ACR7xC supports both standard quadrature feedback (rotary or linear) and SSI. The auxiliary encoder only supports quadrature. By selecting the motor type on the Drive/Motor screen, the Feedback will have already been set.

The ACR7xC also supports stepper drives and servo drives in step-and-direction mode. In this mode, the drive is closing the position loop. The controller does not require feedback but can read the encoder if connected. Position Maintenance is available for end-of-move corrections for stepper systems and can use rotary or linear feedback.

# *Scaling*

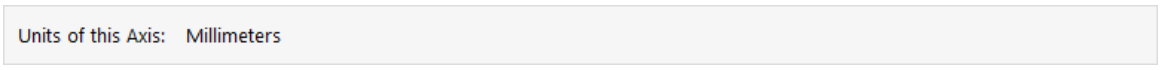
The Scaling screen allows users to define a relationship between encoder or stepper counts and engineering units. The controller needs to know how many counts are in one inch, millimeter, degree, revolution or whatever other unit is in use. Instead of requiring the user to perform this calculation, PMM provides an easy way to configure the unit scale based on easily found data about the components in use.




New for PMM!

Parker actuators, precision stages and gearheads have been added. Use the part number marked on the products to set the order code in PMM. Configuring the Parker mechanics in use will automatically import the correct scaling factors, like gear ratio or screw pitch. Generic screw, belt, chain and gear elements are still available to support non-Parker mechanics.

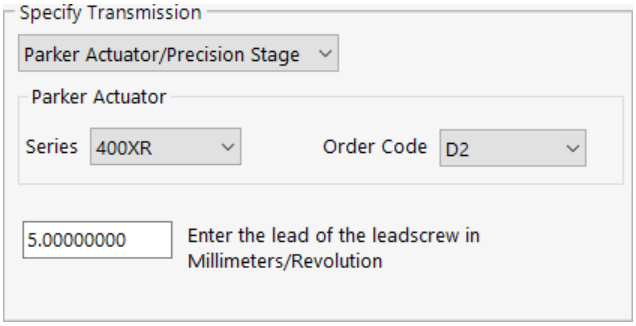
The top subpanel shows the units currently in use on this axis.




**NOTE: The tools on this screen pertain to setting up rotary motors. For linear motors, leave all options at default (*None*) and click *Next*.**

## Specify Transmission

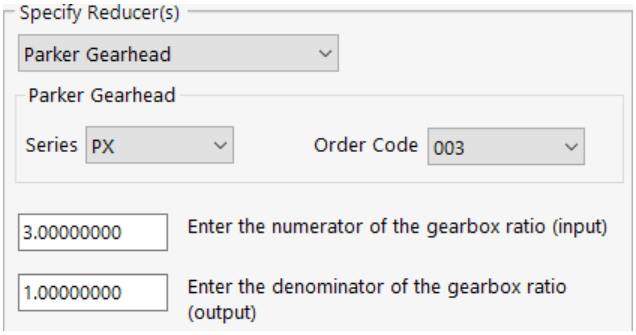
This subpanel allows the user to select the type of linear or rotary mechanics in use. Parker mechanics are available by selecting *Parker Actuator/Precision Stage* or *Parker Rodless Actuator* from the top pull-down menu. Following that, the type of actuator can be selected using the *Series* pull-down. Most Parker actuators have multiple drive train options, which are specified in the model number. The Order Code pull-down can be used to choose from the available options.




If a Parker actuator is not selected, the option is provided to enter the lead of the screw or diameter of the roller in the provided field. This field is filled automatically if a Parker actuator is selected.

## Specify Reducer(s)

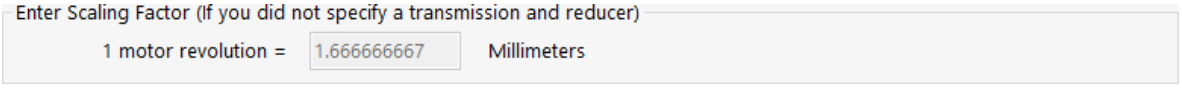
This subpanel works like the previous one, but is intended to be used for gearheads. The top pull-down allows the user to select whether a Parker gearhead is being used or some other gearing system. If a Parker gearhead is in use, the *Series* pull-down can be set to any currently available model family from Parker. The Order Code pull-down is used to select the gear ratio, which will be printed on the product label.




Non-Parker reducers are also supported. Users can choose a custom gearbox and enter the ratio manually. Other less common reduction options are also supported, such as pulleys and chains. Users can enter pulley diameters or tooth counts to configure the scale.

## Enter Scaling Factor

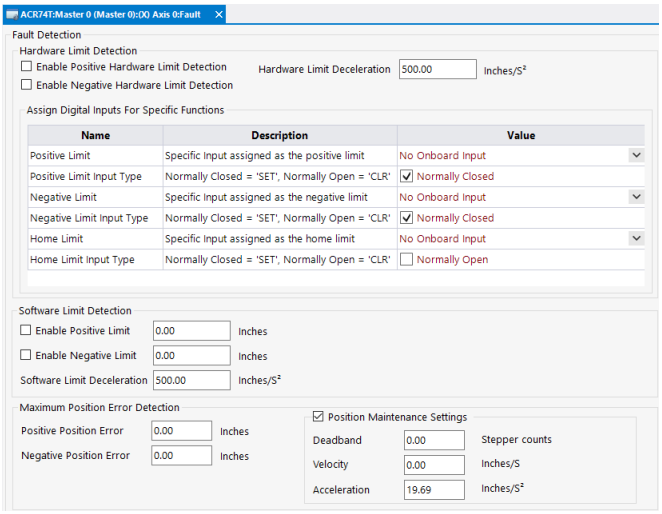
This subpanel displays the ratio of motor revolutions to linear or rotary output units. It normally does not permit changes, but will allow the user to enter any number if *None* is selected as the transmission and reducer.




The axis Scaling screen allows users to set the number of motor revolutions for their units. Predefined Transmissions and Reducers help calculate the scaling.

# Fault

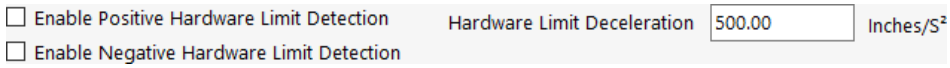
The Fault screen allows the user to set which inputs are connected to the end-of-travel limit sensors and whether they are normally closed or normally open. When a limit sensor is encountered, further motion in that direction is prevented but the drive is not disabled. Motion in the opposite direction can be commanded. End-of-travel sensors should be normally closed in case the sensor fails. For example, if its cable is cut or its connection comes loose, the sensor would fail open, faulting the axis and stopping motion.




## Hardware Limit Detection

The check boxes *Enable Positive Hardware Limit Detection* and *Enable Negative Hardware Limit Detection* are used to turn on limit checking. Without these checked, the axis will not respond to the limit sensors. They are unchecked by default since not all applications need or use limit sensors.

The *Hardware Limit Deceleration* is the stop rate when the controller encounters a limit sensor. Make sure this is high enough to stop the motor/stage before the actual hard-stop is hit.




## Assign Digital Inputs for Specific Functions

This table is where specific digital inputs are assigned as limits and home. Any onboard input can be used for any of the three functions.

New for PMM!

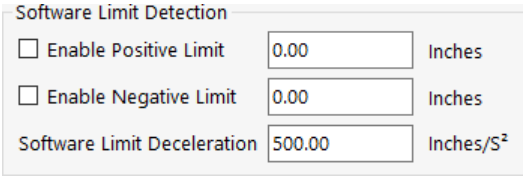
The positive, negative and home inputs can be assigned to any input and are no longer required to be consecutive.

| Name | Description | Value |
|---|---|---|
| Positive Limit | Specific Input assigned as the positive limit | No Onboard Input |
| Positive Limit Input Type | Normally Closed = 'SET', Normally Open = 'CLR' | ☑ Normally Closed |
| Negative Limit | Specific Input assigned as the negative limit | No Onboard Input |
| Negative Limit Input Type | Normally Closed = 'SET', Normally Open = 'CLR' | ☑ Normally Closed |
| Home Limit | Specific Input assigned as the home limit | No Onboard Input |
| Home Limit Input Type | Normally Closed = 'SET', Normally Open = 'CLR' | ☐ Normally Open |

Configuration is done in the Value column. The *Positive Limit*, *Negative Limit* and *Home Limit* pull-downs (set to *No Onboard Input* in the image above) are used to select which input serves this limit/home function. The *Input Type* pull-downs can be used to select between *Normally Open* and *Normally Closed*. Most Parker mechanics that ship with limits have normally closed limit switches and a normally open home switch.

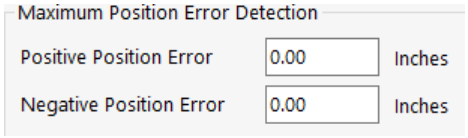
**NOTE: Normally open/normally closed is not the same distinction as sinking/sourcing or NPN/PNP. For more information about I/O, refer to the appropriate hardware manual.**

## Software Limit Detection




Software limits can be used to limit travel range. If the travel range is exceeded, the axis will be brought to a controlled stop. This is especially useful in systems that use absolute encoders but do not have limit switches. The check boxes *Enable Positive Limit* and *Enable Negative Limit* turn on soft limit checking. The associated fields set the distance from zero in user units at which the software limits will stop motion. The *Software Limit Deceleration* field specifies the deceleration ramp that will be applied if the limits are violated.

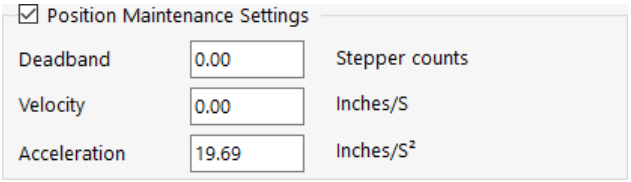
## Maximum Position Error Detection




Maximum Position Error is the maximum allowable error between the commanded position and the actual encoder position. This is required for a servo axis. This is not active for an open-loop stepper. It can be used for a closed-loop stepper or servo in stepper mode with the encoder connected. When the position error limit is violated, the controller assumes it has lost control of the motor and disables the drive. The position error limits should be set large enough that the axis does not generate nuisance faults during normal operation. They should also be set small enough that the drive cannot cause the motor to "run away" in the event of catastrophic failure. The *Positive Position Error* and *Negative Position Error* fields allow the user to set different limits in each direction if desired. Note that the negative limit must be entered as a negative number.

## Position Maintenance Settings

New for PMM!



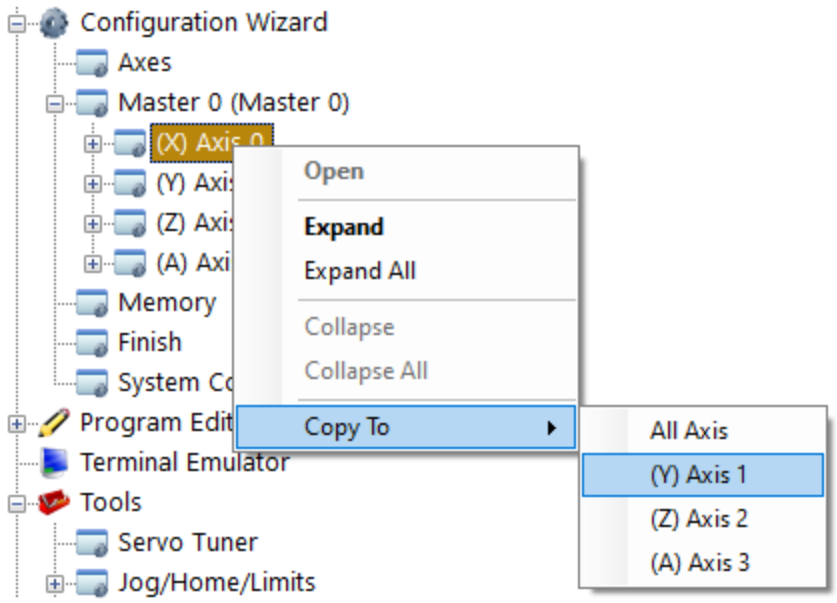

PMM allows Position Maintenance to be enabled for end-of-move corrections with move settings and deadband. This feature only applies to stepper axes with encoder feedback. When active, Position Maintenance tries to improve precision by issuing a small correction move after motion stops to account for any position error at the end of the move. This move is made automatically and is not shown to the user in the ACR's position command registers.

The *Position Maintenance Settings* check box can be left unchecked (default) if Position Maintenance is not desired or if there is no encoder available to support it. The fields within the subpanel configure move dynamics:

- *Deadband* configures the zone in which the axis is considered "settled". If an axis gets close enough to its intended destination that it is within the distance specified by *Deadband*, no corrective move is generated. Note that this field is in stepper counts, not user units.
- *Velocity* configures the maximum velocity of the correction move in user units. It is best to keep this value small to reduce the likelihood of a stall.
- *Acceleration* configures the acceleration and deceleration of the correction move in user units.

New for PMM!

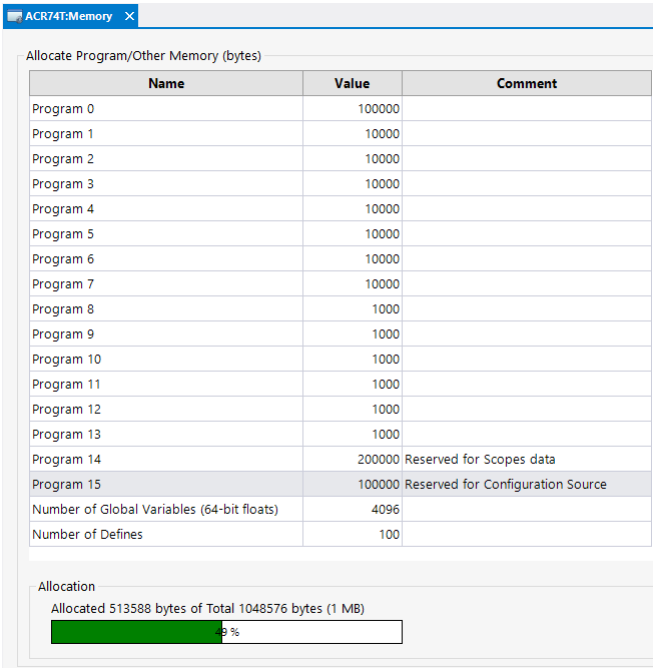
If the application calls for two axes that need to be set up identically, there is now an easy way to do it. Right-click the axis in the Explorer and click *Copy To*. Changes can be made after the copy is complete.




# *Memory*

This screen helps the user allocate the ACR's available memory to programs, global variables and Defines. The memory allocation (in bytes) can be altered in the *Value* column. The *Allocation* bar at the bottom of the screen shows how much memory is used and how much is free.

Memory for programs has already been allocated. Only if the programs are very large (notified on download) would these allocations need to be increased. Program memory is consumed by AcroBASIC code, locally dimensioned variables, program statements with returns (`GOSUB`, `IF`, `WHILE` and `FOR` statements) and interpolated moves (e.g. `X3 Y7`).




Program 14 has a large allocation for the onboard data capture for scopes. The Configuration Wizard settings are stored within Program 15, which has a fixed allocation.

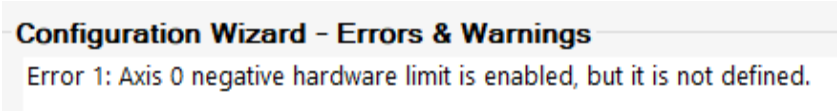
By default, the number of global variables is 4096 (P0—P4095). These are 64-bit floating point values available to the user for any use. They are not retained by default, but their values can be saved to flash using the `FLASH IMAGE` command. It is recommended to leave this allocation at default.

The memory for defines is also set here. By default, the user is given 100 Defines. This is sufficient for many applications, but some may need more. If more are required, the allocation must be increased and the

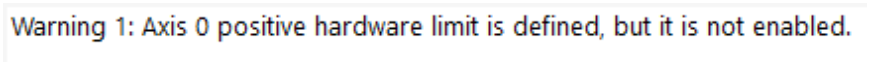
configuration must be downloaded to the controller again. If the configuration is re-downloaded, it is also a good idea to download everything else as well since downloading a configuration wipes programs.

# *Finish and System Code*

The Finish screen completes the Configuration Wizard. It displays any errors and warnings. Errors will need to be corrected before downloading. Click on the error to go to that section and fix it.

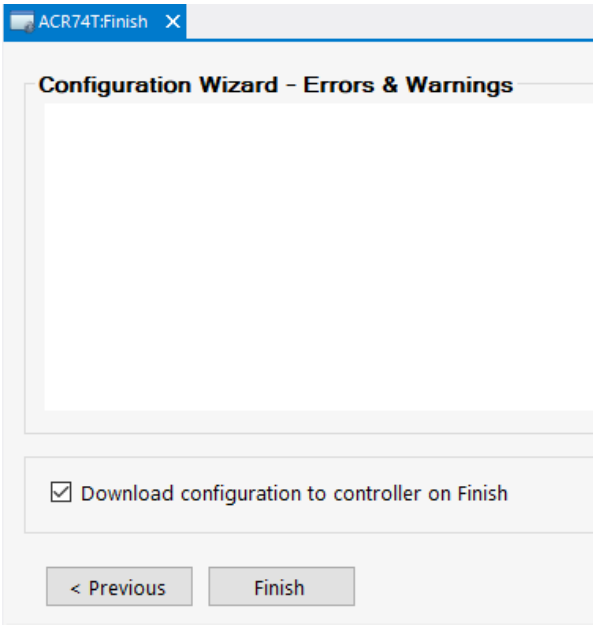


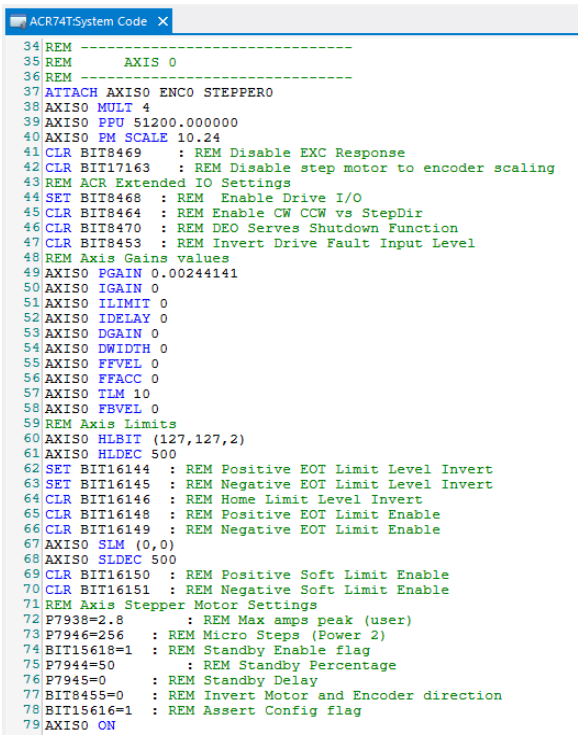

Warnings are a heads-up to double-check the settings. Click on the warning to go to that section and fix it. Warnings do not prevent proceeding to download.




If PMM is connected to the controller, the *Download configuration to controller on Finish* check box will be available and checked by default. Clicking *Finish* completes the Configuration Wizard and generates the System Code. If the aforementioned box is checked, it also initiates a project download.

When *Finish* is clicked, the entire Configuration Wizard settings are used to generate the AcroBASIC code shown in the System Code viewer. The System Code is cleared when the project is closed or a setting within the Configuration Wizard is changed. *Finish* needs to be clicked to regenerate this code before downloading. Clicking *Finish* will also save the project.



# Program Editor

The Program Editor section in the Explorer has fifteen program editors (Program 00 to Program 14) and the Defines editor. The program editors are used for writing programs in AcroBASIC and support syntax highlighting:

- BLUE for AcroBASIC keywords.
- CRIMSON for text strings.
- GREEN for comments.

AcroBASIC programming is covered in detail in Programming Basics.

The Defines editor provides a central location for defined aliases, referred to hereafter as defines. Programmers can use defines to refer to bits and parameters by name in their programs. Any bit or parameter can be assigned a define. Using a define instead of a bit or parameter number can make a program more readable and easier to maintain. Defines are global and are recognized across all programs as well as the system prompt in the Terminal Emulator.

Programmers can also define constants in the Defines editor. The only permissible values are positive integers and zero.

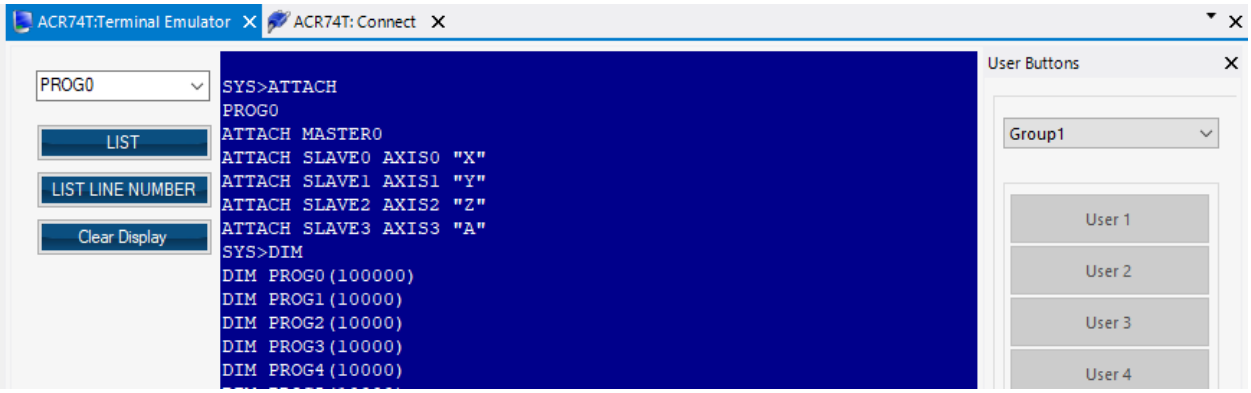
Defines

| | Alias | Description | Type | Address/Value |
|---|---|---|---|---|
| ▸ 1 | rLength | User Variable P001 | Parameter | 1 |
| 2 | iConversion | User Constant | Constant | 32768 |
| 3 | NotInPositionX | NotInPosition | Bit | 768 |
| 4 | JoggingX | JogActive Axis0 | Bit | 792 |

# Terminal Emulator

The Terminal Emulator allows programmers to send AcroBASIC commands directly to the controller. The Terminal Emulator is frequently used to:

- List programs so that their contents can be reviewed without having to do an entire project upload.
- Listen to a program's `PRINT` statements, which are useful for debugging.
- Change settings, like move dynamics or the IP address.
- Set and clear bits.
- Interrogate bits and parameters for their values.
- Issue motion commands for testing, troubleshooting or prototyping.




## *Prompts*

The short sequence of characters that is printed by the controller at the start of every line is called the *prompt*. The prompt displayed represents the context of the Terminal Emulator and where any typed commands will be routed in the controller.

`SYS>` is the system prompt. Here, the programmer can use commands like `ATTACH` and `DIM` to query system-level information like axis attachments and global memory allocation.

`P00>` is the prompt for Program 0. There is a prompt for every program. Each program prompt gives the programmer access to the data within that program. For instance, issuing the `DIM` command from the `P03>` prompt displays local memory allocation for variables and arrays in Program 3. Certain motion command, specifically the ones requiring the use of an axis alias, will only work from the program prompt to which their master is assigned (Program 0 by default).

Use the pull-down menu in the top left to switch prompts. Users can also change the prompt by typing `SYS`, `PROG0`, `PROG1`, etc.

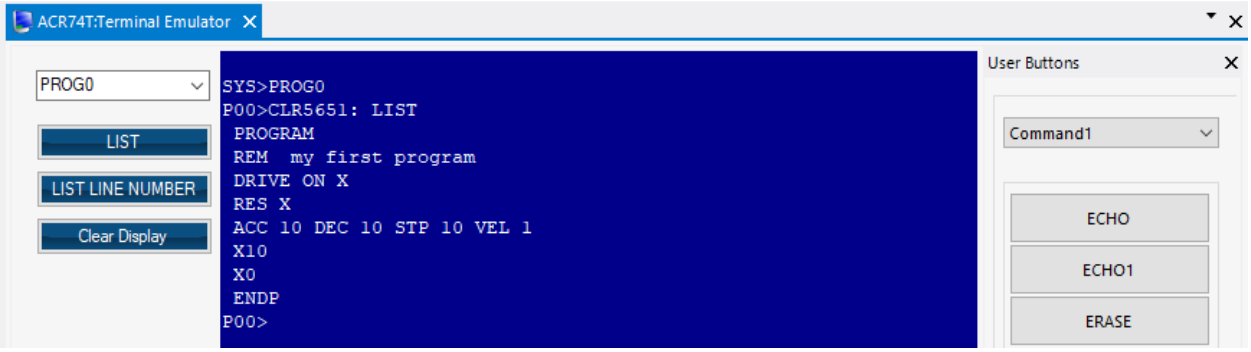
## *Basic Terminal Operations*

Most operations in the Terminal Emulator are accomplished by typing a valid AcroBASIC command. After typing the command, the programmer must press *Enter* for the command to be accepted by the controller. Commands

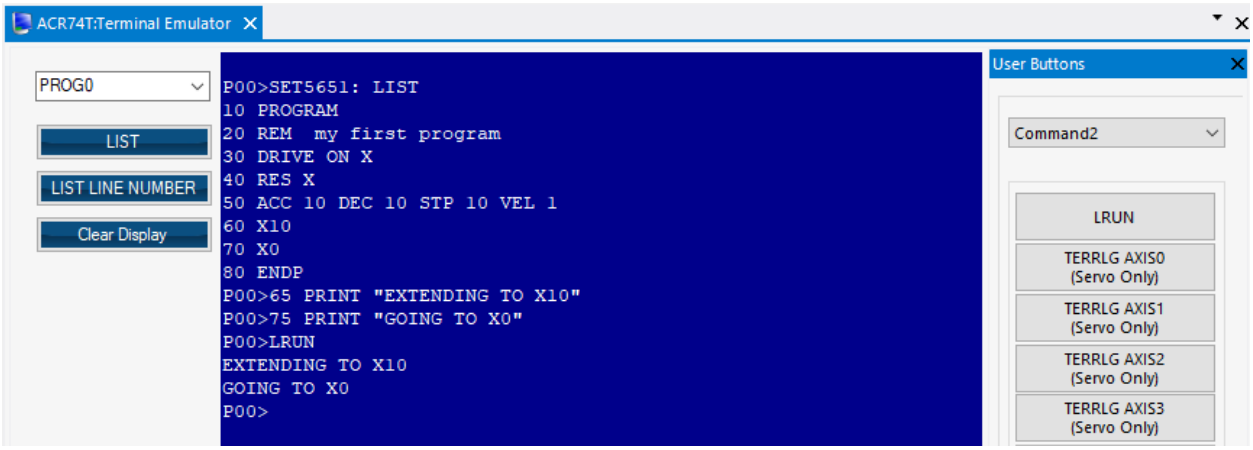
are case-insensitive, but axis aliases (e.g. X or Y) and defines are not. While it is typical for one line to have one command, multiple commands can be put on the same line by separating them with a space, a colon and another space (" : "). This is helpful when trying to get two commands to process in rapid succession when issuing them manually.

New for PMM!

Programmers can save time by using the up and down arrow keys (↑ and ↓) to scroll through commands that have already been used in the session. This is an easy way to repeat commands.




To list a program, switch to that program prompt and click the *LIST* button or *LIST LINE NUMBER*.




Every AcroBASIC program has a line number for each line. Line numbers proceed in increments of 10. These line numbers are shown in the Motion Status Panel when a program is running. When troubleshooting, the line number can be used to find out which command is causing a program to stall or abort.

Programmers can use the Terminal Emulator to temporarily add extra lines of code between the existing ones. This is done by typing a line number followed by a command, for example, "`21 AXIS 0 DRIVE ON`". Lines can also be overwritten using this method. Lines can be deleted by simply typing the line number and pressing *Enter*. These changes remain in effect until power is cycled or a `REBOOT` command is issued. If desired, the changes can be permanently saved to flash using the `FLASH IMAGE` command.

For very large programs, users can partially list a program up to a specific line number.

```
P00>LIST,40
10 PROGRAM
20 REM  my first program
30 DRIVE ON X
40 RES X
```

The LIST command can also display a part of the program between a range of line numbers.

```
P00>65
P00>list 50,80
50 ACC 10 DEC 10 STP 10 VEL 1
60 X10
70 X0
75 PRINT "GOING TO X0"
80 ENDP
```

To start two programs and the same time and listen to one, separate RUN PROGx and LRUN with a "space : space" command delimiter.

```
P00>PROG1
P01>list
10 PROGRAM
20 P10= 0
30 _MAIN
40 INH 82
50 P10= P10+1
60 PRINT P10
70 INH -82
80 GOTO MAIN
90 ENDP
P01>
P01>run prog0 : lrun
1
2
3
4
5
6
```

The LRUN command runs a program and enters listen mode (LISTEN + RUN) and allows users to see output from PRINT statements while the program is running. To exit LISTEN mode, press *Escape*.

In the above sample, bit 82 is a system bit that toggles every second. INH is an inhibit command waiting for bit 82 to turn on and INH -82 waits for it to turn off. Hence, global parameter P10 is incremented every second and is printed as the program loops through _MAIN.

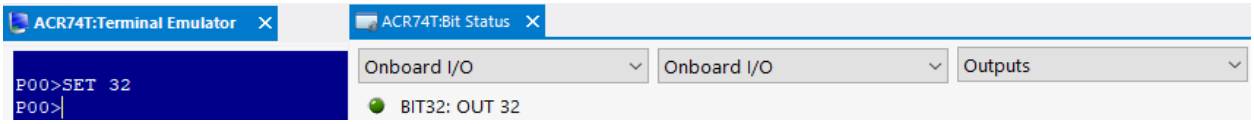
The status of a bit or parameter can be checked in the Terminal Emulator using the PRINT command. When checking bits, the bit is "clear" ("off" or "false") if it returns 0 and "set" ("on" or "true") if it returns -1. The example below checks the status of the PROG0 Running bit and indicates Program 0 is indeed running.

```
P00>PRINT BIT1024
-1
```

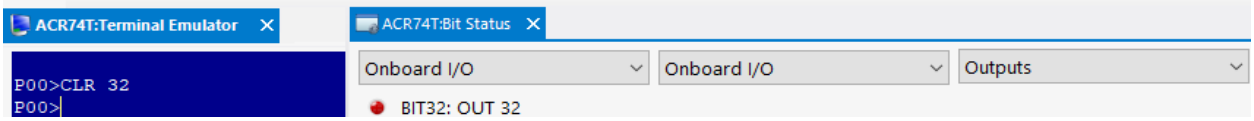
The "?" operator can be used as shorthand for PRINT, saving time when querying bits and parameters.

```
P00>? BIT1024
0
```

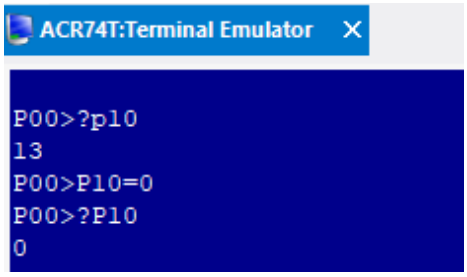
Write to a bit with SET and CLR.




Bit 32 is the controller's first onboard output. If it is not connected to an indicator, its status can be viewed on the Bit Status panel or queried by issuing "? BIT32" in the Terminal Emulator.

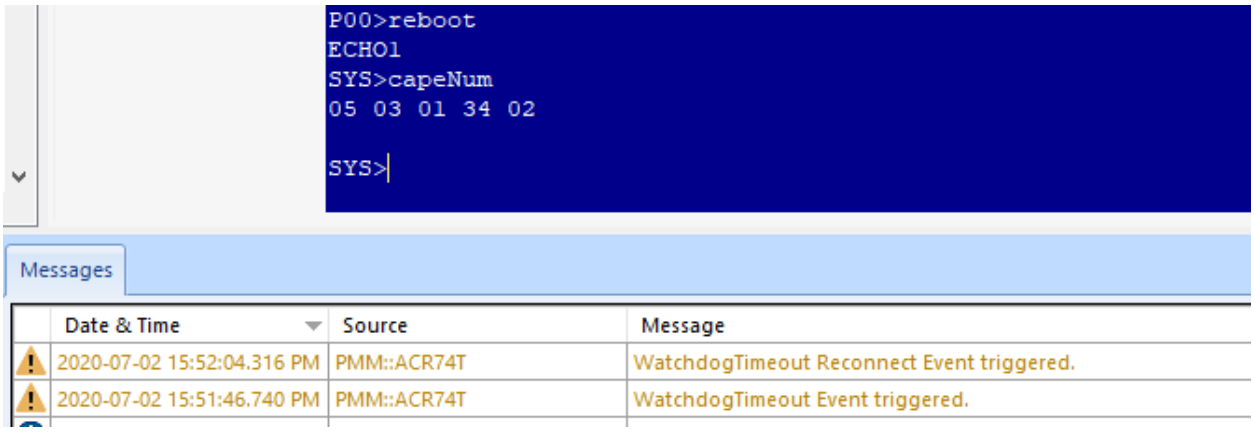



Parameter values can be checked with PRINT or "?". Parameters can be assigned values with "=".




Programs marked with the PBOOT command are automatically started on power-up. To start running all PBOOT programs without having to cycle power, issue the PBOOT command in the Terminal Emulator.

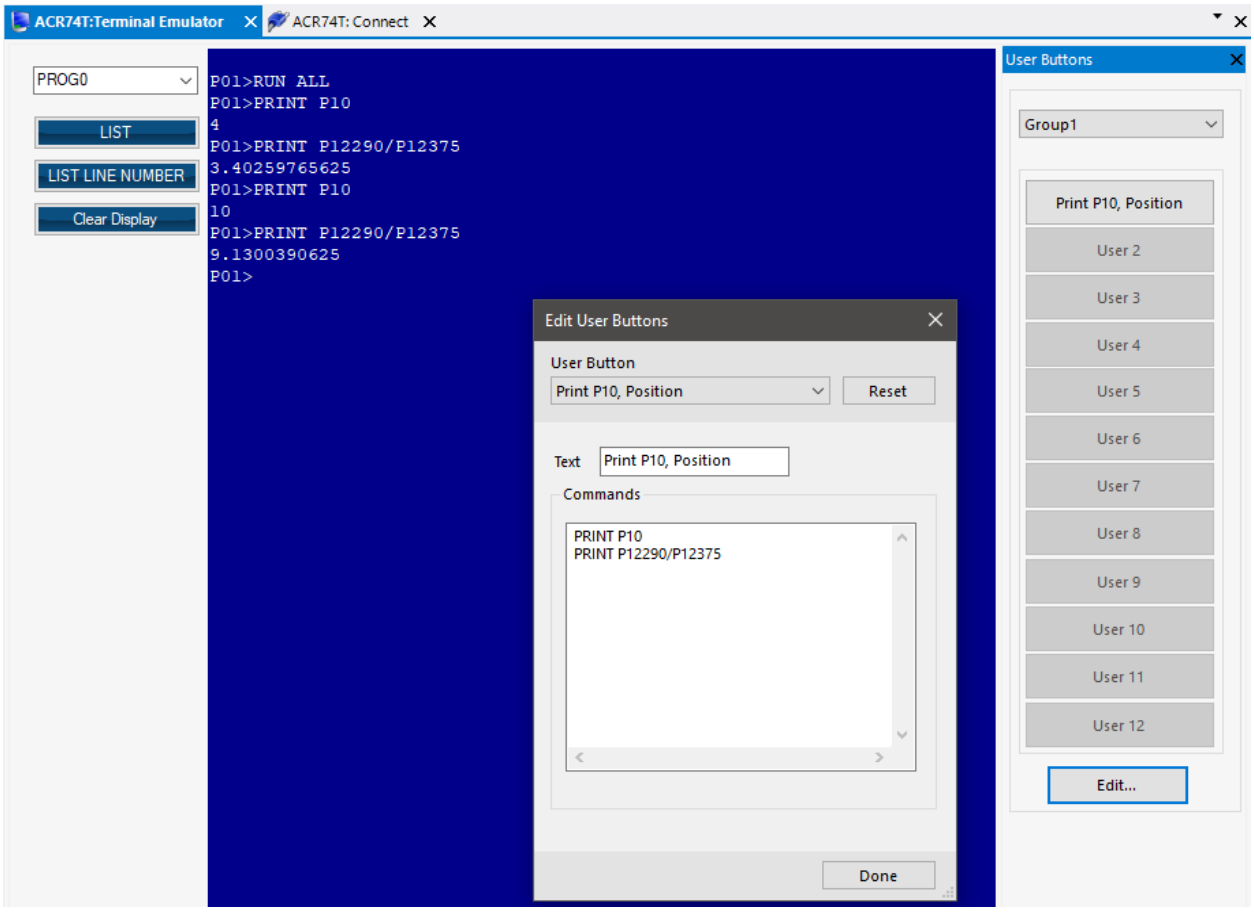
To cycle power on a controller, issue the REBOOT command. Note that the connection will be lost when the controller reboots. PMM will reconnect automatically after a few seconds. The message window will show the connection timeout and reconnection.




# *User Buttons*

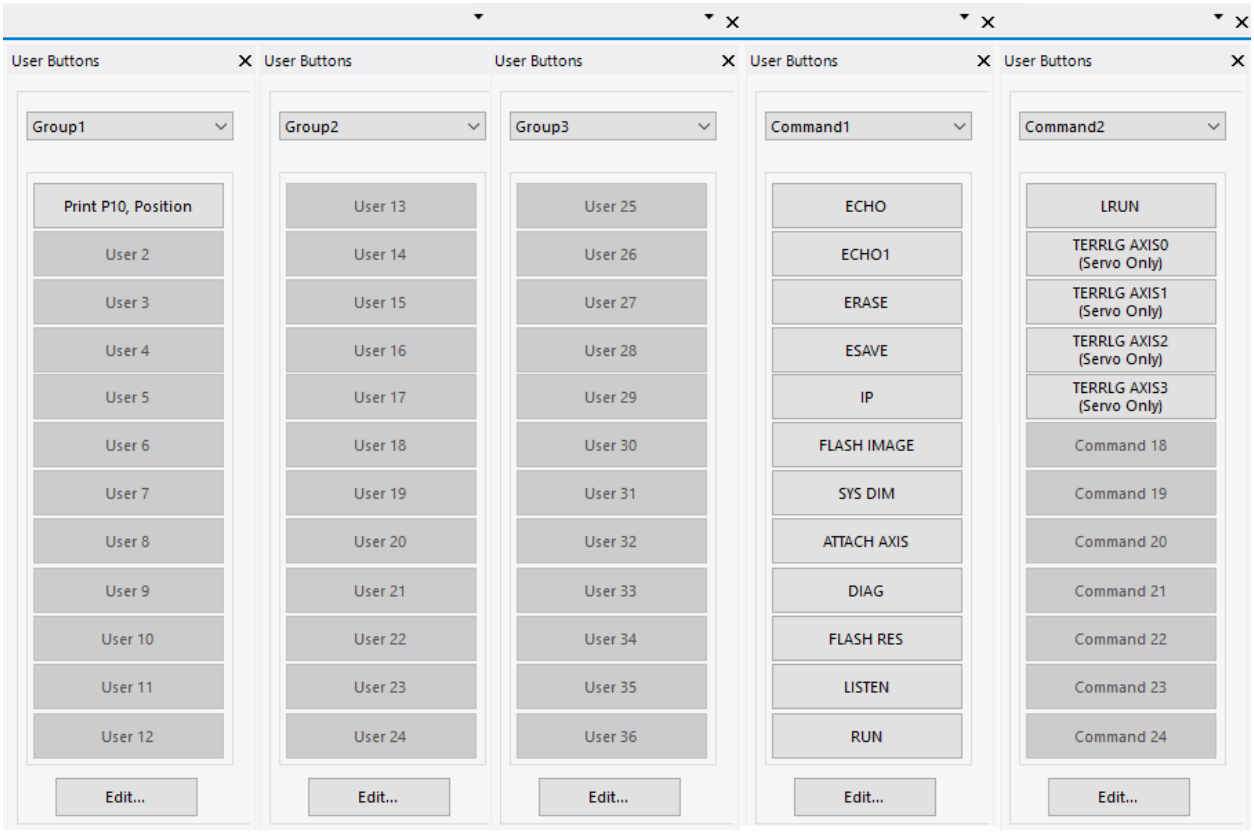
New for PMM!

User Buttons have been improved. Users can now name these buttons and insert commands or multiple lines of code. This can save time and prevent typos while debugging. The code is sent with a mouse click.




In the example above, the `RUN ALL` command starts all programs. `P10` is a global user parameter and `P12290/P12375` is the actual position for Axis 0, scaled in engineering units. These commands can be added to a button to automate tedious and repetitive command sequences.

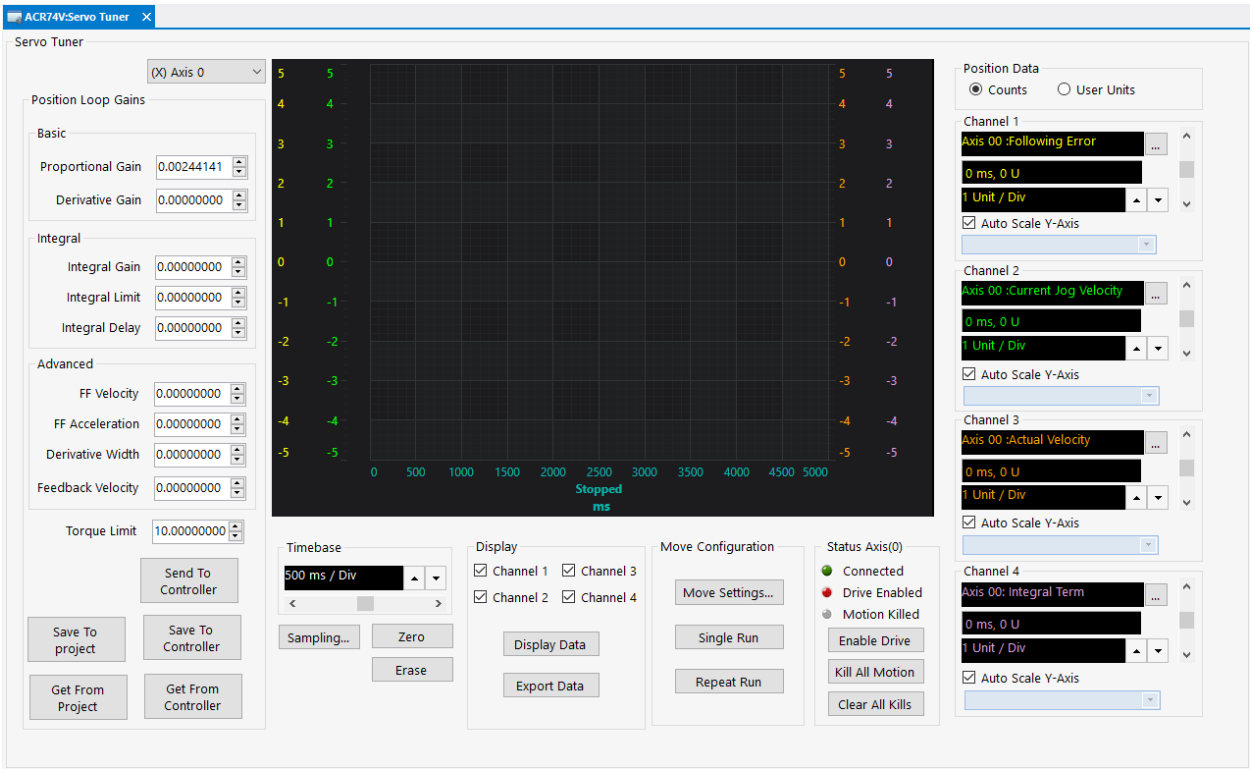
There are 60 User Buttons (5 groups of 12) with common commands preloaded in the last two.



# Tools

The Tools section in the Explorer includes the Servo Tuner, Jog/Home/Limits screen and OS Update panel.

## *Servo Tuner*




The Servo Tuner is a fast and easy way to tune servo axes. The axis can be selected from the pulldown in the upper left-hand corner of the Servo Tuner. Servo gains are listed on the left-hand side.

This section is intended to provide an overview of the Servo Tuner itself. For a procedure on tuning a servo axis see Servo PID Tuning.

The Servo Tuner is broken into several panels:

- Channels
- Position Loop Gains
- Timebase
- Display
- Move Configuration
- Status Axis(0)
- The Graph

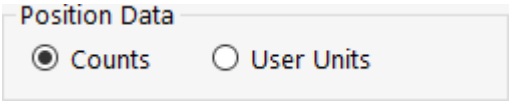
### Channels

By default, the four channels are set up as follows:

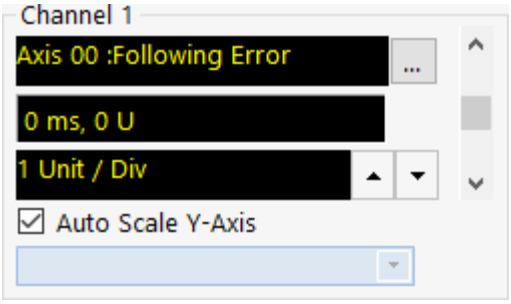
- Channel 1 shows Following Error in yellow. Units are encoder counts.
- Channel 2 shows Current Jog Velocity in green. Units are encoder counts per second.
- Channel 3 shows Final Output Signal in orange. Units are ±10 and represent the torque command in volts.
- Channel 4 shows Secondary Setpoint in purple. Units are encoder counts.

New for PMM!

In addition to controls specific to each channel, there is a global setting for all position-related parameters to be graphed in encoder counts (default) or in user units.

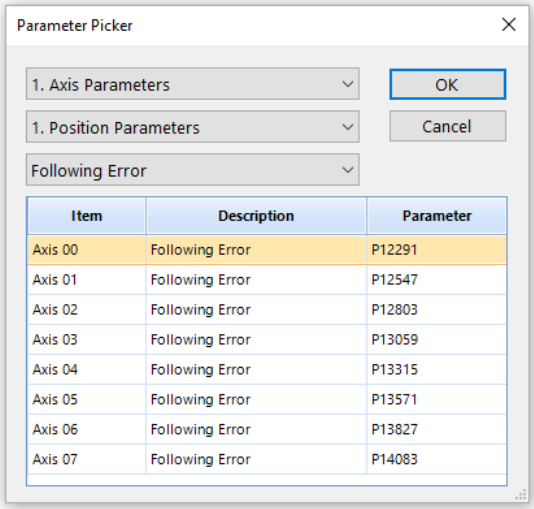



Each channel has several controls:




- The parameter field (top) displays the parameter being graphed. This can be changed to any parameter desired using the "…" button to the right of the field (details below).
- The time field (second from top) displays the horizontal and vertical shift for this channel. The vertical shift can be altered using the vertical slider at the right. See Timebase for details on the horizontal shift.
- The unit field (bottom) field displays the vertical scale (default 1 unit/div). This can be altered using the up and down arrow buttons to the right. Check *Auto Scale Y-Axis* (checked by default) to make the graph fit the available vertical space. This option is usually preferred as it makes the data easy to read.

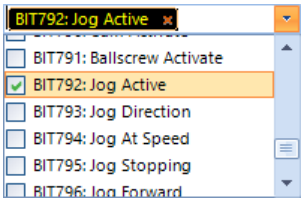
When the user clicks the "…" button to select a new parameter, the Parameter Picker dialog appears. This dialog helps the user drill down to a parameter of interest by using three pull-down menus. The top menu selects the parameter group, the second menu selects the subgroup and the third selects the individual parameter type. The list at the bottom breaks a specific parameter type (Following Error in this case) down into enumerated options, often based on axis number (otherwise encoder number, stream number, ADC number, etc.).




The Parameter Picker dialog is consistent across PMM and is also used in the Oscilloscope and Strip Chart tools. It is conceptually very similar to the Pick A Bit dialog that serves the same purpose for bits. The same pull-down menus used in the Parameter Picker are also used in the Numeric Status panel.

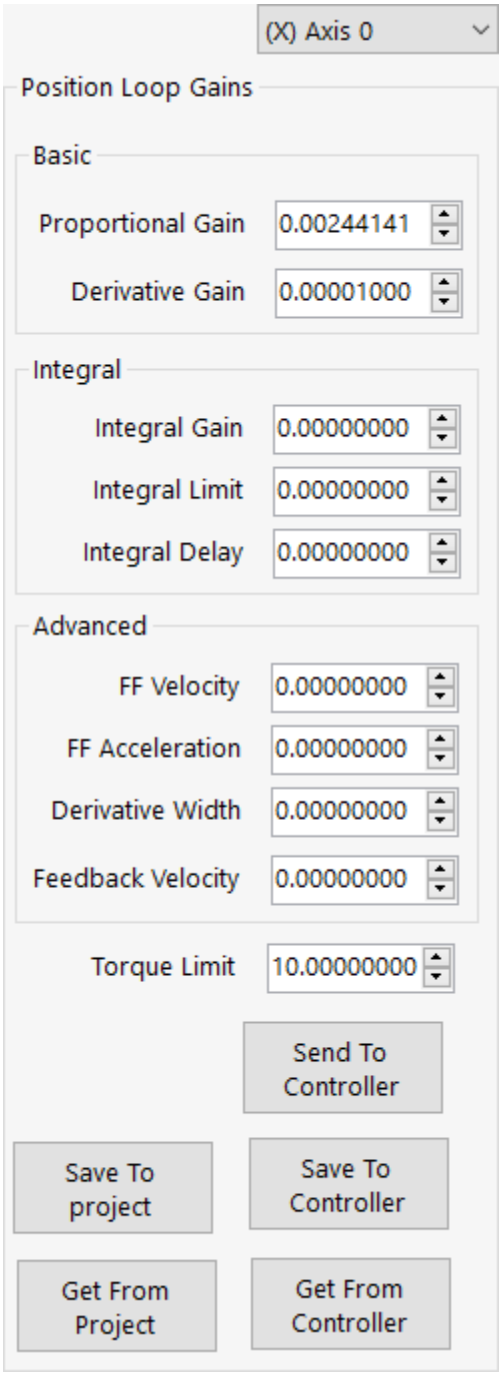
New for PMM!

When a flag parameter is selected (P4096-4375), the bottom pull-down makes it possible to select a specific bit (or bits) to watch in the scope.



## Position Loop Gains

Servo gains can be changed using the fields on the left-hand side of the Servo Tuner. There is a pull-down at the top of the panel to select which axis to modify.

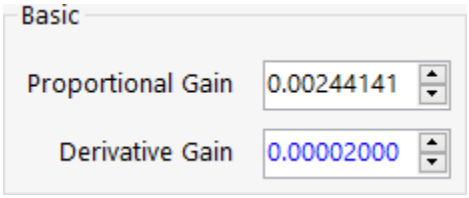



The gains are broken into three groups:

- Basic gains are used on every application.
- Integral gains are used on applications requiring very precise settling.
- Advanced gains are used on applications requiring precision tracking or high acceleration.

There is also a Torque Limit field near the bottom. Scaled 0-10, this allows users to limit output torque on an axis. On an ACR7xC standalone controller, the value in this field represents the physical voltage limit of the torque command analog output.

After changing a value, press *Enter* to send that value to the controller. Otherwise, the text will turn blue to indicate that the value has not been sent. The *Send To Controller* button can be used to send multiple values at once.




This panel comes with several buttons:

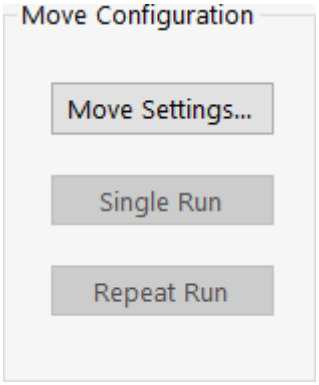
- *Save To Project* saves the values in the panel to the project file on the hard drive.
- *Get From Project* loads values from the project file into the panel, but does not send them to the controller.
- *Send To Controller* sends the values from the panel to the controller.
- *Save To Controller* issues an ESAVE command to the controller, which tells the controller to save its current values to flash memory.
- *Get From Controller* uploads values from the controller to the panel.

After getting the gains dialed in, it is a good idea to press *Save To Controller* and *Save To Project* to make sure the gains are preserved and will not be lost when power is cycled.

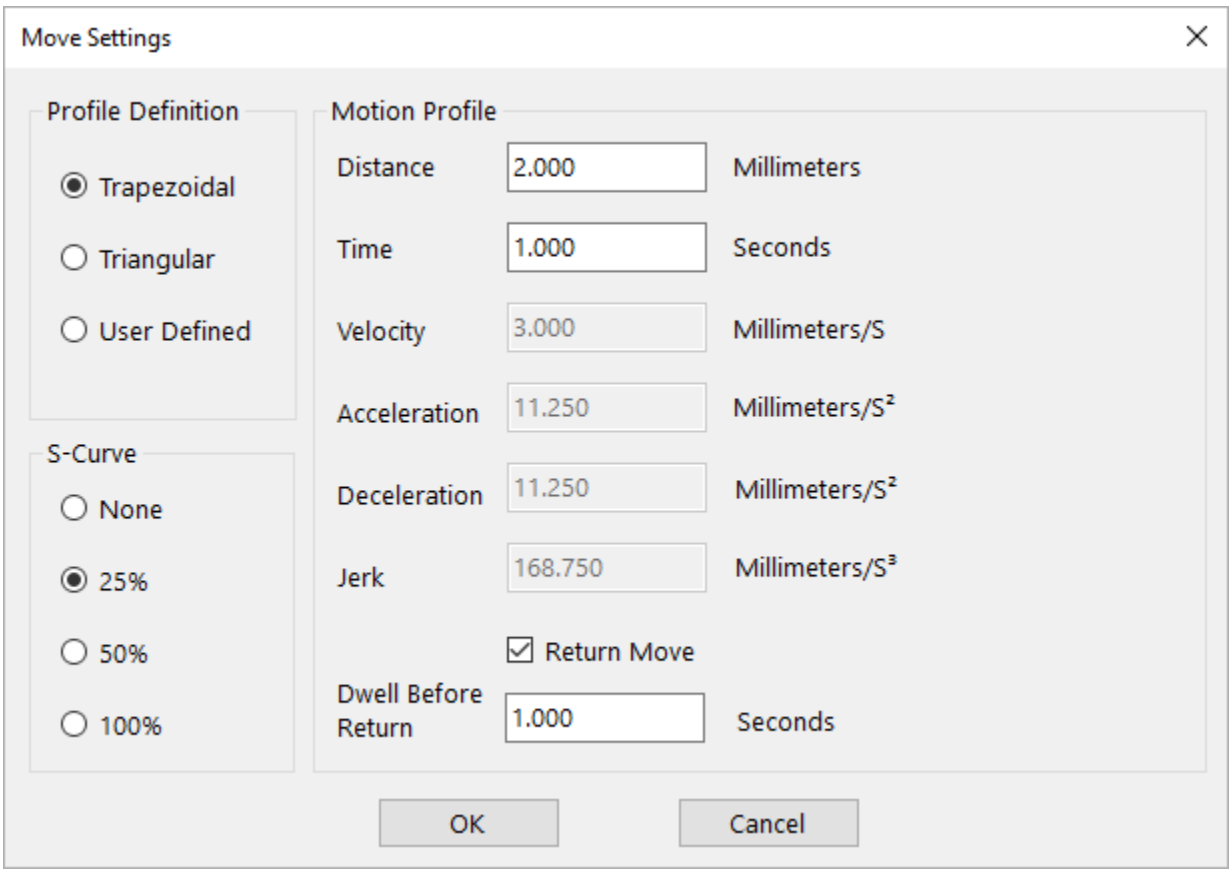
## Move Configuration

The Move Configuration panel has three buttons:

- Move Settings
- Single Run
- Repeat Run




The Move Settings dialog allows users to input a distance and time for a test move that will be executed by the Servo Tuner. It automatically calculates the velocity and acceleration ramps required and allows the user to specify several levels of S-curve profiling (jerk limiting). Users can check *Return Move* to return to the starting position after the end of the move. Users can also select a triangular motion profile or their own user-defined profile. It is best to start testing with a small move, tune the axis with a basic move and then tune to a move typical for the application.

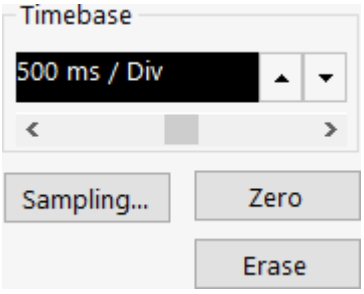



Users can execute the move once by clicking *Single Run* or multiple times by clicking *Repeat Run*. This provides the ability to change tuning gains while doing the same move over and over to see the effect of the changes (rather than having to click to start the move repeatedly). Note that *Repeat Run* requires onboard sampling, covered under Timebase.

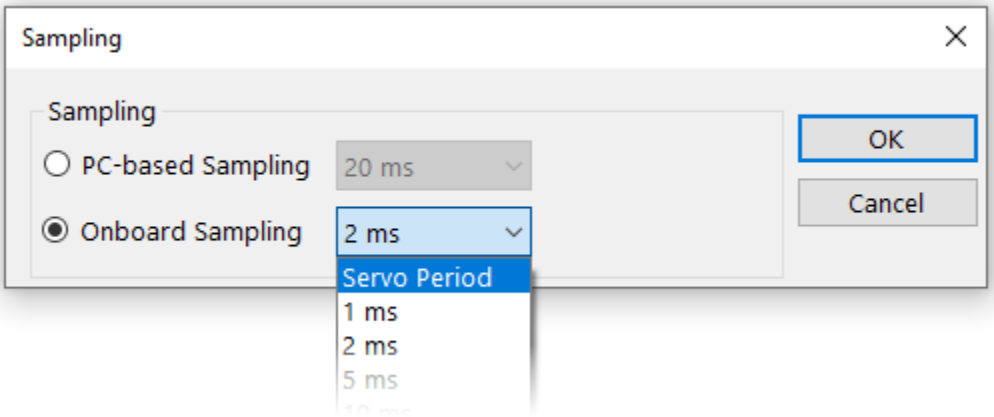
## Timebase

The Timebase panel controls the graph's time (horizontal) axis. It consists of several tools:

- The time/division indicator shows the length of time represented by one division on the horizontal axis. This can be changed using the up and down buttons to its left.
- The slider beneath the time/division indicator can be used to scroll data in the graph back and forth horizontally.
- Clicking *Zero* resets the time slider as well as the vertical slider for each channel, an effect only noticeable if Auto Scale is disabled.
- Clicking *Erase* erases all data from the graph.




Clicking *Sampling* opens the sampling dialog. Here, the user can select *PC-based Sampling* (default) or *Onboard Sampling*.




PC-based sampling means that PMM will request the parameter value over Ethernet at the specified rate. The sample data is transmitted as needed without buffering. This is convenient and does not impose a memory burden on the controller, meaning the graph can store very large data samples. However, PMM does not permit sampling faster than 20 ms with this option to avoid taxing network and processor resources on the controller and the user's PC.

Onboard sampling means that the controller will allocate a memory buffer for the data it needs to take in advance. When the user clicks *Single Run* or *Repeat Run*, the controller will store the data it acquires in the buffer and transmit it all at once after the test is finished. The main advantage of this option is that it allows the user to acquire data at a faster interval (all the way down to the servo period). However, ACR controllers have limited memory and large data samples are not always possible. If there is not enough memory available to run the test with onboard sampling, PMM will log the error message "failed to allocate program memory for sampling buffer" to the Messages window.

**NOTE: The Repeat Run button requires onboard sampling.**

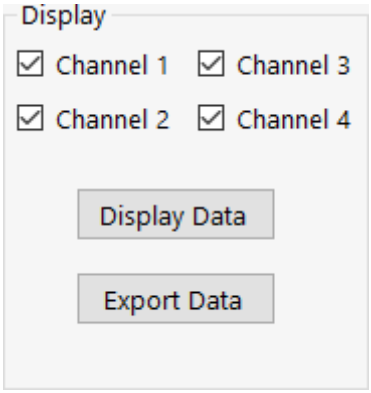
The approximate amount of memory in bytes required to run a test move with onboard sampling can be calculated using the following formula:

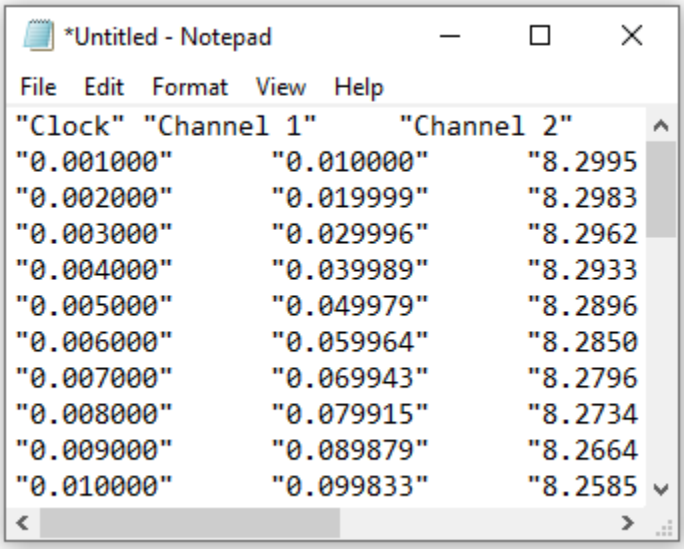
$$M_{bytes} = 4N_{channels}\left(\frac{t_{test}}{t_{sample}}\right)$$

$N_{channels}$ is the number of channels in use, $t_{test}$ is the length of time visible on the horizontal axis in seconds and $t_{sample}$ is the sample time in seconds. If the sample time is set to *Servo Period*, the actual sample time depends on the controller in use. For ACR7000 controllers, the default servo period is 500 µs. For the IPA, it is 250 µs.

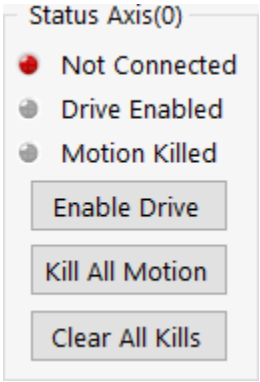
## Display

The Display panel has four checkboxes that allow the user to show or hide each channel on the graph. Users can click *Display Data* to display all of the captured data in a textual format, which makes it easy to copy the data to other applications like Microsoft Excel. Users can click *Export Data* to directly save the data to a text or CSV file.




## Status Axis(0)




This panel has indicators to show whether PMM is connected to the controller, whether this axis is enabled and whether a Kill All Motion Request is active for this axis. A Kill All Motion Request prevents all motion on an axis.

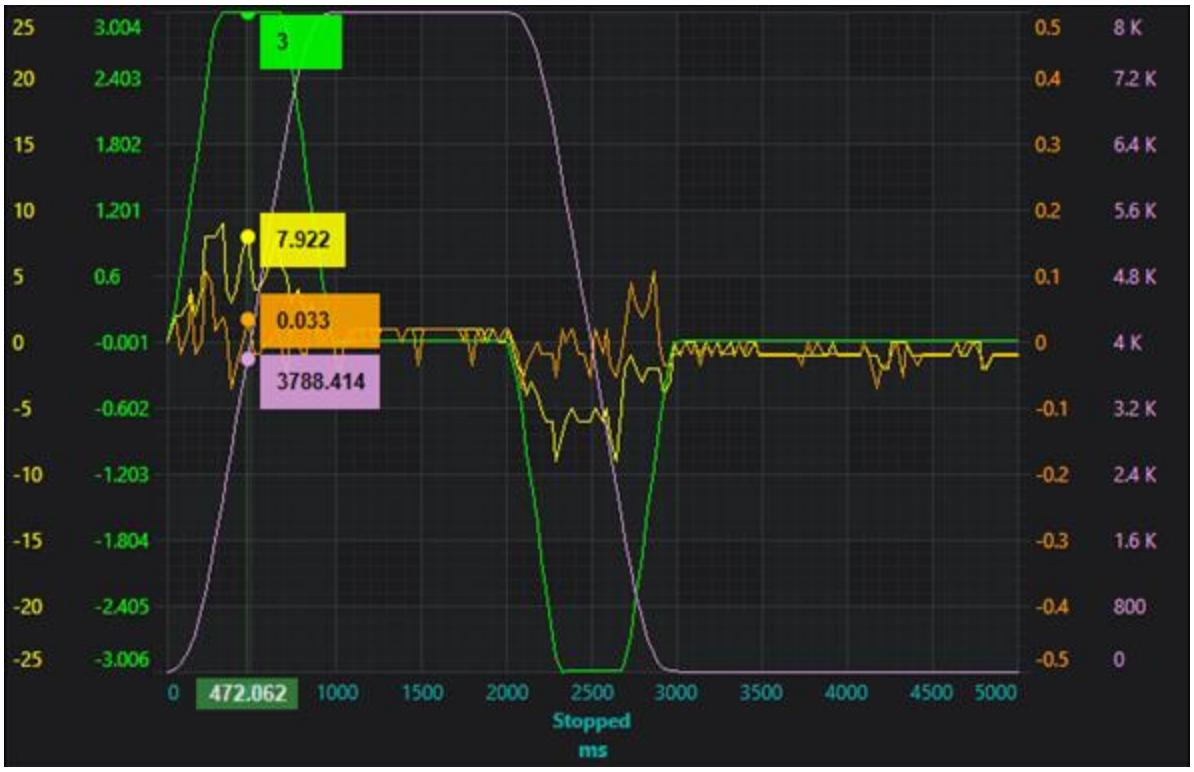
Users can click *Enable Drive* to enable this axis. *Kill All Motion* issues a Kill All Motion Request to the current axis. *Clear All Kills* removes the Kill All Motion Request to allow motion again.

## The Scope

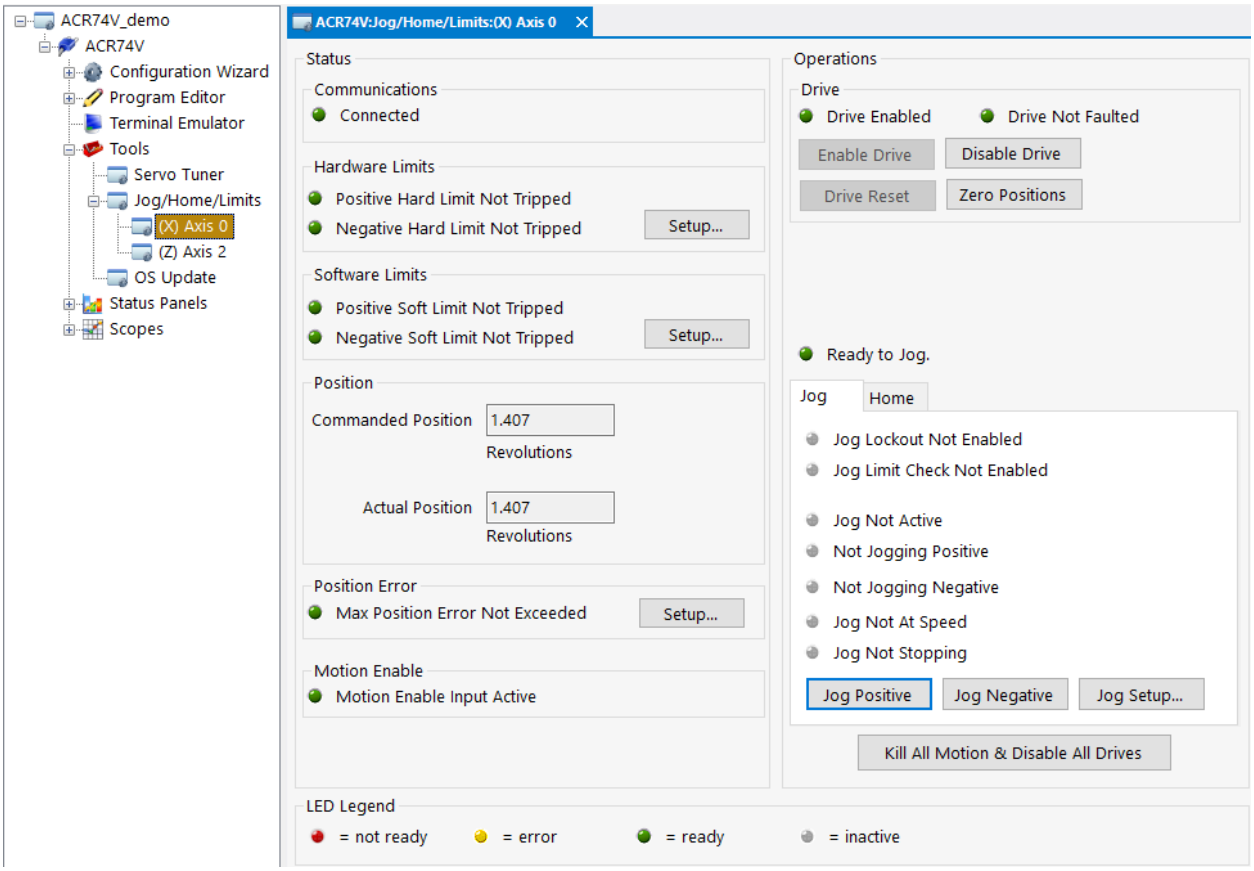
The scope is the central feature of the Servo Tuner and shows data captured from the controller during a test. This helps users visually understand what their axis is doing during the test. It is common to graph parameters like Following Error, Secondary Setpoint, Actual Torque and other control loop parameters.

New for PMM!

Data channels display their current values when sweeping the cursor over the scope. This makes it easy to correlate specific channel values with specific times. One of the biggest new features to the Servo Tuner in PMM is Auto Scale (on by default), which makes the scope much easier to read.


25
20
15
10
5
0
-5
-10
-15
-20
-25
3.004
2.403
1.802
1.201
0.6
-0.001
-0.602
-1.203
-1.804
-2.405
-3.006
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
8 K
7.2 K
6.4 K
5.6 K
4.8 K
4 K
3.2 K
2.4 K
1.6 K
800
0
3
7.922
0.033
3788.414
0
472.062
1000
1500
2000
2500
3000
3500
4000
4500
5000
Stopped
ms

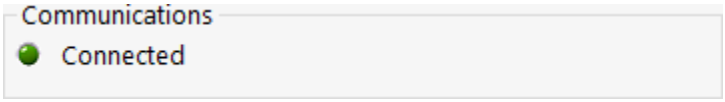
# Jog/Home/Limits




The Jog/Home/Limits screen gives users the ability to enable the drive and jog the motor in either direction and provides additional dialogs for users to fine-tune their limit and home settings. The screen is divided into several subpanels that either display status or allow the user to perform an operation with the axis. Each subpanel is discussed in detail here.

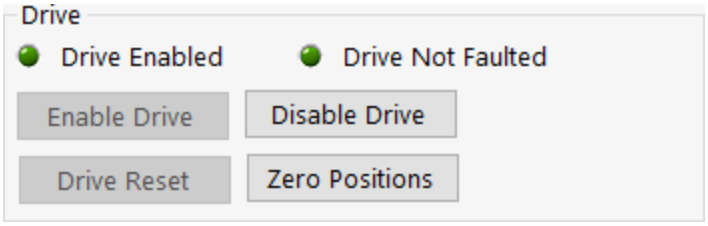
## Communications

The Communications subpanel has a single indicator that shows whether PMM is connected to the ACR. PMM must be connected for most of the tools on this screen to function.

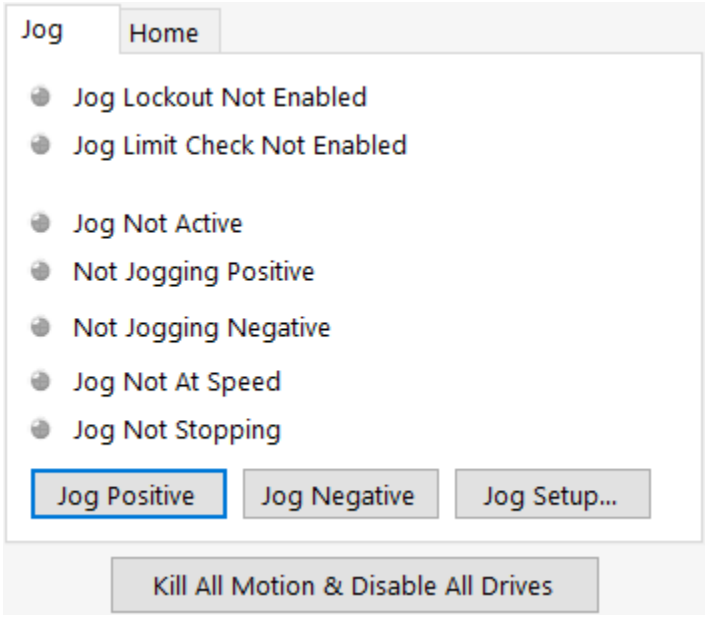



## Drive

This subpanel has an indicator to show whether the drive is enabled and another to indicate whether it is faulted. It has buttons to enable or disable the drive. Click *Drive Reset* to recover from drive-related faults, such as encoder loss or overtemperature. Click *Zero Positions* (equivalent of `RES` command) to reset the commanded and actual position to zero for this axis.




Below the Drive subpanel is the control panel for jogging. When the drive is enabled, click *Jog Positive* or *Jog Negative* to jog the axis. Clicking *Kill All Motion & Disable All Drives* will stop all motion on all axes and disable torque on all drives.

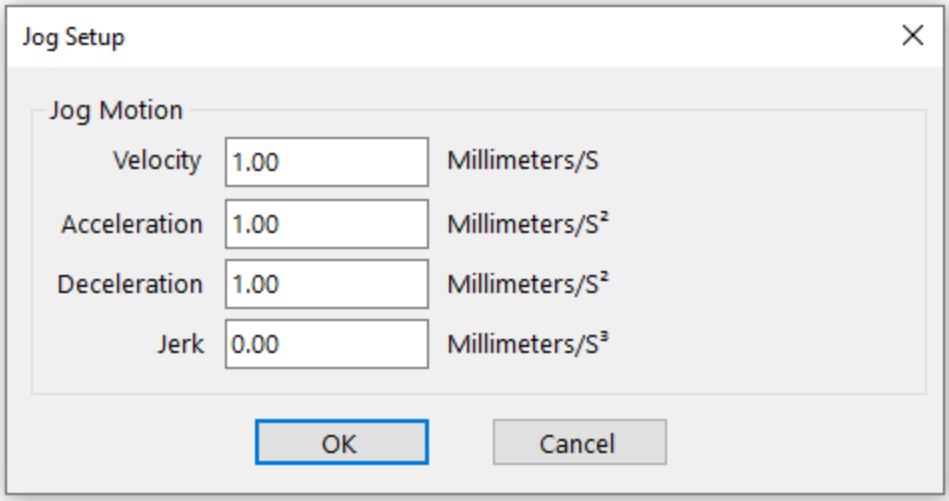



**NOTE: The Kill All Motion & Disable All Drives button is a software feature. It is not designed or tested for fault tolerance and is not a replacement for an Emergency Stop and machine safety plan.**

There are also several useful indicators on this panel:

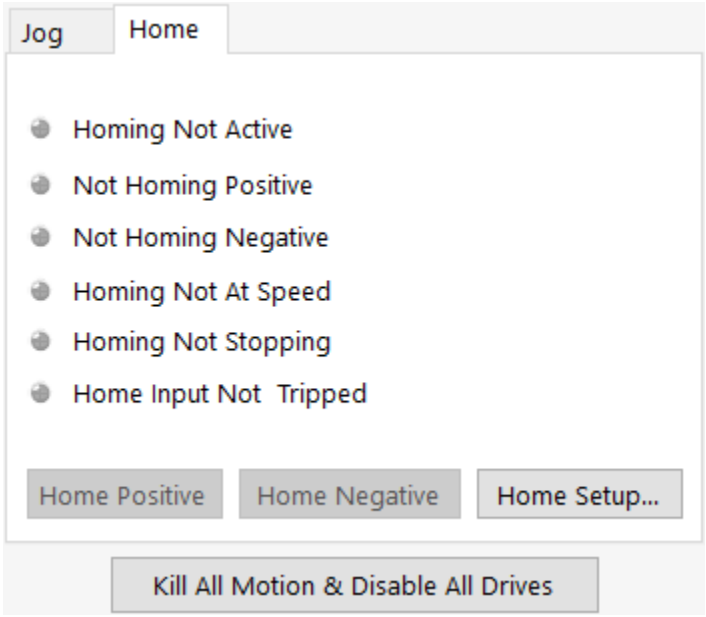
- Jog Lockout. When this bit is on, jog motion is inhibited.
- Jog Limit Check. On when the jog limits (`JLIM` command) are in effect.
- Jog Active. On when the axis is jogging.
- Jogging Positive. On when the axis is jogging in the positive direction.
- Jogging Negative. On when the axis is jogging in the negative direction.
- Jog At Speed. On when the Jog Profiler has finished ramping up to the user-set `JOG VEL` speed.
- Jog Stopping. On when the Jog Profiler is ramping speed to zero in preparation to stop.

The jog velocity, acceleration and deceleration can also be altered here. Click *Jog Setup* and a dialog will appear where new dynamics can be entered.




**NOTE: Remember that in the ACR architecture, setting acceleration, deceleration or jerk parameters to 0 is interpreted as setting them to infinity. In the example above, zero jerk will result in a trapezoidal or triangular move profile.**

Select the Home tab to view indicators and controls for homing the axis. Click *Home Positive* to start searching for home in the positive direction or click *Home Negative* to start searching for home in the negative direction.

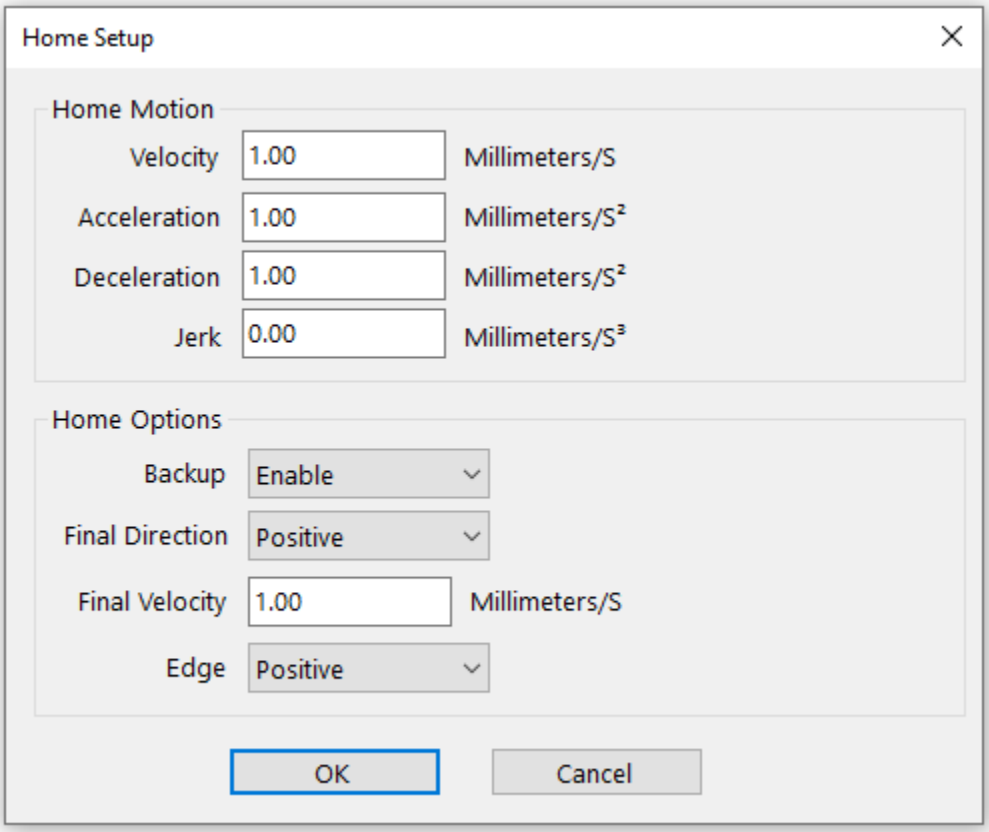



This panel has several indicators:

- Homing Not Active. Indicates whether the axis is homing.
- Not Homing Positive. Indicates whether the axis is homing with an initial positive direction.
- Not Homing Negative. Indicates whether the axis is homing with an initial negative direction.
- Homing Not Stopping. Indicates whether the axis is stopping its homing move.

- Home Input Not Tripped. Indicates whether the home input has turned on.

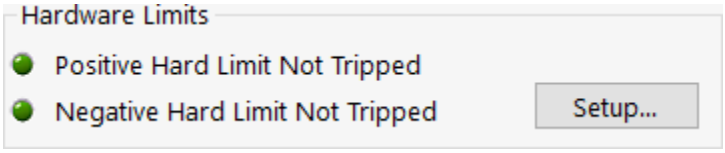
Homing dynamics can be altered on this screen. Click *Home Setup* to change them.



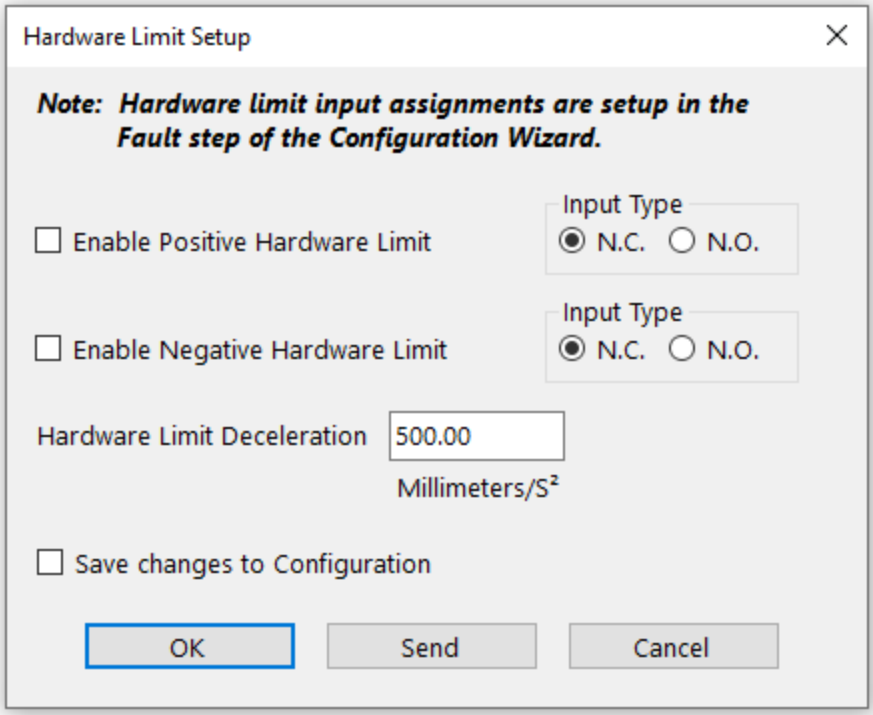

**NOTE: These options are not saved with the project configuration. However, the code to set these options can be copied out of the Terminal Emulator as long as it is open when clicking OK. Copy the code into a program for easy homing setup.**

## Hardware Limits

The Hardware Limits subpanel has indicators to display whether either of the limits have been tripped.

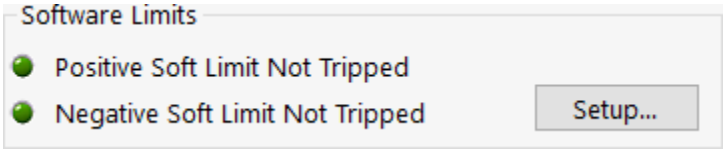



Click *Setup* to make adjustments to the limit switch configuration. Most of the options in this dialog should be familiar since they are also found on the Fault screen of the Configuration Wizard. Click *Send* to apply the new settings to the controller. To make the changes permanent, check the *Save changes to Configuration* box and click *OK*. This will apply the changes to the project configuration.

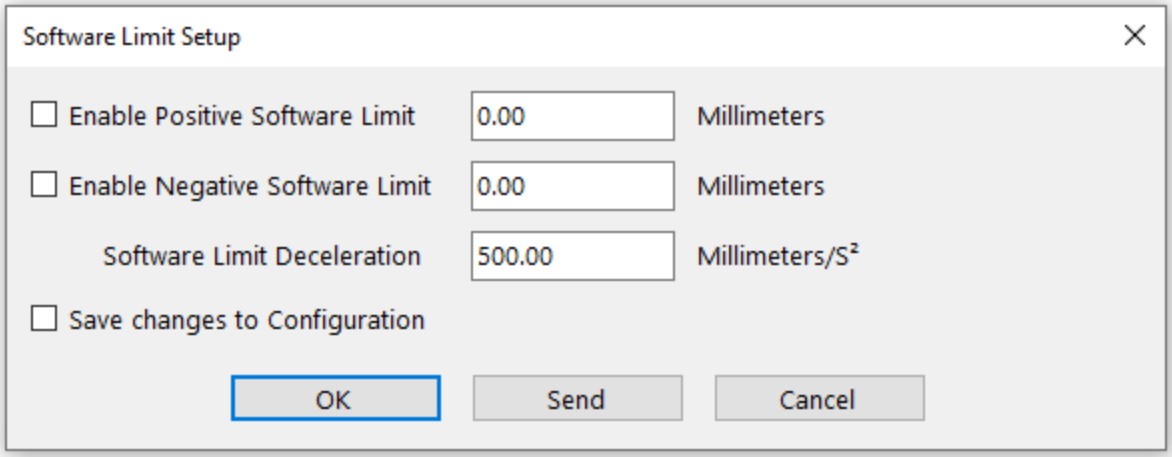



## Software Limits

The Software Limits subpanel shows the status of the position soft limits for the axis.

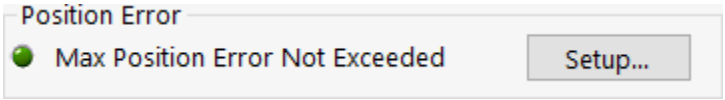



Click *Setup* to make changes. The options presented in this dialog work the same way as the ones on the Fault screen in the Configuration Wizard. Click *Send* to apply the new settings to the controller. To make the changes permanent, check the *Save changes to Configuration* box and click *OK*. This will apply the changes to the project configuration.

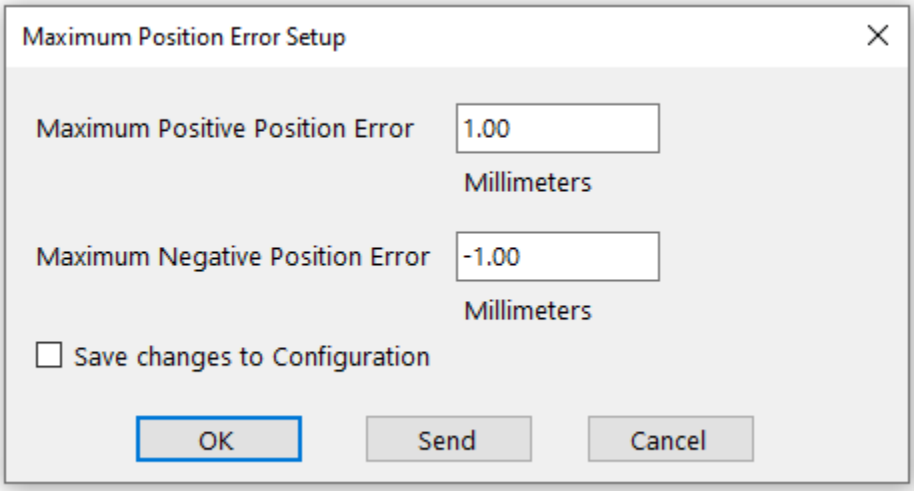



## Position Error

The Position Error subpanel shows whether this axis has exceeded its maximum allowable position error.




Click *Setup* to make changes. The options presented in this dialog work the same way as the ones on the Fault screen in the Configuration Wizard. Click *Send* to apply the new settings to the controller. To make the changes permanent, check the *Save changes to Configuration* box and click *OK*. This will apply the changes to the project configuration.




## LED Legend

This subpanel just displays four sample LEDs with information about what their color codes mean.

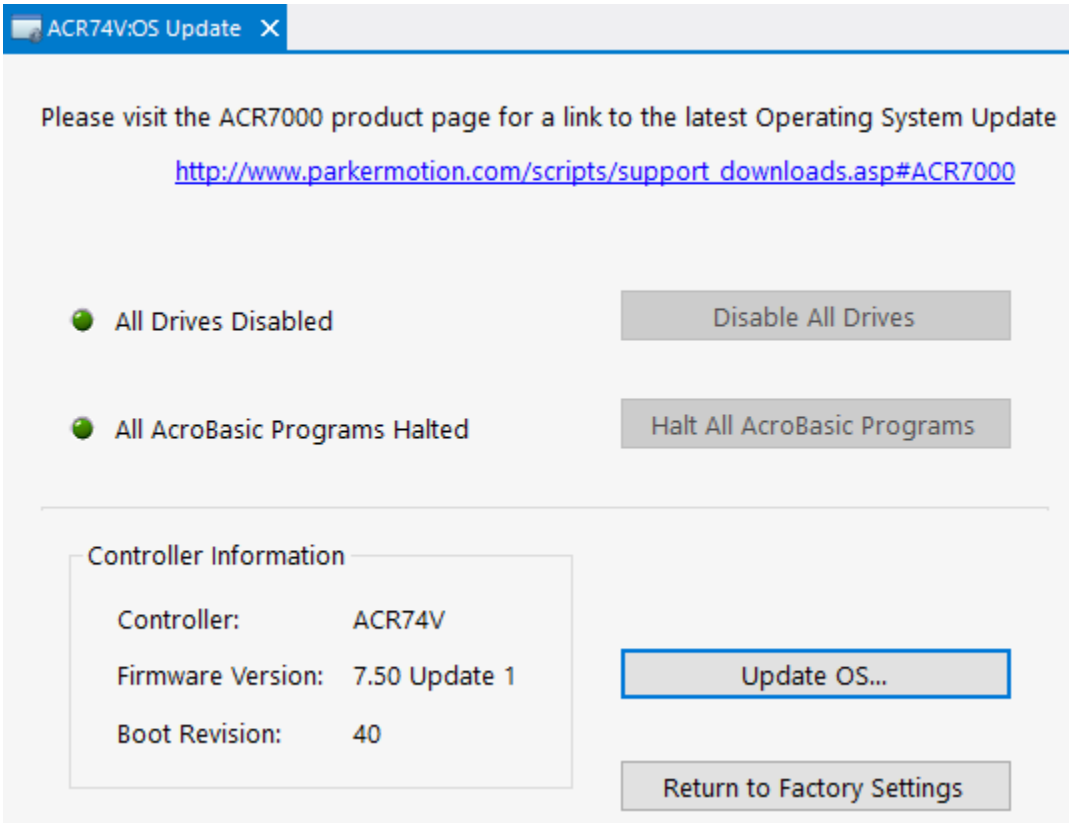
# OS Update

The OS Update screen is used to install a newer version of firmware into an existing controller. This may be required to take advantage of a previously unavailable feature or improvement. Some users prefer to standardize on one OS version and "back-rev" new units. This screen can be used to revert a controller to an older firmware revision as well.

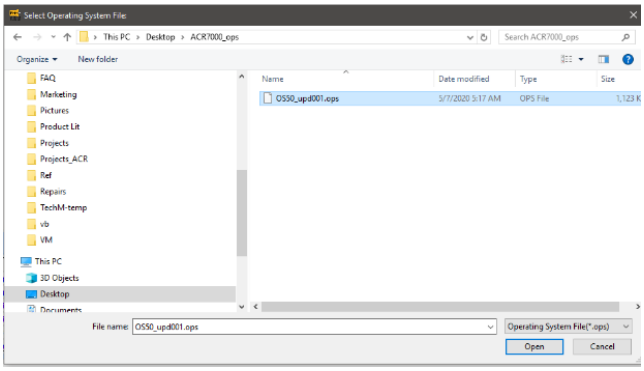
Basic controller information, including the model family, firmware revision and bootloader revision is shown under Controller Information.

It is required that the controller's memory be wiped before downloading a new OS. To do this, click *Return to Factory Settings* (same as issuing the `FLASH RES` command). This will also set the IP address back to the factory default. For the ACR7000, this is 192.168.100.1. For the IPA, the default address is 192.168.100.x, where "x" is determined by the rotary switches.

**NOTE: Make sure the program is backed up and saved on a PC prior to clicking Return to Factory Settings. All application data will be deleted from the controller by clicking this button.**



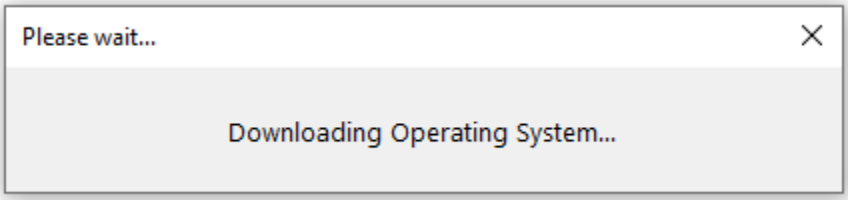

To install a new OS, click *Update OS*. This will halt any running programs and disable all drives (buttons and indicators are also provided to do this manually). A dialog will appear permitting the user to select a ".ops" operating system file and download it. Operating system packages for the ACR7000 or IPA can be downloaded from the link on this screen.




Click *Open* to start the download.

**NOTE: Do not interrupt the OS download process. Cycling power on the controller or disconnecting during an OS download can "brick" the unit and leave it in an unbootable state. If this happens, the unit may need to be returned to the factory for repair.**

Another dialog will then indicate the status of the download and confirm success.




After the OS update is complete, make sure to download the application to the controller again.

# Status Panels

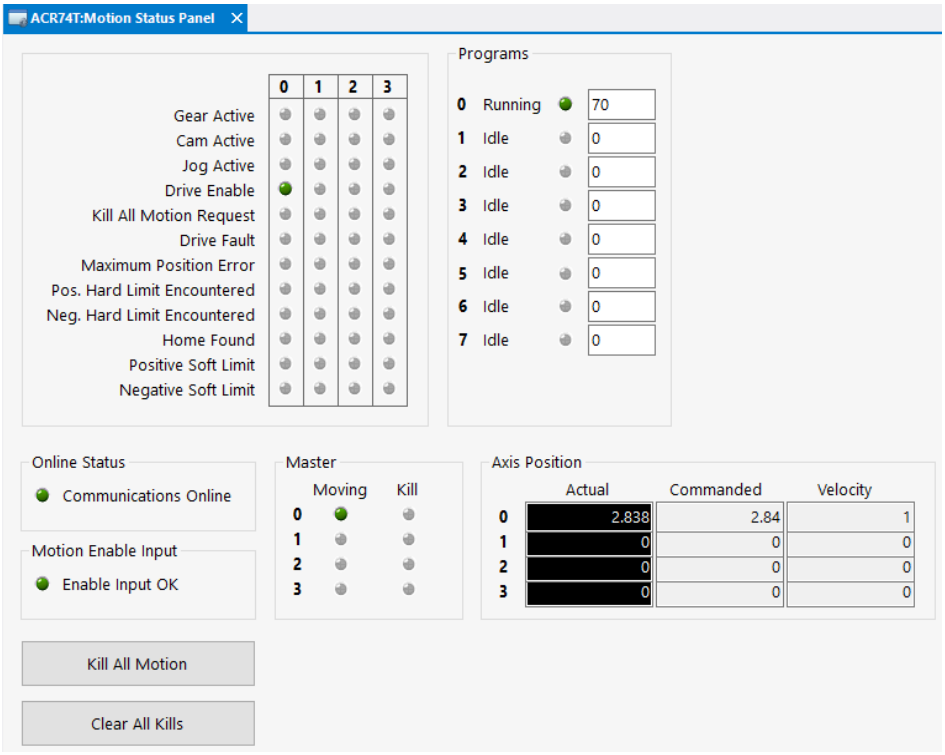
The status panels are designed to aide in commissioning and troubleshooting a machine. When problems arise, it usually helps to find a status panel that displays the data you need and pin it somewhere convenient.

The available status panels are:

- Motion Status Panel (ACR7000 family).
- Drive Status Panel (ACR7xT and ACR7xV only).
- Common Status Panel (IPA only).
- Numeric Status.
- Bit Status.
- EtherNet/IP Status Panel.
- Servo Loop Status.

## *Motion Status Panel (ACR7000 Family)*

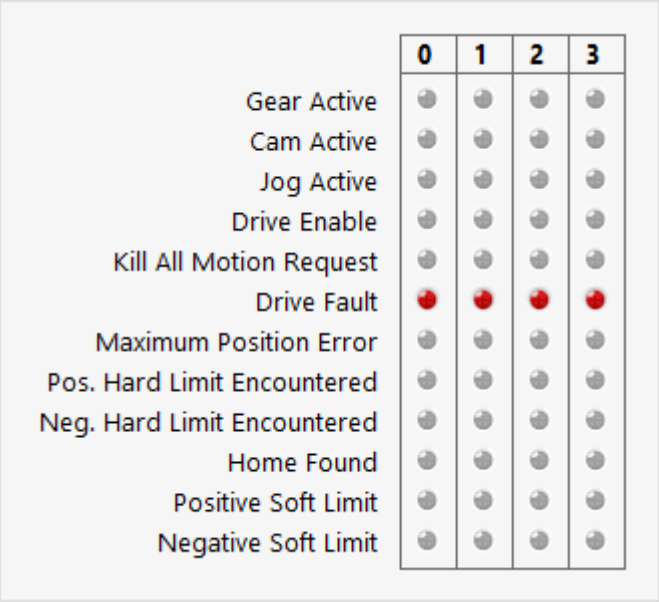
The Motion Status Panel, available for all ACR7000 models, displays basic status and fault data about all the axes at once. It also shows basic status information on programs, communications and the Enable Input.



## Axis Status Bits

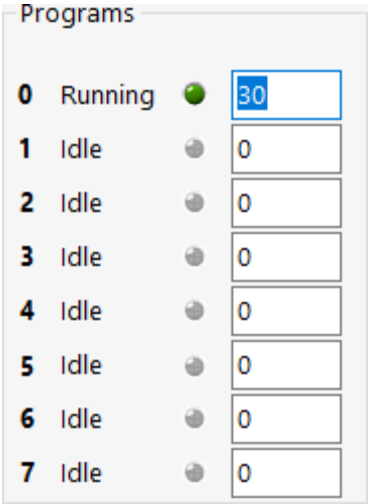
The upper left-hand subpanel displays status bits for each axis. The bits are each labeled with easy-to-understand descriptions on the left.

For more information on a particular status light, hover the mouse over the light in question. A tooltip with appear with the bit number for that indicator. The bit number can be searched in the help file for full documentation. If you want to add that bit to a Watch List, simply right-click it and click *Watch → Watch 1* (or any other Watch List number).




## Programs

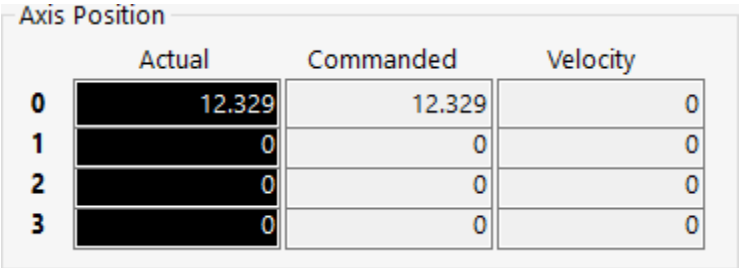
This subpanel shows which programs are currently running. It also shows their current state and line number. Line numbers typically go by 10s in ACR programs. To see a program listed with its line numbers, go into the Terminal Emulator and click *List Line Number*. Programs can have the following status codes:




- Idle. Program is not running or has stopped running. This is common if the program has not been commanded to run, in which case the line number should be zero. In the case of a program crash, the line number will show the line on which the crash occurred.
- Running. Program is executing code. The line number should be constantly changing. This is normal when the `RUN` command is given to a program. Programs can also go int run mode if they use the `PBOOT` command or if their Run Request flag has been set.
- Dwelling. The program has encountered a `DWL` command and will stay on that line until the programmed dwell time has elapsed.
- Inhibited. The program has stopped on a line and will stay there until a certain condition is met. This is caused by the `INH` and `IHPOS` commands.

## Axis Position

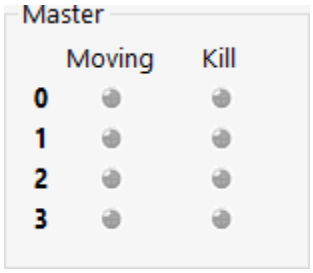
This subpanel shows the Commanded Position, Actual Position and Actual Velocity for each axis. The position values are in user units and the velocity is in user units/s. It is important to remember that Commanded Position is the sum of four



different move profilers—more on that later.

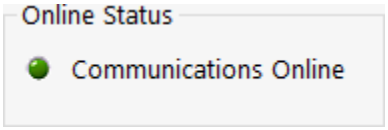
### Master

This subpanel just displays the status of the Moving and Kill All Moves flags for each master. When a Kill All Moves Flag is turned on, all master moves (e.g. `X12 Y/5`) will be prohibited until it is cleared. Click *Kill All Motion*, located in the bottom left-hand side of the Motion Status Panel, to issue a Kill All Motion request to each axis and a Kill All Moves request to each master. Click Clear All Kills to reverse that action and ready the system for motion again.



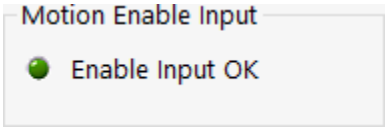

### Online Status

This subpanel just indicates whether PMM is currently connected to the controller.



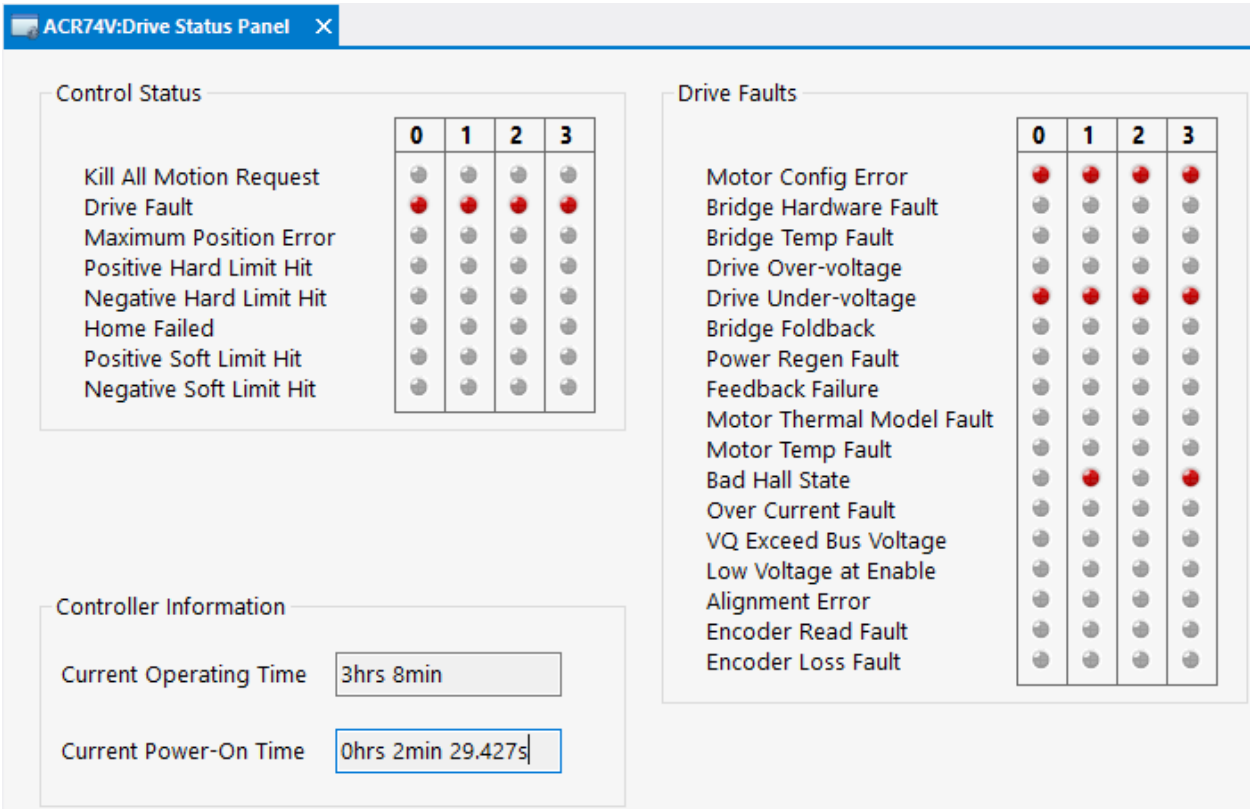

### Motion Enable Input

This subpanel indicates whether the Motion Enable Input is closed. If this input is open, none of the drives will be permitted to enable.



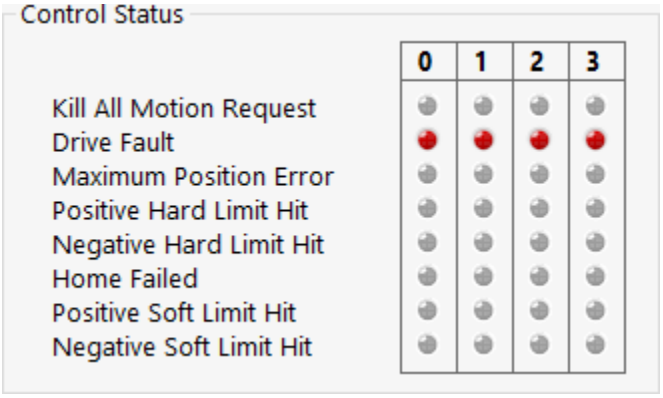

# *Drive Status Panel (ACR7xV and ACR7xT)*

The Drive Status Panel, available for ACR7xV and ACR7xT models, shows fault-related data for each of the built-in servo or stepper drives. Some of the information displayed is duplicated from the Motion Status Panel to provide the user with a complete interface for fault-finding.



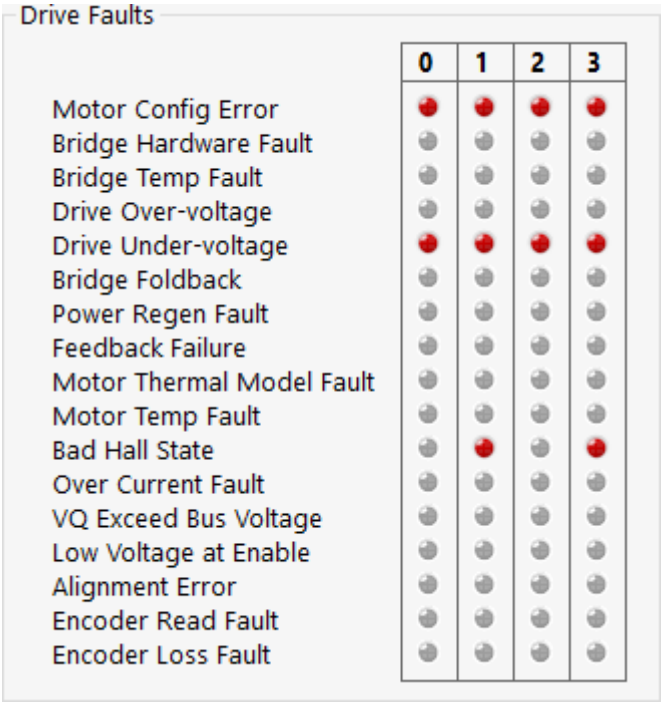

## Control Status

The Control Status subpanel shows information also available on the Motion Status Panel. Anything not related to fault conditions has been omitted. These "controller level" faults are not specific to any particular model of ACR and are also found on the IPA and older ACR9000 series.

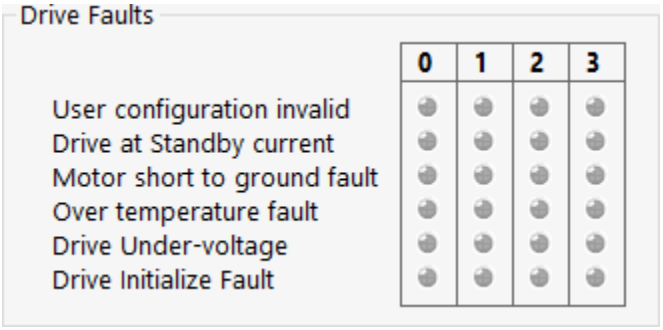


## Drive Faults

The Drive Faults subpanel varies depending on whether the axis is a stepper or servo axis. The image here shows the servo version. Some of these bits are faults and others are warnings. All of them are related to a hardware problem on the axis. If a drive fault is present, it can be cleared by issuing a `DRIVE RES` command followed by a `DRIVE ON` command. If that does not succeed, the fault condition is still present and will need to be addressed by a hardware or configuration change (e.g. reconnect encoder cable, increase available power, etc.).

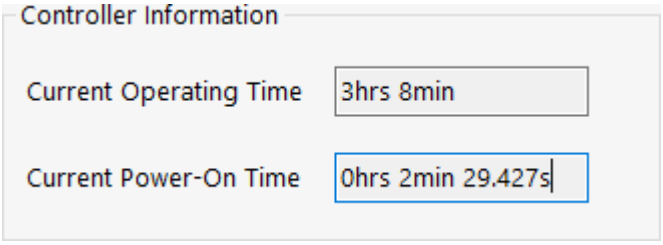



The Drive Faults subpanel for stepper axes is more limited as there are fewer possible fault causes.




## Controller Information

The Controller Information subpanel, available for the ACR7xV, shows the Current Operating Time and Current Power-On Time. The Current Operating Time shows the total powered-on time for this controller since the last factory reset (`FLASH RES`). The Current Power-On Time shows how long the controller has been powered on since the last power cycle or `REBOOT` command. These values are useful for maintenance purposes.

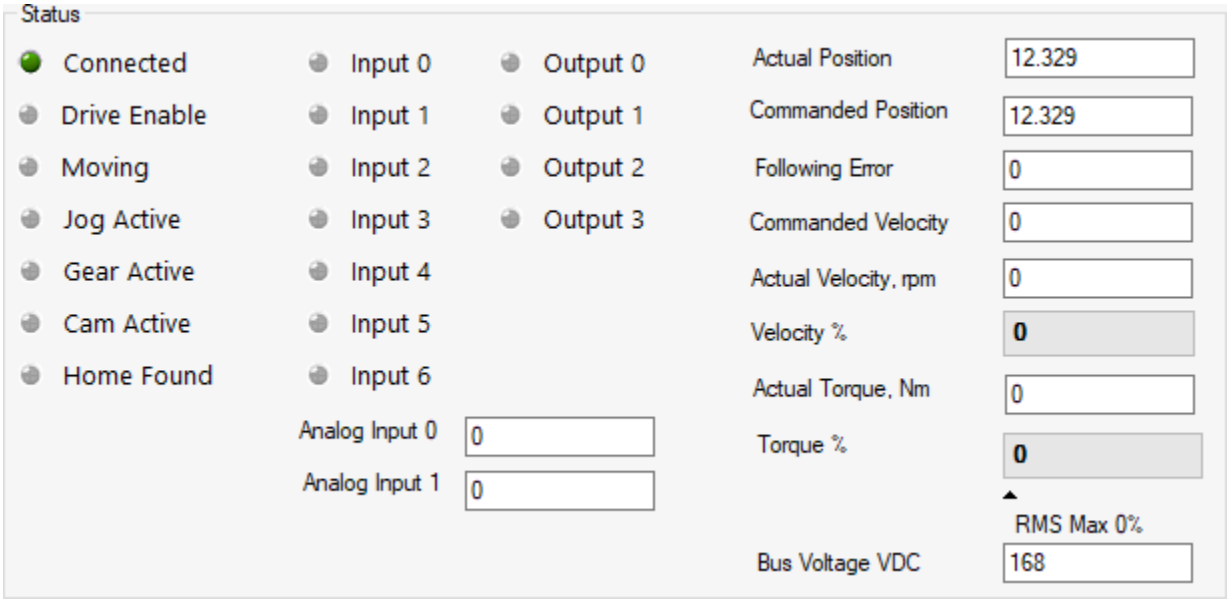


# *Common Status Panel (IPA)*

The Common Status Panel, available for the IPA, combines the Motion Status Panel and Drive Status Panel into a single interface to show the user all relevant status and fault data at once. This is feasible because the IPA only has a single axis.

## Status

The Status subpanel displays several types of information. The left-hand side shows motion status bits, the center shows I/O and the right-hand side shows various status parameters.




Most of the fields and indicators here are self-explanatory, but a few deserve special mention:

- The Moving indicator references bit 516, the In Motion bit. This bit only reflects the status of master moves (e.g. `X15`). In other words, motion can occur without this bit turning on. For instance, the command `JOG FWD X` is a jog move and would not turn on the In Motion bit.
- The Home Found indicator will turn on if the homing move succeeds. However, it will turn off again as soon as a new move is commanded.
- The Analog Input fields show the ADC input values after the offset and gain are applied. The default range is ±10 VDC, but this can be changed using the `ADC OFFSET` and `ADC GAIN` commands.
- Actual Position, Commanded Position and Following Error are shown in user units. Commanded Velocity is shown in user units/s.
- The Velocity % and Torque % fields show actual values. These fields take into account physical limits imposed by the motor, drive and available bus voltage. For instance, if the motor is designed for 340 VDC but is being run on 48 VDC, the Velocity % field will display the actual speed relative to the motor's estimated performance at the lower voltage. The Torque % field makes similar adjustments in cases where the drive has a lower rated current than the motor.
- The RMS Max field shows root mean square current usage, useful for making sure the application is not exceeding the continuous current limits of the motor or drive.

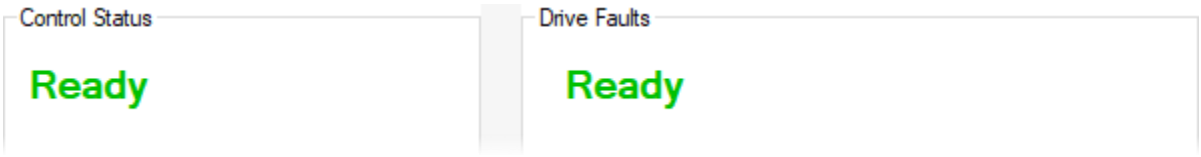
## Buttons

The Common Status Panel comes with four buttons to make basic troubleshooting tasks easier:

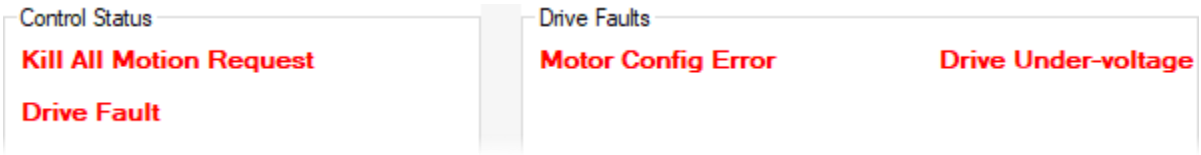
- Kill All Motion. Issues a Kill All Motion Request, which will bring the axis to a stop and prevent further motion.
- Clear All Kills. Clears any previously initiated Kill All Motion Request, allowing motion again.
- Drive Enable. Issues a `DRIVE ON` command, enabling the drive. If the drive is already enabled, this button will instead be labeled *Drive Disable* and can be used to disable the drive.
- Drive Reset. Issues a `DRIVE RES` command, necessary for clearing drive level faults (e.g. undervoltage).

## Control Status and Drive Faults

The Common Status Panel has a simplified fault reporting interface that relies on text descriptions rather than labeled indicators. If there are no problems, both panels will display "Ready" in green text.



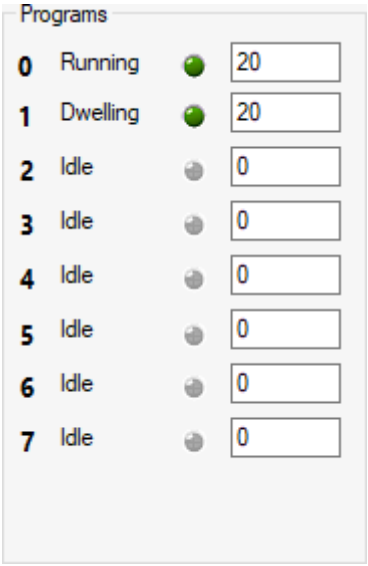

When a fault occurs, a red message will appear in the appropriate subpanel. Controller level faults like excess position error or hard limit trips are displayed under Control Status. Hardware-related faults like undervoltage or feedback faults are displayed under Drive Faults.




## Controller Information

The Controller Information subpanel shows the Current Operating Time and Current Power-On Time. The Current Operating Time shows the total powered-on time for this controller since the last factory reset (`FLASH RES`). The Current Power-On Time shows how long the controller has been powered on since the last power cycle or `REBOOT` command. These values are useful for maintenance purposes.

### Programs

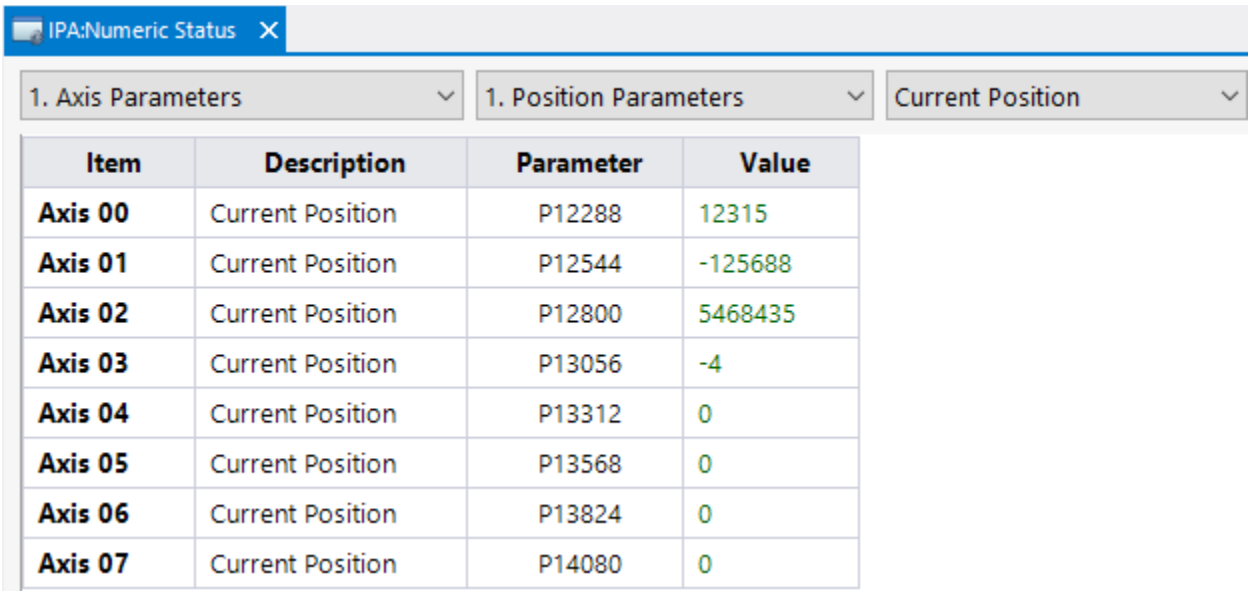



This subpanel shows which programs are currently running. It also shows their current state and line number. Line numbers typically go by 10s in ACR programs. To see a program listed with its line numbers, go into the Terminal Emulator and click *List Line Number*. Programs can have the following status codes:

- Idle. Program is not running or has stopped running. This is common if the program has not been commanded to run, in which case the line number should be zero. In the case of a program crash, the line number will show the line on which the crash occurred.
- Running. Program is executing code. The line number should be constantly changing. This is normal when the `RUN` command is given to a program. Programs can also go int run mode if they use the `PBOOT` command or if their Run Request flag has been set.
- Dwelling. The program has encountered a `DWL` command and will stay on that line until the programmed dwell time has elapsed.
- Inhibited. The program has stopped on a line and will stay there until a certain condition is met. This is caused by the `INH` and `IHPOS` commands.

# *Numeric Status*

The Numeric Status provides users with access to view any parameter in the ACR. Parameters are grouped based on function. In many cases, the parameters shown related to an enumerated resource, such as an axis, an encoder, a master, an ADC or a program. The pull-down menus can be used to select a parameter group.

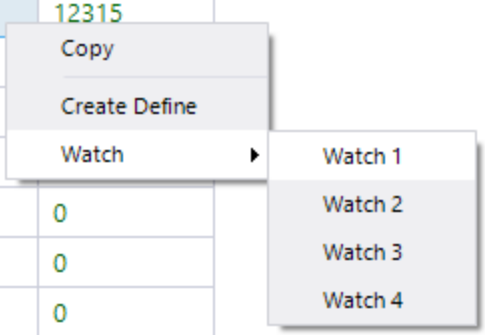



| Item | Description | Parameter | Value |
|---|---|---|---|
| **Axis 00** | Current Position | P12288 | 12315 |
| **Axis 01** | Current Position | P12544 | -125688 |
| **Axis 02** | Current Position | P12800 | 5468435 |
| **Axis 03** | Current Position | P13056 | -4 |
| **Axis 04** | Current Position | P13312 | 0 |
| **Axis 05** | Current Position | P13568 | 0 |
| **Axis 06** | Current Position | P13824 | 0 |
| **Axis 07** | Current Position | P14080 | 0 |

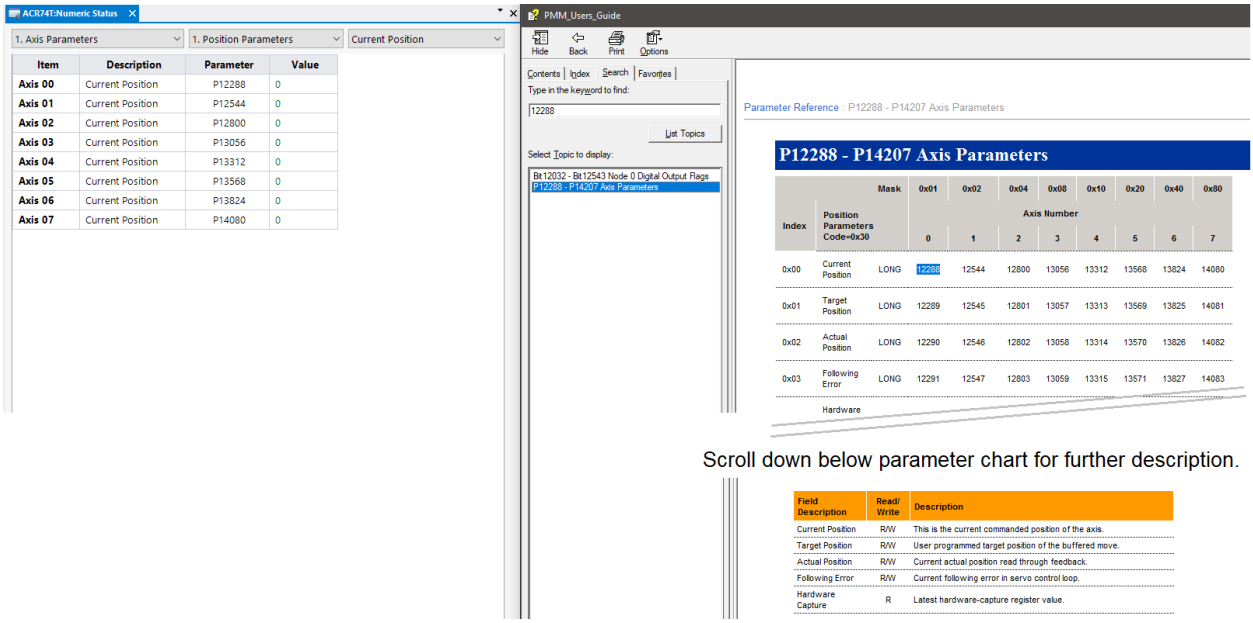
New for PMM!

Right-click a parameter to create a define for it or add it to a Watch List.

| Item | Description | Parameter | Value |
|---|---|---|---|
| **Axis 00** | Current Position | P12288 | 12315 |
| **Axis 01** | Current Position | P12544 | |
| **Axis 02** | Current Position | P12800 | |
| **Axis 03** | Current Position | P13056 | |
| **Axis 04** | Current Position | P13312 | 0 |
| **Axis 05** | Current Position | P13568 | 0 |
| **Axis 06** | Current Position | P13824 | 0 |




Axis parameter indices are separated by 256 (12290 + 256 = 12546). For example, Axis 0 Current Position is P12288 and Axis 1 Current Position is P12544. This pattern can be seen with other resources (like encoders), but the offset is not always 256. For more information on specific parameters, click *Help → Parker Motion Manager User's Guide*. Click the *Search* tab and enter the parameter number.




Numeric Status has hundreds of system parameters, such as the program line numbers.

| Item | Description | Parameter | Value |
|---|---|---|---|
| **Program 00** | Line Number | P7168 | 30 |
| **Program 01** | Line Number | P7184 | 30 |
| **Program 02** | Line Number | P7200 | 0 |
| **Program 03** | Line Number | P7216 | 168 |
| **Program 04** | Line Number | P7232 | 0 |
| **Program 05** | Line Number | P7248 | 0 |
| **Program 06** | Line Number | P7264 | 0 |

Or an axis' drive settings.

ACR74T Stepper

ACR74T:Numeric Status

Drive - Stepper Parameters | Stepper Drive Parameters | Stepper Drive Axis 0

| Item | Description | Parameter | Value |
|---|---|---|---|
| **Stepper Drive Item 00** | User Maximum motor current (Amps rms) | P7938 | 2.8 |
| **Stepper Drive Item 01** | Motor resistance (ohms) | P7939 | 0.9 |
| **Stepper Drive Item 02** | Motor inductance (mH) | P7940 | 2.5 |
| **Stepper Drive Item 03** | Standby Current Percentage | P7944 | 50 |
| **Stepper Drive Item 04** | Standby Current Delay (msec.) | P7945 | 0 |
| **Stepper Drive Item 05** | Micro-steps per full step (power of 2) | P7946 | 256 |
| **Stepper Drive Item 06** | Configuration error code | P7947 | 0 |
| **Stepper Drive Item 07** | Drive write value | P7948 | 0 |
| **Stepper Drive Item 08** | Drive read value | P7949 | 0 |

ACR74V Servo

ACR74V:Numeric Status

Servo Drive Parameters | Servo Drive Parameters | Servo Drive Axis 0

| Item | Description | Parameter | Value |
|---|---|---|---|
| **Axis 00** | Feedback Resolution | P28674 | 4000 |
| **Axis 00** | Continuous Current | P28704 | 1.99 |
| **Axis 00** | Continuous Current Derating | P28705 | 13.2 |
| **Axis 00** | Peak Current | P28706 | 5.98 |
| **Axis 00** | Motor Inductance | P28707 | 5.98 |
| **Axis 00** | Motor Inductance Factor | P28708 | 0.7 |
| **Axis 00** | Motor Maximum Temperature | P28709 | 125 |
| **Axis 00** | Winding Resistance | P28710 | 9.65 |
| **Axis 00** | Motor Rated Speed | P28711 | 125 |
| **Axis 00** | Motor Pole Pairs | P28675 | 2 |
| **Axis 00** | Motor Damping | P28712 | 36.41 |
| **Axis 00** | Motor Rotor Inertia | P28713 | 132.2 |
| **Axis 00** | Linear Motor Pole Pitch | P28715 | 0 |
| **Axis 00** | Motor Maximum Torque | P28717 | 4.09 |
| **Axis 00** | Motor Constant, Ke | P28714 | 50.67 |
| **Axis 00** | Torque Scaling | P28718 | 4.09 |
| **Axis 00** | Encoder Polarity | P28776 | 0 |
| **Axis 00** | Invert Halls | P28678 | 0 |
| **Axis 00** | Hall Only Commutation | P28775 | 0 |
| **Axis 00** | Motor Ambient Temperature | P28719 | 25 |
| **Axis 00** | Motor Thermal Resistance | P28720 | 0.37 |
| **Axis 00** | Motor Thermal Time Constant | P28721 | 23.33 |
| **Axis 00** | Winding Time Constant | P28722 | 0.33 |
| **Axis 00** | Disable Thermal Switches | P28679 | 0 |
| **Axis 00** | Motor Velocity Limit | P28716 | 125 |
| **Axis 00** | Encoder Commutation Offset | P28725 | 0 |
| **Axis 00** | Serial Encoder Valid Bits | P28769 | 0 |

**NOTE: Not all parameters are used in all products. For example, the ACR7xC controller does not have integrated steppers and thus the stepper drive parameters would all be 0.**

New for PMM!

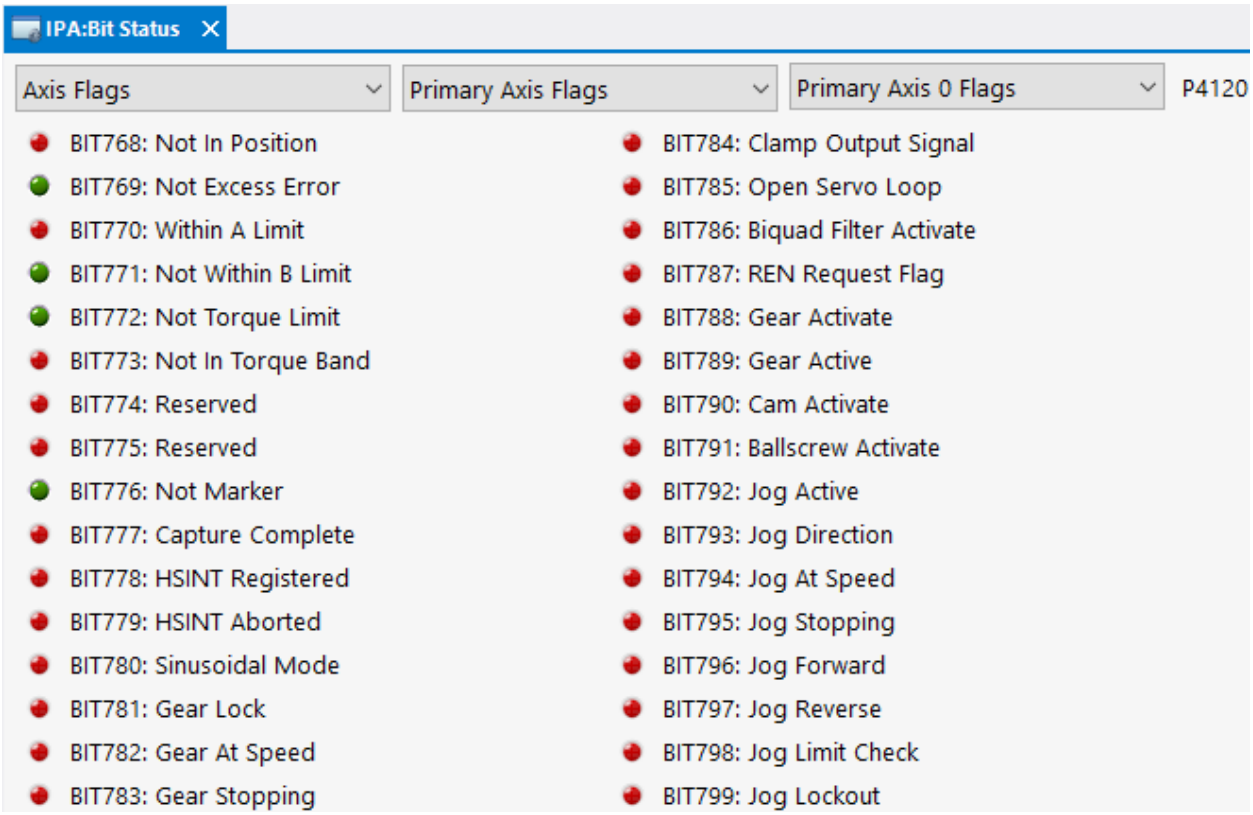
User parameters are now in the Numeric Status. This includes User Doubles (P0-P4095), User Non-Volatile Longs (P38912-P39167) and User Non-Volatile Floats (P39168-P39423).

User Variables | User Doubles | Group 0

| Item | Description | Parameter | Value |
|---|---|---|---|
| **User Variable P000** | User Variable | P00 | 0 |
| **User Variable P001** | User Variable | P01 | 10 |
| **User Variable P002** | User Variable | P02 | 3.14159 |
| **User Variable P003** | User Variable | P03 | 26.2 |
| **User Variable P004** | User Variable | P04 | 524288 |
| **User Variable P005** | User Variable | P05 | 1234.567 |

# Bit Status

The Bit Status shows the status of every bit in the ACR. It works much like the Numeric Status. Indicators show green for on (a.k.a. set or non-zero) and red for off (a.k.a. clear or zero).

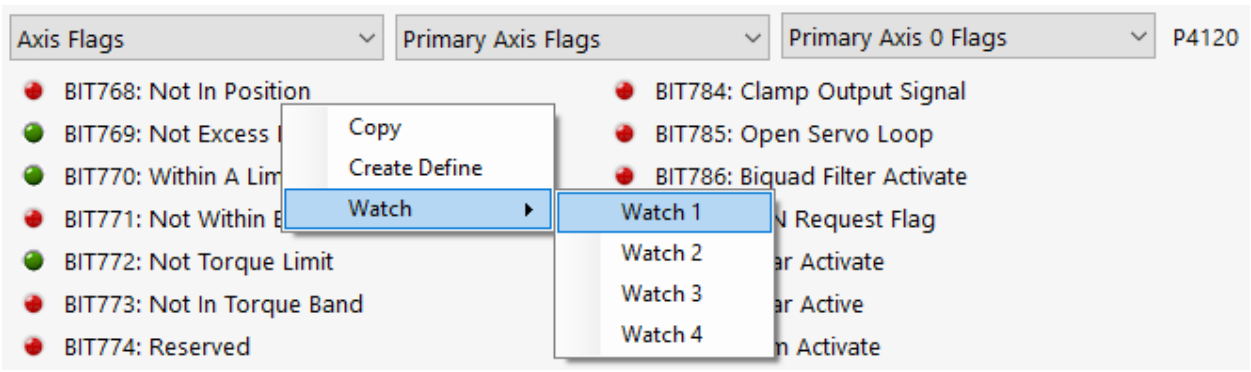



The parameter for all 32 bits is also shown. For example, P4120 is the 32-bit integer containing the Axis 0 Primary Axis Flags. Bit 768 is bit 0 of P4120's 32 bits. Bit 799 is bit 31 of P4120.

Axis bit indices are separated by 32, so if the Not In Position bit for Axis 0 is bit 768, the Not In Position bit for Axis 1 is bit 800.

New for PMM!

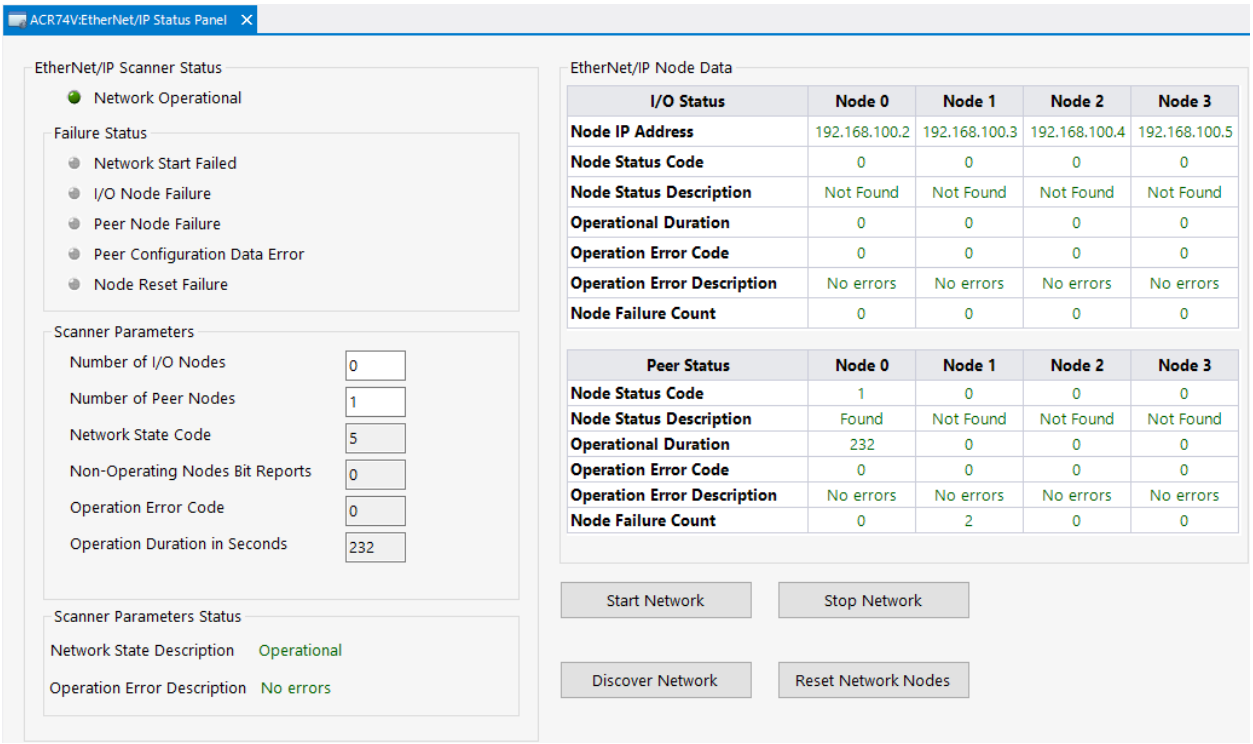
Users can quickly add specific bits to their Defines list or Watch Lists by right-clicking from this panel.




Similar to parameters, if needing a better description of a specific bit, search the online help file with the bit number and scroll down past the bit table.

# Ethernet/IP Status Panel

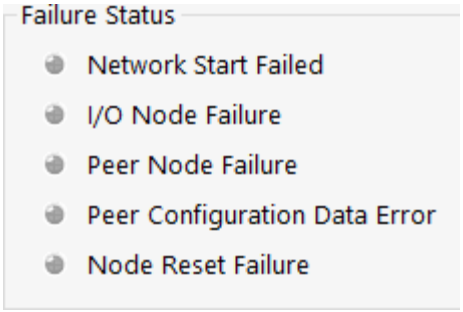
The EtherNet/IP Status Panel shows detailed status information for the controller's EtherNet/IP adapter, master and peer-to-peer connections.




The EtherNet/IP Status subpanel houses fields and indicators pertaining to overall network status. The Network Operational indicator at the top shows whether the EtherNet/IP network is running and exchanging data. Note that this does not mean the network is free of errors. It is possible for the network to start even if not all of the nodes are found.

## Failure Status

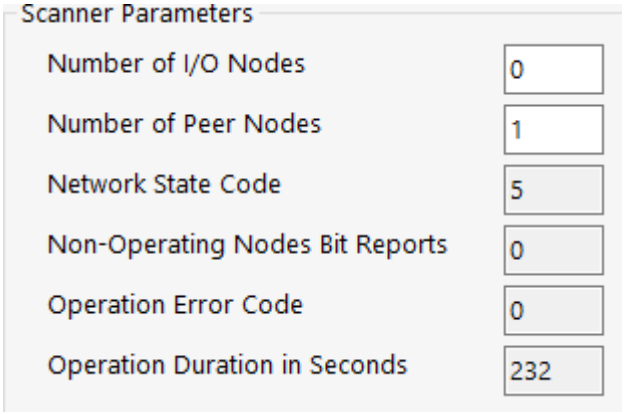
This subpanel displays indicators that address specific failure conditions:




- Network Start Failed. Errors occurred during startup that prevented operation.
- I/O Node Failure. Network failure occurred on a specific I/O node.
- Peer Node Failure. Network failure occurred on a specific peer (another ACR).
- Peer Configuration Data Error. Configuration data invalid for peer connection (e.g. parameter range exception).
- Node Reset Failure. Node was unable to reset connection.

## Scanner Parameters

This subpanel displays basic network status data:



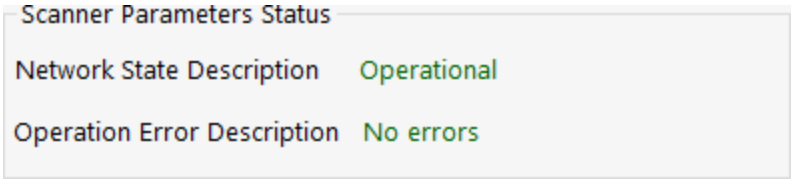

- Number of I/O Nodes. This is the number of connected PIO-363 (Wago 750-363) EtherNet/IP bus couplers. Up to four connections are supported.
- Number of Peer Nodes. This is the number of connected ACR controllers. Up to four connections are supported.
- Network State Code. Represents overall network status. Typical states are:
  - 0—Reset or Not Active.
  - 5—Operational.
  - 6—Stopped.
- Non-Operating Nodes Bit Reports. Each non-operating I/O Node or Peer Node will trigger a bit to turn on in this parameter. I/O Nodes start at bit 0 and Peer Nodes start at bit 16.
- Operation Error Code. Network error code (0 indicates no error). Full listing below.
- Operation Duration Time in Seconds. Time elapsed since network was started.

### *Operation Error Code Descriptions*

| Value | Description | Value | Description |
|---|---|---|---|
| 0 | No errors | 11 | I/O Node Online, but no response |
| 1 | Invalid user supplied I/O Node count | 12 | I/O Node error response |
| 2 | Invalid user supplied Peer Node count | 13 | UCMM data range |
| 3 | Error in user supplied external node data | 14 | Internal error in EtherNet/IP cycle start |
| 4 | Invalid IP address | 15 | Invalid number of Peer parameter groups |
| 5 | Error in UCMM transfer | 16 | Invalid Peer parameter direction |
| 6 | UCMM request timeout | 17 | Invalid number of parameters in a Peer group |
| 7 | Excess I/O on node or system | 18 | Invalid parameter number in a Peer group |
| 8 | Unknown I/O node vendor | 19 | Excess inputs or outputs within a single Peer |
| 9 | Unexpected I/O node device type | 20 | Excess inputs or outputs for total Peer collection |
| 10 | I/O Node offline | | |

## Scanner Parameter Status

This is a text summary of the network status, specifically the Network State Code and Operation Error Code.




## EtherNet/IP Node Data

This subpanel, located on the right-hand side, houses tables displaying node-specific status data. This is helpful when troubleshooting node-specific problems.

The top table displays the status of the I/O nodes:

| I/O Status | Node 0 | Node 1 |
|---|---|---|
| Node IP Address | 192.168.100.2 | 192.168.100.3 |
| Node Status Code | 0 | 0 |
| Node Status Description | Not Found | Not Found |
| Operational Duration | 0 | 0 |
| Operation Error Code | 0 | 0 |
| Operation Error Description | No errors | No errors |
| Node Failure Count | 0 | 0 |

- Node IP Address. IP address of the node. Note that every IP address on a network must be unique—this address should not be the same as the controller's address.
- Node Status Code. Typical codes are 0 (not found), 1 (found) and 2 (found then lost).
- Node Status Description. Text description reflecting the Node Status Code.
- Operational Duration. Seconds since the node was connected.
- Operation Error Code. 0 means no error.
- Operation Error Description. Text description of Operation Error Code.
- Node Failure Count. Number of dropped packets recorded since the node connected.

The bottom table displays the status of the peer nodes:

| Peer Status | Node 0 | Node 1 |
|---|---|---|
| Node Status Code | 1 | 0 |
| Node Status Description | Found | Not Found |
| Operational Duration | 232 | 0 |
| Operation Error Code | 0 | 0 |
| Operation Error Description | No errors | No errors |
| Node Failure Count | 0 | 2 |

- Node Status Code. Typical codes are 0 (not found), 1 (found) and 2 (found then lost).
- Node Status Description. Text description reflecting the Node Status Code.
- Operational Duration. Seconds since the node was connected.
- Operation Error Code. 0 means no error.
- Operation Error Description. Text description of Operation Error Code.
- Node Failure Count. Number of dropped packets recorded since the node connected.

## Controls

There are also four buttons on this panel to provide basic control over the network:

- Start Network. Attempt to start the network. This will connect to any configured nodes and begin exchanging data.
- Stop Network. Stops data updates and disconnects from all nodes.

- Discover Network. Checks to verify the availability of all nodes. Good for verifying network integrity before starting the network.
- Reset Network Nodes. Reset connections to all nodes and restart the network.

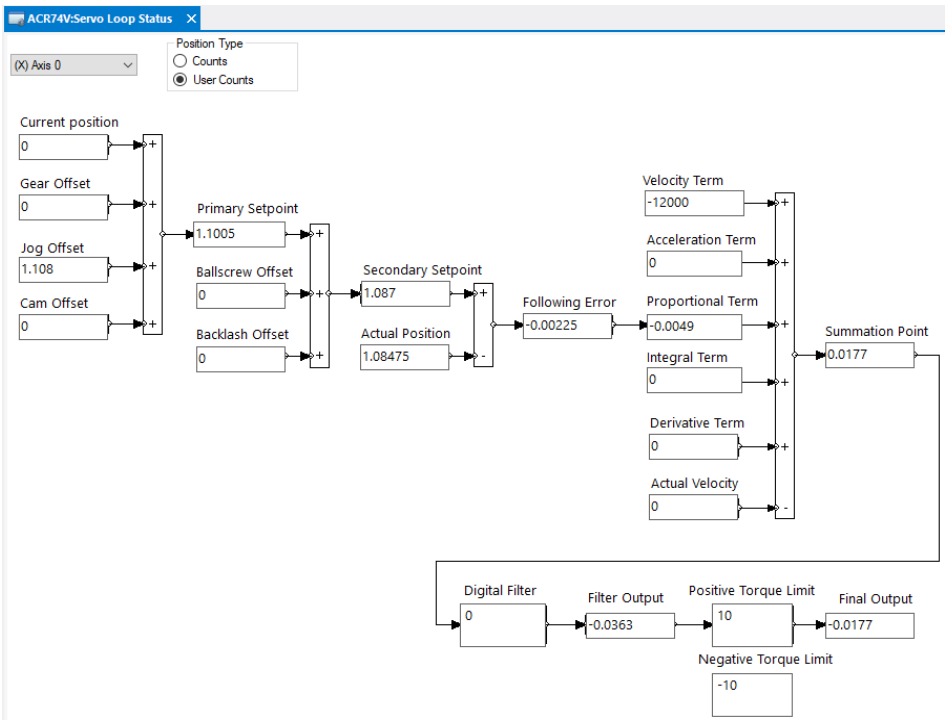
Other useful data related to EtherNet/IP can be found in the Numeric Status and Bit Status. In particular, the values of actual I/O data and peer-to-peer data can be found there.

# *Servo Loop Status*

The Servo Loop Status panel gives users immediate visibility as to the different types of motion being commanded for an axis. Interpolated, Gear, Jog and Cam together are the Primary Setpoint. Backlash and Ballscrew compensation are added to generate the Secondary Setpoint. Current Position is the commanded position from interpolated MOV commands.

Actual Position is based on the servo feedback, typically a rotary or linear encoder. Following error is generated by subtracting the Secondary Setpoint from Actual Position. This is provided to the servo loop, which processes the error and multiplies it by gains to generate the Proportional Term, Integral Term and Derivative Term. The Velocity Term and Acceleration Term are feed-forward values. The Summation Point is the just the sum of all of these terms.

The output from the Summation Point is processed by a lowpass and a notch filter. The result is clamped to ±10 VDC or a lower limit if configured by the user. The ±10 VDC range represents the full range from peak positive torque to peak negative torque (force in the case of linear motors). This value is proportional to the current that will be delivered to the motor.




New for PMM!

The Servo Loop Status can display position and velocity data in encoder counts, but can also display it in user units, saving the effort of translating counts into meaningful information.

# Scopes

There are four independent scopes in PMM:

- Oscilloscope 1.
- Oscilloscope 2.
- Strip Chart.
- XY Plot.

These scopes are not tied to any specific purpose and can be used for general troubleshooting. Most of the tools will be familiar to anyone who has already used the Servo Tuner.

# *Common Tools*

The Scopes are all built on a common interface with variations for each use case. Those common tools are covered here. The scopes themselves are covered afterward.

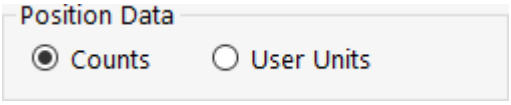
## Channels

By default, the four channels are set up as follows:

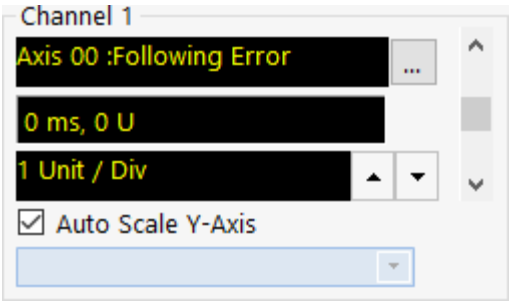
- Channel 1 shows Following Error in yellow. Units are encoder counts.
- Channel 2 shows Current Jog Velocity in green. Units are encoder counts per second.
- Channel 3 shows Final Output Signal in orange. Units are ±10 and represent the torque command in volts.
- Channel 4 shows Secondary Setpoint in purple. Units are encoder counts.

New for PMM!

In addition to controls specific to each channel, there is a global setting for all position-related parameters to be graphed in encoder counts (default) or in user units.

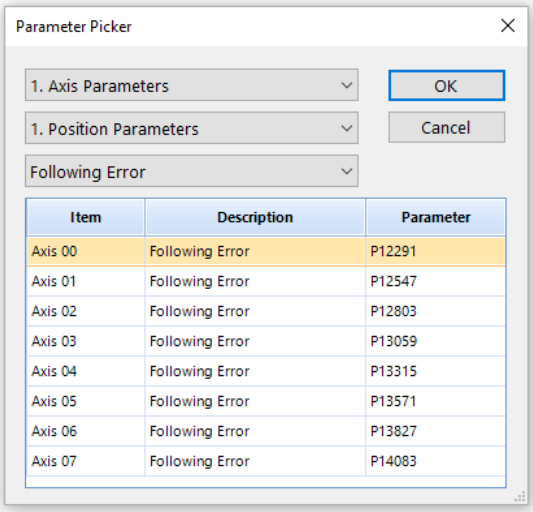



Each channel has several controls:




- The top field displays the parameter being graphed. This can be changed to any parameter desired using the "…" button to the right of the field (details below).
- The middle field displays the horizontal and vertical shift for this channel. The vertical shift can be altered using the vertical slider at the right. See Timebase for details on the horizontal shift.
- The bottom field displays the vertical scale (default 1 unit/div). This can be altered using the up and down arrow buttons to the right. Check *Auto Scale Y-Axis* (checked by default) to make the graph fit the available vertical space. This option is usually preferred as it makes the data easy to read.

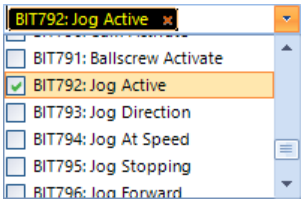
When the user clicks the "…" button to select a new parameter, the Parameter Picker dialog appears. This dialog helps the user drill down to a parameter of interest by using three pull-down menus. The top menu selects the parameter group, the second menu selects the subgroup and the third selects the individual parameter type. The list at the bottom breaks a specific parameter type (Following Error in this case) down into enumerated options, often based on axis number (otherwise encoder number, stream number, ADC number, etc.).




The Parameter Picker dialog is consistent across PMM and is also used in the Servo Tuner. It is conceptually very similar to the Pick A Bit dialog that serves the same purpose for bits. The same pull-down menus used in the Parameter Picker are also used in the Numeric Status panel.

New for PMM!

When a flag parameter is selected (P4096-4375), the bottom pull-down makes it possible to select a specific bit (or bits) to watch in the scope.



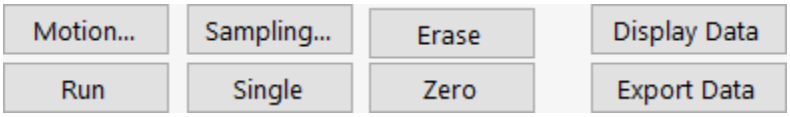

## Timebase

The Timebase panel controls the graph's time (horizontal) axis. It consists of several tools:

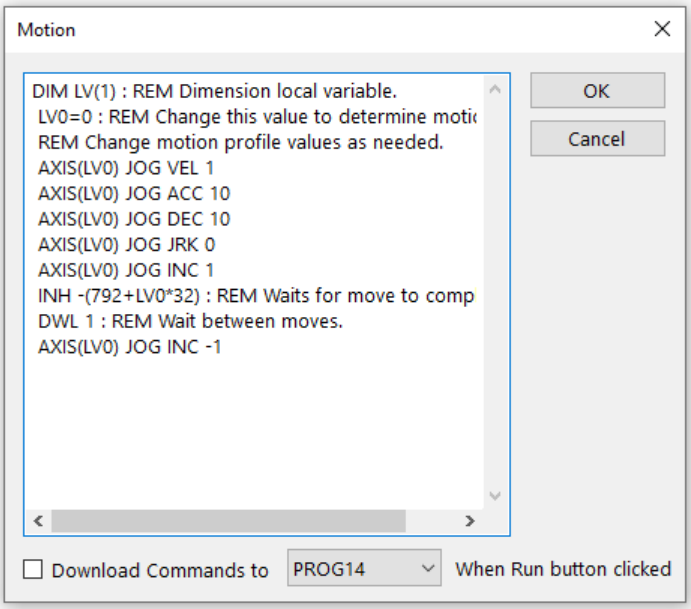

- The time/division indicator shows the length of time represented by one division on the horizontal axis. This can be changed using the up and down buttons to its left.
- The slider beneath the time/division indicator can be used to scroll data in the graph back and forth horizontally.

## Controls

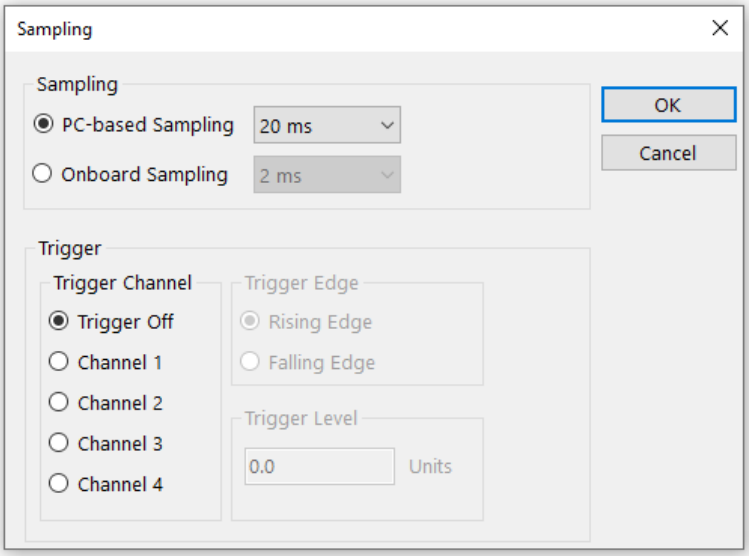
Most scopes have two rows of buttons near the bottom left-hand corner that perform major functions, although there will be variations on exactly which buttons are present.




Clicking *Motion* provides a dialog where the user can write a small snippet of code. This code will be sent down the controller and executed during a test. The default code just does an incremental jog move on an axis defined by the user. To turn off the motion code, simply uncheck the *Download Commands* box. Some users may find it useful to alter the program to which the commands are sent (particularly if interpolated motion is required), which can be done with the program pull-down at the bottom of the dialog.




Clicking *Sampling* opens the sampling dialog. Here, the user can select *PC-based Sampling* (default) or *Onboard Sampling*.




PC-based sampling means that PMM will request the parameter value over Ethernet at the specified rate. The sample data is transmitted as needed without buffering. This is convenient and does not impose a memory burden on the controller, meaning the graph can store very large data samples. However, PMM does not permit sampling faster than 20 ms with this option to avoid taxing network and processor resources on the controller and the user's PC.

Onboard sampling means that the controller will allocate a memory buffer for the data it needs to take in advance. When the user clicks *Single* or *Run*, the controller will store the data it acquires in the buffer and transmit it all at once after the test is finished. The main advantage of this option is that it allows the user to acquire data at a

faster interval (all the way down to the servo period). However, ACR controllers have limited memory and large data samples are not always possible. If there is not enough memory available to run the test with onboard sampling, PMM will log the error message "failed to allocate program memory for sampling buffer" to the Messages window.

**NOTE: The Run button requires onboard sampling.**

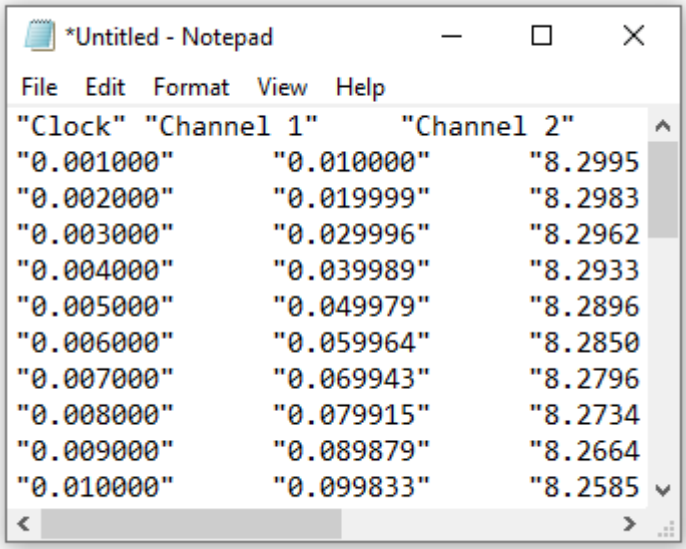
The approximate amount of memory in bytes required to run a test move with onboard sampling can be calculated using the following formula:

$$M_{bytes} = 4N_{channels}\left(\frac{t_{test}}{t_{sample}}\right)$$

$N_{channels}$ is the number of channels in use, $t_{test}$ is the length of time visible on the horizontal axis in seconds and $t_{sample}$ is the sample time in seconds. If the sample time is set to *Servo Period*, the actual sample time depends on the controller in use. For ACR7000 controllers, the default servo period is 500 µs. For the IPA, it is 250 µs.

The other buttons have simpler functions and are explained below:

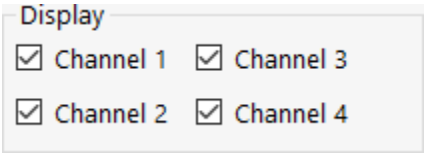
- Clicking *Display Data* displays all of the captured data in a textual format, which makes it easy to copy the data to other applications like Microsoft Excel.
- Clicking *Export Data* directly saves the data to a text or CSV file.
- Clicking *Run* will run the Motion code over and over, producing graphs over and over. This is useful for viewing how an issue reacts over many cycles. It is helpful in troubleshooting intermittent problems.
- Clicking *Single* will run the Motion code exactly once and produce one graph. This is the most common way to use the oscilloscope.




- Clicking *Zero* resets the time slider as well as the vertical slider for each channel, an effect only noticeable if Auto Scale is disabled.
- Clicking *Erase* erases all data from the graph.

## Display

The Display panel has four checkboxes that allow the user to show or hide each channel on the graph.

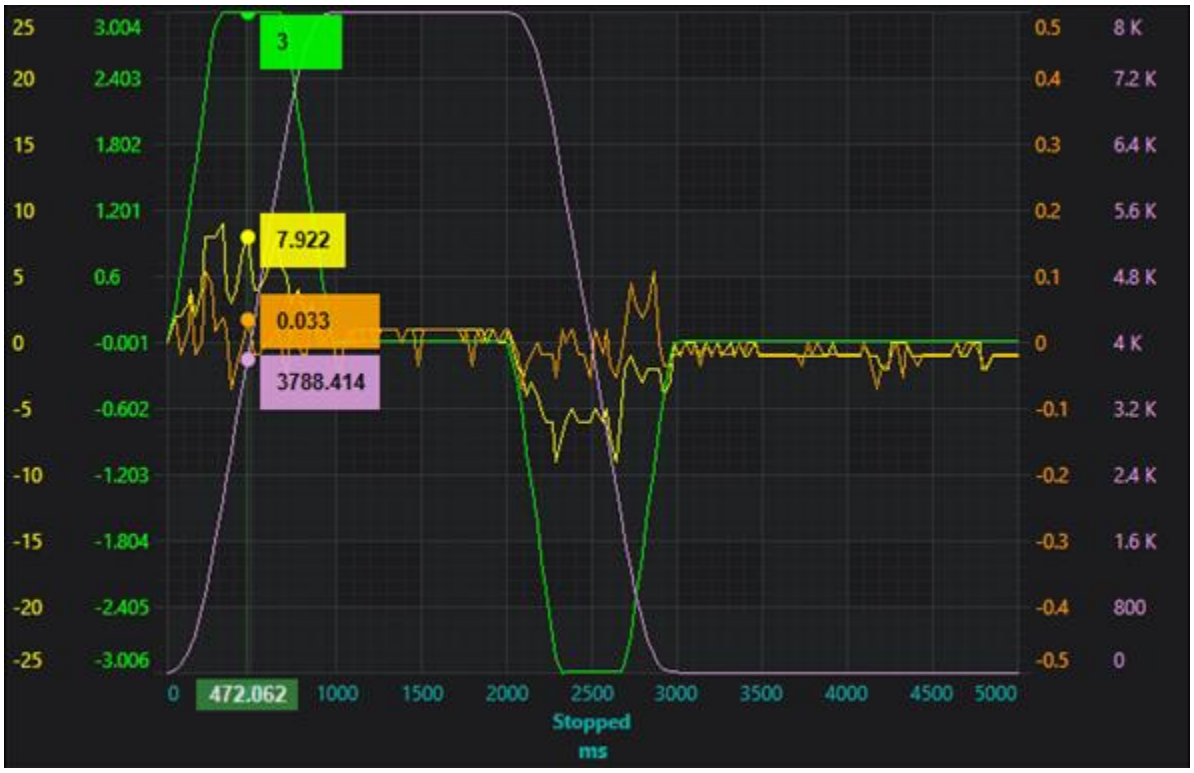


## The Scope

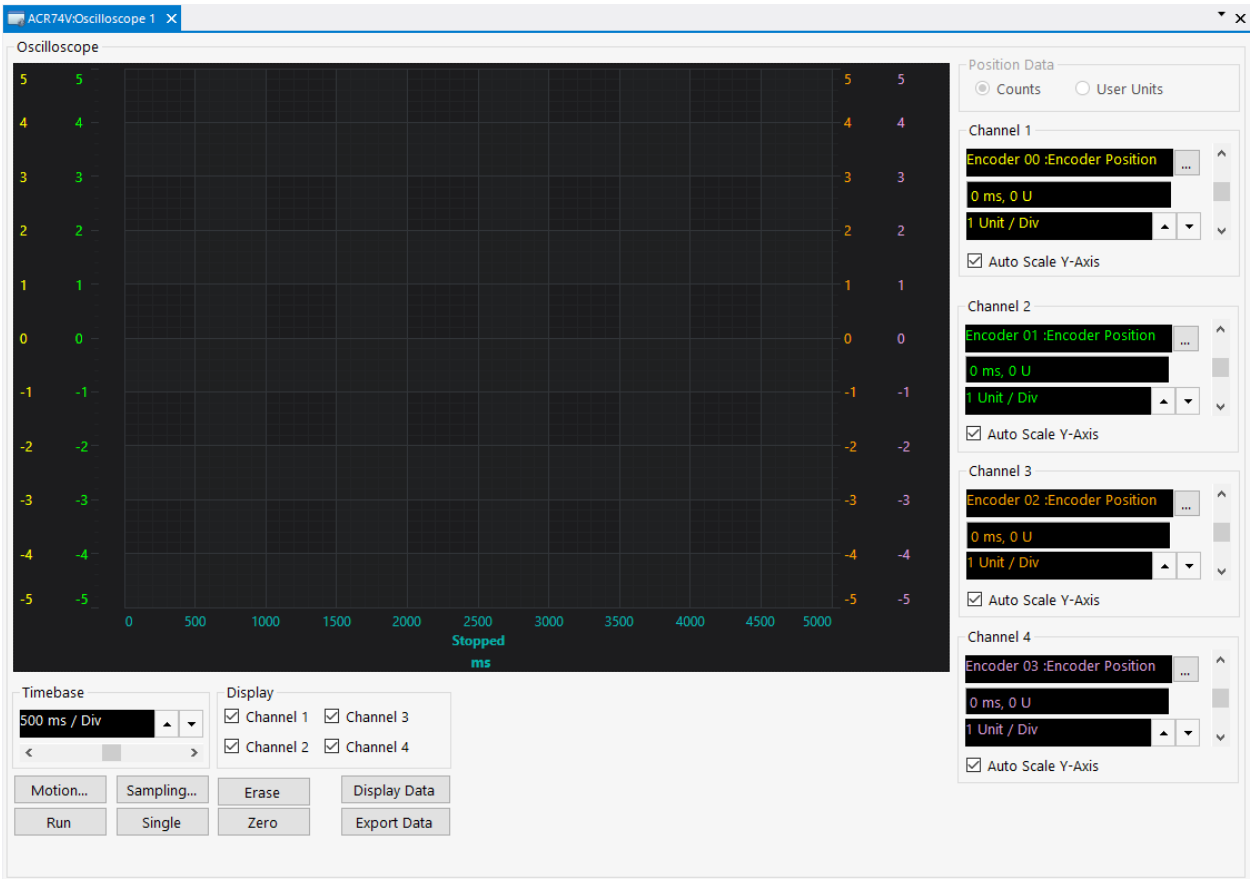
The graphical scope is the central feature of each scope tool and shows data captured from the controller during a test. This helps users visually understand what their axis is doing during the test. It is common to graph parameters like Following Error, Secondary Setpoint, Actual Torque and other control loop parameters.

New for PMM!

Data channels display their current values when sweeping the cursor over the scope. This makes it easy to correlate specific channel values with specific times. One of the biggest new features to the Servo Tuner in PMM is Auto Scale (on by default), which makes the scope much easier to read.

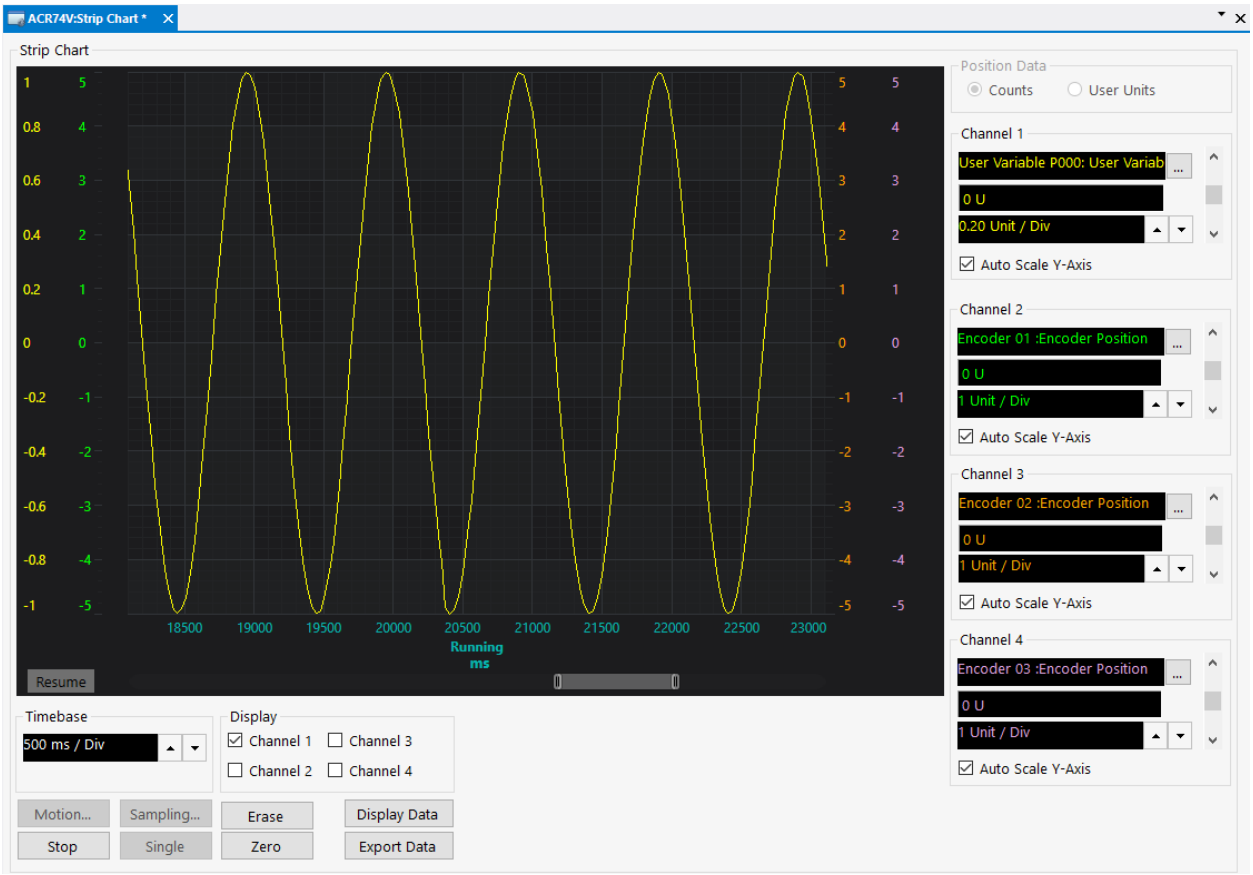


# *Oscilloscope*




The Oscilloscope is the most flexible scope tool in PMM. For convenience, there are two of them. This allows users to monitor or troubleshoot two entirely different issues without needing to constantly reconfigure their scope. The Oscilloscope is very similar to the Servo Tuner and has access to all of the tools mentioned previously. It can be considered the basic form of all the scopes.

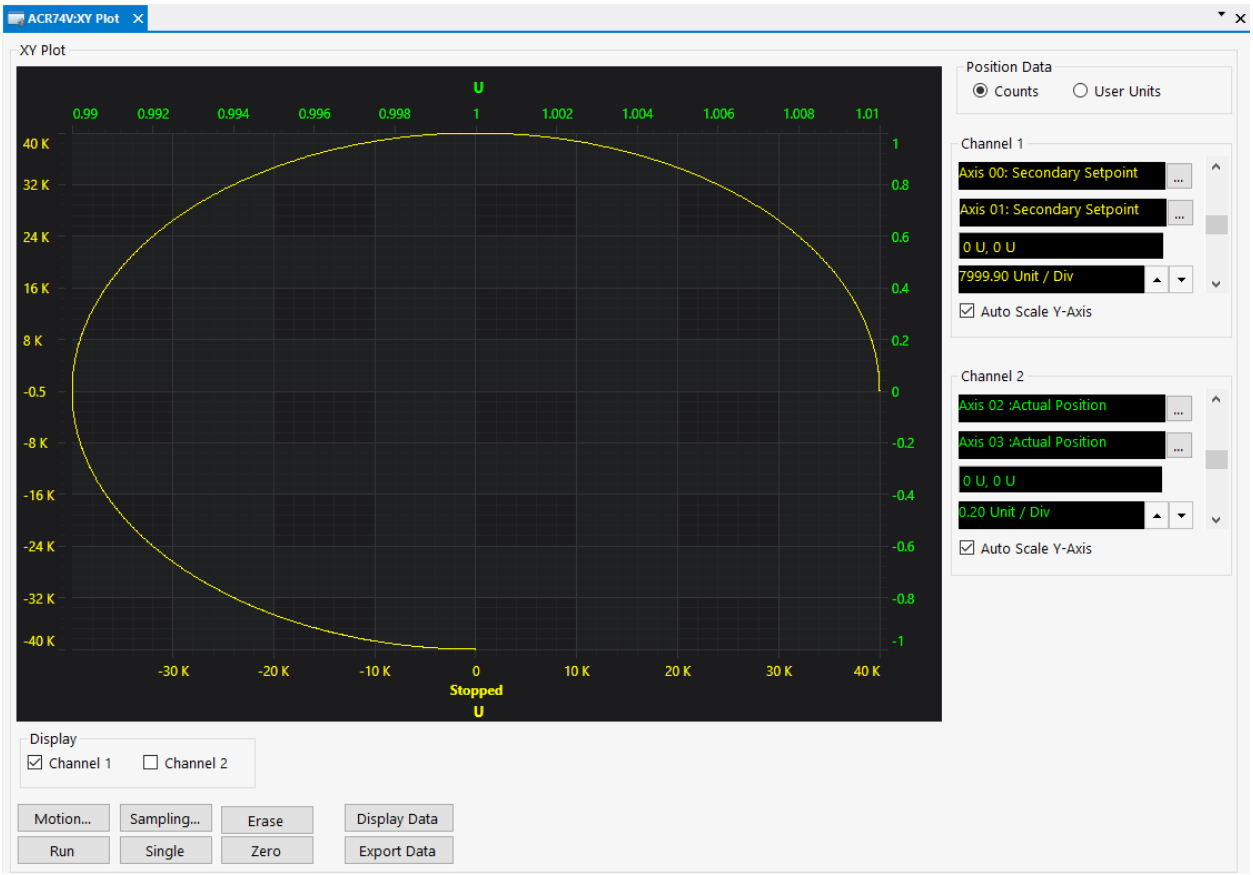
# Strip Chart




The Strip Chart is like the Oscilloscope, but is designed to monitor signals for longer periods. Because it is meant for monitoring slower signals, onboard sampling is not an option. The Strip Chart only uses PC-based sampling.

The scope for the Strip Chart comes with an extra slider at the bottom. This slider can be used to select a subset of the total elapsed time for viewing. If the user changes the position of the slider, the scope will pause at the desired section, but will keep collecting data. Clicking *Resume* will put the Strip Chart back in "scrolling" mode.

# XY Plot




The XY Plot replaces the time axis with another value axis. It is designed to compare two signals that have a relationship, like the positions of an X and a Y axis.

The XY Plot only has two channels instead of the usual four. Each channel maps one parameter to the horizontal axis (top parameter) and another to the vertical axis (bottom parameter). The rest of the tools work normally.

# CHAPTER 2
# ACR Basics

# ACR Basics

The AcroBASIC programming language accommodates a wide range of needs by providing basic motion control building blocks, as well as sophisticated motion and program flow constructs.

The language comprises simple ASCII mnemonic commands each on its own line or separated by a delimiter.

Let us start by taking a look at a basic program and what each line does:

| | |
|---|---|
| `PROGRAM` | ← Starts program definition. First line of any program. Valid AcroBasic commands appear in blue, but that is not shown here for formatting reasons. |
| `DRIVE ON X` | ← Enable the X axis motor. |
| `RES X` | ← Reset the X position to 0. |
| `ACC 10 DEC 10 STP 10 VEL 1` | ← Sets acceleration/deceleration/stop/velocity. |
| `X10` | ← Go to 10 position on X. Units based on config wizard scaling. |
| `X0` | ← Return to starting position of 0 on X axis. Controller waits until X is at 10 before returning. |
| `ENDP` | ← End of program. Must always be last line of a program. |

Note most commands are on their own lines but `ACC`, `DEC`, `STP` and `VEL` are all used by the motion calculator simultaneously and thus can be on one line, saving vertical programming space.

The AcroBASIC programming language uses a parent child approach. A parent can have child statements. A child statement is considered a sub-statement of the parent.

Parent Command `DRIVE ON` `X` Child Command

You can issue many parent statements alone—some provide the current status related to that particular command, others perform an action. For example, issuing the `VEL` command online with the controller in the terminal emulator at the Program 0 prompt provides the velocity setting. Conversely, issuing the `PROGRAM` command initiates defining a program.

## *Delimiter*

Commands can be on their own line; a carriage return or line feed at the end of a line separates one command from the next. Or, you can put multiple commands on the same line separated with a "space-colon-space" (" : "). This can be used to separate two commands on the same line to save vertical space.

```
DRIVE ON X : REM Enable the X axis motor.
X0         : PRINT "MOVING TO 0 POSITION"
```

The multiple spaces between X and : are extra and okay; the minimum is one space, followed by a colon, followed by one space. This allows programmers to align their program notes for readability. `REM` is a remark for programmer's notes but is technically a command and thus needs to be separated from `DRIVE ON X` with the delimiter. Though commands are on the same line, they are executed sequentially left to right.

```
JOG REV Y Z : JOG FWD X : PRINT "RETRACTING YZ, ADVANCING X"
```

More than two commands can be on the same line, but this can be harder to read in larger programs.

# *Remarks*

Remarks are for programmers to add notes on the purpose of different sections or lines of code. They can be on their own line or at the end of line with space-colon-space.

`REM` Remarks are stored within the controller. Remarks in PMM automatically change to green in the program editors.

By using an apostrophe (`'`) in place of `REM`, the controller strips comments on downloading the program. This can help minimize program space within the controller. This originated from memory limitation of controllers many years ago and is not a limitation for new controllers such as the ACR7000 or IPA controllers. Apostrophe comments must appear on their own line.

Remarks cannot be on `PROGRAM` or `ENDP` lines or on program labels.

No commands can be at the end of a remark. Programmer comments must be on their own line or at the end of line. Let us take the first program and add remarks.

```
PROGRAM
' My first program.      This would not be stored in the controller
DRIVE ON X               : REM Enable the X axis motor.
RES X                    : REM Resets current X position to 0.
REM Each comment line must have REM or ' at the beginning.
REM These would be stored within the controller.
REM Sets accel, decel, stop and velocity.
ACC 10 DEC 10 STP 10 VEL 1
X10                      : REM Move to 10 on X axis.
X0                       : REM Return to start on X axis.
ENDP
```

This program could be inserted into Program 0 and downloaded as is. The program could be run several ways:

- Use the Start button in the Toolbar.
  
- From the Terminal Emulator, type `RUN PROG0`.
- Set the program run request flag, bit 1032.

# Program Labels

Labels are program pointers which provide a method of branching to specific locations, including subroutines, within the same program. Labels can only be defined within a program and executed with a GOTO or GOSUB from within the same program.

Observe the following rules when creating and using labels:

- Precede the label with an underscore ( _ ) character.
- Use letters (case-sensitive) and numbers, but not spaces or symbols.
- Use the RETURN command to indicate the end of the subroutine.
- Do not put a REM command on the same line as a label.

Example:

```
PROGRAM
GOSUB STARTUP           : REM Go subroutine to STARTUP label.
ACC 10 DEC 10 STP 10 VEL 1
_MAIN
X10                     : REM Move to 10 on X axis.
X0                      : REM Return to start on X axis.
GOTO MAIN               : REM continue program at MAIN label.
_STARTUP
DRIVE ON X              : REM Enable the X axis motor.
RES X                   : REM Resets current X position to 0.
RETURN                  : REM Go back to GOSUB STARTUP and continue.
ENDP
```

This program adds two labels: STARTUP and MAIN. The program first does a GOSUB (subroutine) to _STARTUP, where it enables the X axis motor and resets the position. With the RETURN it goes back from where it was called and continues to line of ACC 10 DEC 10... It then enters a section of code with label of _MAIN and with the GOTO MAIN would repeatedly move the motor back and forth between position of 10 and 0. The Halt Program button (stop sign icon on the Toolbar) can be used to stop the program. Even though ENDP is never reached, it is still needed as part of the download to define the end of the program.

# Move—Default Motion

The ACR controllers are programmable motor controllers. The default move type is MOV, which is a coordinated move in a straight line for all axes involved.

An axis with a number would imply moving to that position. The ACR will interpret these as coordinated moves by default and hence the MOV command is optional. These lines are the same:

```
X10         : REM Move to 10 on X axis.
MOV X10     : REM Move to 10 on X axis.
```

This saves typing and thus the MOV doesn't need explicitly typed but is the parent to the child axis X.

```
X/10        : REM Increment X +10  /is an incremental move
```

```
X0          : REM Move 0 position.  This is an absolute move
X-5 Y5      : REM Linearly interpolated move to X-5 Y5
```

With a linearly interpolated move, all axes start and stop at the same time. The velocity, accel and decel are for the path of all commanded axes.

For `MOV` commands, the program starts the move and then continues the program. If there is a new `MOV` command, it will wait until the first `MOV` is done before starting the new `MOV`.

All `MOV` interpolated moves must be commanded from the program to which their master is attached. The master is the motion calculator and ACR controllers have eight masters available. The above commands need to be in Program 0 because, by default, X and Y are attached to Master 0, which is attached to Program 0.

More motion types are available and further explanation is provided in Making Motion.

# *Axis Names*

In ACR Series example code, Axis 0 is the X axis and Axis 1 is the Y axis, unless otherwise specified. Axis names (aliases) are only recognized within the program to which their master is attached!

PMM's Configuration Wizard allows users to assign axis names on Axes within the Alias column. This adds the alias to the axis in the System Code with the `ATTACH` command. Or, programmers could also assign an axis name to an axis through the `ATTACH SLAVE` command. The name can be 1-4 alpha character string (no numbers or special characters).

By default, ACR74 will have four axes, ACR78 will have eight axes and IPA will have one axis:

```
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH SLAVE2 AXIS2 "Z"
ATTACH SLAVE3 AXIS3 "A"
ATTACH SLAVE4 AXIS4 "B"
ATTACH SLAVE5 AXIS5 "C"
ATTACH SLAVE6 AXIS6 "U"
ATTACH SLAVE7 AXIS7 "V"
```

The axis name is valid at the program prompt in the Terminal Emulator:

```
P00>DRIVE ON X
P00>DRIVE ON X Y
```

It is also valid from within the program to which it is attached—Program 0 by default.

Outside of the program to which the axis is attached, users can set axis parameters using `AXIS` syntax:

```
AXIS0 DRIVE ON
```

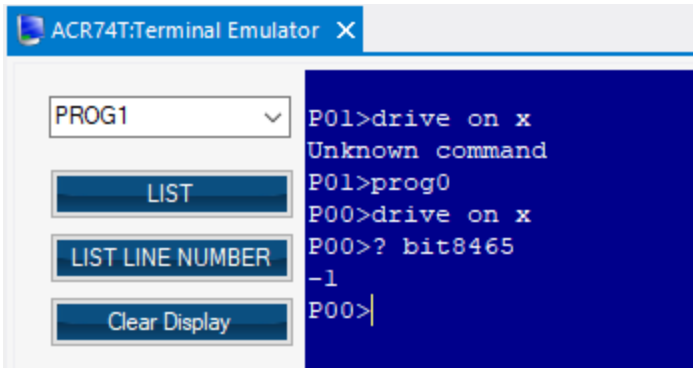
…Or…

```
DRIVE ON AXIS0
```

Or, for multiple axes:

```
DRIVE ON AXIS0 AXIS1
```

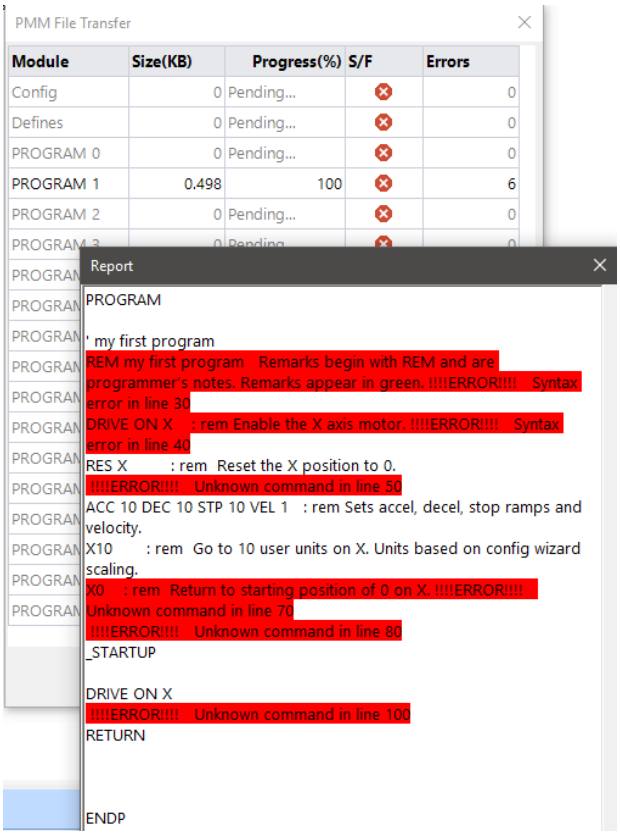
Programmers will get an *Unknown command* error message if trying to use the axis name outside the program to which it is attached:




To see a controller's attachments, type `ATTACH` in PMM's Terminal Emulator and press *Enter*. Here is an example for an ACR74T four-axis stepper:

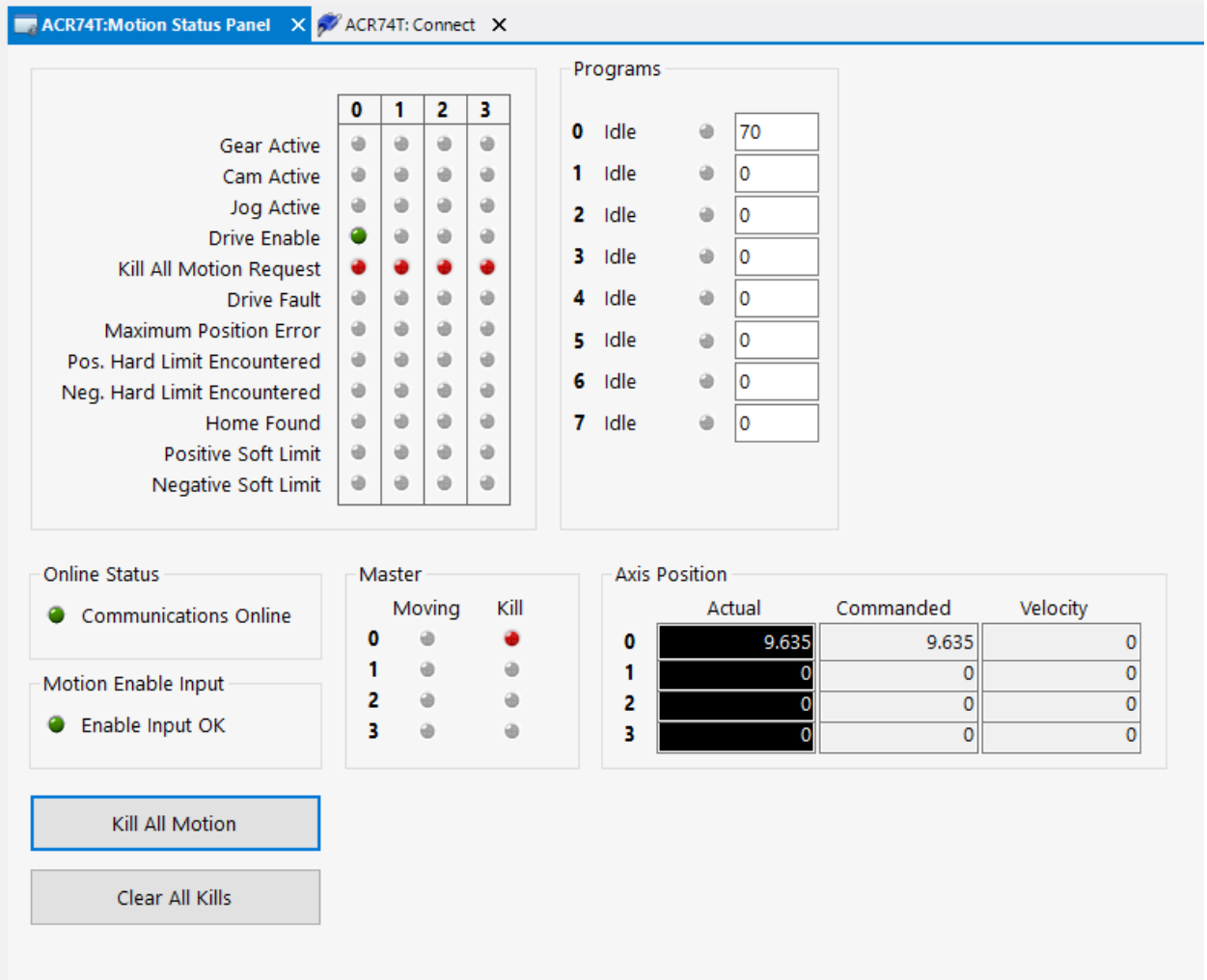
```
SYS>ATTACH
PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH SLAVE2 AXIS2 "Z"
ATTACH SLAVE3 AXIS3 "A"
```

These attachments are already set by PMM's Configuration Wizard after download. As the sample programs command motion with X, the sample programs must be used within Program 0. Syntax errors occur within other programs as X is not recognized outside of Program 0:




# *Stopping Motion*

When running and testing a program, users should be ready to kill motion—there is a button on the Motion Status Panel for this purpose.




With the *Kill All Motion* button, all programs are halted and the Master Kill is set as well as each axis' Kill All Motion Request (KAMR) bit. The axes will not be disabled. To run the program again, click *Clear All Kills*. This will clear the Kill for the Master(s) as well as the KAMR for the axes. Then the program can be run again.

There are several ways to kill motion from Parker Motion Manager:

- Ctrl+X in terminal window – Kill All Motion.
- Ctrl+Z in terminal window – Kill All Motion and Disable drives.
- *Kill All Motion* button in Motion Status panel (shown above), sends Ctrl+X.
- *Kill All Motion & Disable All Drives* button in Jog/Home/Limits, sends Ctrl+Z.
- *Kill All Motion* button in Servo Tuner, sends Ctrl+X.
- Disabling the drives is the same as sending the `DRIVE OFF` command.

Motion can be stopped programmatically from an AcroBASIC program, external HMI or PLC by setting the Kill All Motion Request (KAMR) flag.

Killing/stopping motion this way relies on being connected to the controller. If you lose the connection, users will need to stop motion on the machine either with a switch or by removing power. Use an E-stop switch into the IPA's Torque Enable inputs to remove power from the motor or for ACR7000 to cut the enable input. See the

controller's hardware installation manuals for details. Local safety standards may require removing power from the controllers. Note that when the motors are disabled, the drive's brake outputs (for servo drives) will turn off, engaging brakes on the motor. Parker servo motors have standard fail-safe brakes as an option, where the brake needs to be powered to disengage.

Killing or stopping motion is stopping commanded motion. If a servo system is unstable or position error is very high, a servo system can still be moving. If position maintenance is on it can also still be moving though position maintenance velocity is typically set low for slow end-of-move corrections.

Clicking a Kill button in PMM or doing a Ctrl+X or Ctrl+Z will set the Kill All Motion Request flag on all axes.

Any motion command issued while this flag is set will result in an error message *Associated Slave Kill Motion Request Active* in the terminal emulator. This is true if any axis assigned to the same master is commanded to move.

The Kill All Motion Request will also be set if an axis trips a limit sensor or if a servo or closed-loop stepper faults on a tracking error. To restart the controller, click the Clear All Kills before running the program or starting motion again.

**NOTE: Enabling drives using DRIVE ON command will clear the Kill All Motion Request (KAMR) and Kill All Moves bits if the drive is not currently enabled.**

We will discuss more about stopping in Stopping Moves and Motion.

PMM's Toolbar allows the user to start and halt programs.


This presumes the controller has been setup with the Configuration Wizard, with the motor selected, end of travel sensors connected and the program downloaded.

We recommend having the Motion Status Panel open to Kill All Motion when first running the program in case unexpected motion occurs.

# *Program Flow*

Code is executed sequentially, following the order in which it is written. But based on some input, you can stop and wait, or shift code execution elsewhere in a program using conditional statements. Using conditional statements, you can create code that tests for specific conditions and repeats code statements.

The conditional statement provides a logical test—a truth statement—allowing decisions based on whether the conditions are met. In the code, you create an expression and test whether the result is true.

The selection structure controls the direction of program flow. Think of it as a branch in your program. You can divide conditional statements into three sub-categories: wait, selection and repetition.

# *Wait for Bit or Parameter*

There are two commands that pause program execution waiting for a condition: INH and IHPOS. The INH command lets you inhibit (pause) program execution until the state of a selected bit occurs. Similarly, the IHPOS command lets you inhibit program execution until a parameter value is reached.

Either can pause the program execution forever until the bit/parameter condition is met and thus both have an optional timeout.

```
INH 0                   : REM Wait until bit 0 is on or true.
INH-0                   : REM Wait until bit 0 is off or false.
IHPOS P12290 (40000,0)  : REM Wait until Axis0 actual position > 40000 cts.
IHPOS -P12290 (40000,5) : REM Wait until Axis0 act. pos. < 40000 or 5 sec.
IHPOS P12290 (40000,5)  : REM Wait until Axis0 act. pos. > 40000 or 5 sec.
```

**NOTE: Position parameters in ACR are in counts. To convert from user units, multiply by axis PPU. This can be done inside parentheses.**

```
IHPOS P12290 ((5*P12375),0)    : REM Wait until X reaches 5 user units.
```

Modifying our previous example, we can use INH and IHPOS to wait until X is past 1 unit to turn on an output. After reaching a position of 10, it turns the output off and returns to 0.

```
PROGRAM
GOSUB STARTUP                  : REM Go subroutine to STARTUP label
ACC 10 DEC 10 STP 10 VEL 1     : REM Set move parameters
_MAIN
X10                            : REM Move to 10 on X axis
IHPOS P12295 ((1*P12375),0)    : REM Wait until X axis is past 1 unit
SET 32                         : REM Turn on output bit32
INH-516 : ? "AT X10"           : REM Wait until move is done, print
CLR 32                         : REM Turn off output bit32
X0                             : REM Return to start on X axis
INH-516 : ? "AT X0"            : REM Wait until move is done, print
GOTO MAIN                      : REM continue program at MAIN label
_STARTUP
DRIVE ON X                     : REM Enable the X axis motor
INH 8465 (3)                   : REM Wait until Axis0 is enabled or 3seconds
IF (NOT BIT8465) THEN ? "DRIVE DIDN'T ENABLE" : END
RES X                          : REM Reset current X position to 0
RETURN                         : REM Go back to GOSUB STARTUP and continue
ENDP
```

Because ACR controllers buffer MOV moves, AcroBASIC programs will continue to process command lines. If another MOV move is encountered, the program will wait there until the first move is done.

For interpolated MOV moves, an inhibit command using the In Motion flag can be used to make the program wait until the move is complete before proceeding.

The Jog Active bit can be used for Jog moves. Jog moves are not buffered and a second Jog move will interrupt the first one.

`INH -516` This will inhibit the program until Master 0's In Motion flag is off. This can be used for any `MOV`, such as a single-axis `X10` or a multiaxis `MOV` such as `X5 Y5`.

`INH -792` This will inhibit the program until Axis 0's Jog Active bit is off. This would be for a single-axis Jog move such as `JOG ABS X10`.

From the program above:

```
X10                             : REM Move to 10 on X axis
IHPOS P12295 ((1*P12375),0)     : REM Wait until X axis is past 1 unit
SET 32                          : REM Turn on output bit32
INH-516 : ? "AT X10"            : REM Wait until move is done, print
```

Without the `IHPOS` and `INH-516`, the move would start, then output 32 would immediately turn on and the message would print even though the axis is still moving.

**NOTE: Do not use INH or DWL in programs 8-15. If you have multiple non-motion programs, an inhibit or dwell in one non-motion program affects all non-motion programs.**

With a timeout, the condition would need to be checked to see if the program continued because the condition was true or because the timeout elapsed. From the program above:

```
DRIVE ON X                      : REM Enable the X axis motor
INH 8465 (3)                    : REM Wait til Axis0 is enabled or 3seconds
IF (NOT BIT8465) THEN ? "DRIVE DIDN'T ENABLE" : END
```

This second line will print a message and end (stop) program execution if the drive did not enable.

# *Selection*

The selection structure controls the direction of program flow. Think of it as a branch in your program. When the conditions are met, the program moves to a different block of code. AcroBASIC provides the following conditional statements:

- `IF/THEN`
- `IF/ELSE/ENDIF`
- `GOSUB/RETURN`
- `GOTO`

## IF/THEN

Programs need to run code based on specific conditions. The `IF/THEN` statement lets a program test for a specific condition and respond accordingly.

The IF portion is the condition to test; if the condition proves true, the THEN portion of the statement executes. If instead the condition proves false, the THEN statement is ignored and program execution moves on to the next statement.

**NOTE: Enclose the condition being tested in parentheses.**

Though the IF/THEN statement provides a single-line test, it can execute multiple statements when the condition proves true. All the statements must appear on a single line and be separated by a delimiter (space, colon, and another space).

When using an IF/THEN statement, users can nest GOTO and GOSUB statements.

```
IF (BIT 24) THEN P0 = P0+1
IF (P0 < 2) THEN GOSUB LoadParts : P0 = 100
```

Or, from the previous program sample, check if the drive did not enable. If it did not enable, print a message and end the program:

```
IF (NOT BIT8465) THEN ? "DRIVE DIDN'T ENABLE" : END
```

## IF/ELSE/ENDIF

The IF/ELSE statement provides a powerful tool for program branching and program flow control. The IF/ELSE statement allows you to run one set of code if the condition is true, and another set of code if the condition is false. The IF/ELSE statement must end with ENDIF.

When using an IF/ELSE statement, observe the following:

- You can nest GOSUB statements in an IF/ELSE statement. The GOSUB provides a return into the IF/ELSE statement.
- Do not nest GOTO statements in IF/ELSE statements. The GOTO statement exits the IF/ELSE statements and does not provide any link back inside.
- Do not nest IF/THEN statements in IF/ELSE statements—the logic may not provide the results you expect.

Tip: When troubleshooting programs, use the LIST command to view the program stored on the controller. In recognizing IF/ELSE statements, the controller indents the statements under the IF including the ENDIF. If any statements in the IF/ELSE are not indented but should be, check the code in the program editor and re-download.

The following demonstrates different actions based on conditions being true or false. If the input (bit 24) is true, the long array increments and axis X moves an incremental distance of 25 units. If false, the long array decrements and axis Y moves to an absolute position of 5.

```
IF (BIT 24)
    LA0(1) = LA0(1)+ 1
    X/25
ELSE
    LA0(1) = LA0(1)- 1
    Y5
```

```
ENDIF
```

## ELSE IF Condition

The IF/ELSE statement can include the ELSE IF condition. The ELSE IF condition lets you create a series of circumstances to test. There is no practical limit to the number of ELSE IF conditions you can include. However, they must come before the ELSE condition.

Here is how it works. When the IF condition is true, the subsequent statements are executed. When the IF condition is false, each ELSE IF statement is tested in order. When an ELSE IF condition tests true, the subsequent statements are executed. When the ELSE IF condition test false, the statements following the ELSE condition execute. After executing the statements following an IF, ELSE IF or ELSE, the program moves past the ENDIF to continue program execution.

When using the ELSE IF condition, you can omit the ELSE condition. When the IF and ELSE IF conditions test false, statement execution after the ENDIF continues. Think of it as creating a series of IF/THEN statements.

```
IF (P0>0)
    X/25
ELSE IF (P0=0)
    X0
ELSE
    X-10
ENDIF
```

## GOSUB/RETURN

The GOSUB branches to a subroutine and returns when complete. You can use GOSUB and RETURN anywhere in a program, but both must be in the same program. A procedure can contain multiple RETURN statements. However, on encountering the first RETURN statement, the program execution branches to the statement directly following the most recently executed GOSUB statement.

***Example***

The following example demonstrates a simple GOSUB routine.

```
GOSUB Label1
…
_Label1
PRINT "Inside Label1 subroutine"
RETURN
```

## GOTO

The GOTO statement provides an unconditional branch within a procedure. You can only use the GOTO in the procedure in which it appears.

You can nest GOTO statements in an IF/THEN statement.

**NOTE: The GOTO statement makes code difficult to read and maintain. Use it wisely.**

The following demonstrates a simple GOTO statement. The program sets output bit 32, then moves axis X one incremental unit in the positive direction. The program pauses until Axis 0's Not In Position bit (bit 768) turns off (meaning it is in position), then clears the output, waits 2 second, and goes to LOOP1.

```
ACC10 DEC10 STP10 VEL1
_LOOP1
    SET 32
    X/1
    INH -768
    CLR 32
    DWL 2
GOTO LOOP1
```

This would loop continuously until the program was halted by another program (HALT PROG0), by the user using PMM (Halt button on Toolbar or via the Terminal Emulator) or by setting bit 1033 (Program 0 Halt Request bit).

# *GOTO and GOSUB Sample Program*

Runs on Auto mode or Jog mode based on inputs.

```
PROGRAM
GOSUB STARTUP                  : REM Go subroutine to STARTUP label
ACC 10 DEC 10 STP 10 VEL 1     : REM Set move parameters
_MAIN
'AUTO MODE
IF (BIT 0)                     : REM  Input0 Auto Switch on
X10                            : REM Move to 10 on X axis
SET 32                         : REM Turn on output bit32
DWL 0.5                        : REM Dwell 0.5 sec
CLR 32                         : REM Turn off output bit32
X0                             : REM Return to start on X axis
P0=P0+1                        : REM increment a part counter
'JOG MODE
ELSE IF (BIT1)                 : REM If Jog+ switch
    JOG FWD X
ELSE IF (BIT2)                 : REM If Jog- switch
    JOG REV X
ELSE
    JOG OFF X                  : REM If Auto and Jog are off, Stop Jog
ENDIF
IF (P0>=3) THEN END            : REM end program after 3 cycles
GOTO MAIN                      : REM continue program at MAIN label
_STARTUP
DRIVE ON X                     : REM Enable the X axis motor
INH 8465 (3)                   : REM Wait until Axis0 is enabled or 3seconds
REM  If drive does not enable, end program
IF (NOT8465) THEN ? "DRIVE NOT ENABLED" : END
RES X                          : REM Reset current X position to 0
```

```
RETURN                          : REM Go back to GOSUB STARTUP and continue
ENDP
```

# *Repetition*

The repetition structure—known as a loop—controls the repeated execution of a statement or block of statements.

While the conditions remain true, the program loops (or iterates) through the specific code. Typically, the repetition structure includes a variable that changes with each iteration. And a test of the value determines when the conditions of the expression are satisfied. The program then moves to the next statement past the repetition structure.

If the condition is not met, the loop does not execute. In many cases that is acceptable behavior. Conversely, if the condition is always met, then the loop does not end. An endless loop is probably not a desired result, so be mindful when writing the loop conditions.

AcroBASIC provides the following conditional looping commands:

- `FOR/TO/STEP/NEXT`
- `WHILE/WEND`

## FOR/TO/STEP/NEXT

When you expect to loop through a block of code for a number of times, the `FOR/NEXT` loop is a good choice. It contains a counter, to which you assign starting and ending values. You also assign a `STEP` value (positive direction only), the value by which the counter increments.

When the `FOR/NEXT` loop executes the first time, the end value and the counter are compared. If the current value is past the end value, the `FOR/NEXT` loop ends and the statement immediately following executes. Otherwise, the statement block within the `FOR/NEXT` loop executes.

On each encounter of the `NEXT` statement, the counter increments and loops back to the `FOR` statement. The counter is compared to the end value with each loop. When the counter exceeds the end value, the loop skips the statement within, and proceeds to execute the statement immediately following the `FOR/NEXT` statement.

You can exit a `FOR/NEXT` loop before the counter is complete using a `BREAK` statement. When the condition is met, the statement immediately following the `FOR/NEXT` loop executes.

```
FOR LV0 = 0 TO 499 STEP 1
    PRINT LA0(LV0), SA0(LV0)
    DWL 0.01
    IF (BIT 24)
        BREAK
    ENDIF
NEXT
```

## WHILE/WEND

The `WHILE/WEND` loop executes as long as its condition remains true. You can use the `WHLE/WEND` anywhere in a program.

The WHILE sets the condition and is followed by statements you want executed when the condition is true. When the condition is false, the statement immediately following WEND executes. The condition is evaluated only at the beginning of the loop.

When using a WHILE/WEND statement, observe the following:

- Do not nest GOTO statements in a WHILE/WEND statement.
- At the start of each loop through the WHILE condition, the validity of the condition is tested.

The following demonstrates a WHILE/WEND loop. While the encoder position for Axis 2 is less than 1500 units, the WHILE statement evaluates as true. As the loop runs, the array acts as a counter, incrementing with each loop; axis X move an incremental 25 units; the program pauses for 1.5 seconds, then prints the current value of the array; if the input (bit 24) is set, the loop breaks. When the encoder count exceeds 1500, the condition is false and execution moves past the WEND statement.

```
WHILE (P6176 < 1500)
    LA0(1) = LA0(1) + 1
    X/25
    DWL 1.5
    PRINT LA0(1)
    IF (BIT 24)
        BREAK
    ENDIF
WEND
```

# *Bits, Parameters and Variables*

The ACR uses parameters (registers) and flags (bits) to store information and define the behavior of the controller. Users have almost unlimited access to the parameters and flags for use in programs or a user interface. Most applications only need to utilize a small set of the thousands of registers available. The more complex the application, the more parameters are likely to be used.

Parameters are registers of numeric data that are either 32-bit integers (LONGs) or 32-bit decimal values (FLOATs). Flags (bits) are binary and are either TRUE (high or -1) or FALSE (low or 0).

There are two types of bits: request and non-request.

Request Bits: The bit is self-clearing when processed by the main processor. All request bits include "request" in the name. In most cases, there are complimentary flags that perform the opposite action. For example, the Run Request bit and the Halt Request bit control the running and halting of programs.

Non-Request Bits: The bit requires clearing through a program or manually through a terminal.

There are separate parameter and bit tables within PMM's online help and also the separate ACR Parameter and Bit Reference. Following each is a table providing descriptions of the parameters or bits and the read/write attributes.

**NOTE: The values for some parameters and bits change automatically through operation of the ACR controller. Changing (writing) a value does not ensure the parameter or bit retains the value over the course of operations. Use caution—forcing a value to change can cause unpredictable results.**

Following is a list of the most commonly used parameter and bit tables:

- Master Parameters
- Master Flags
- Axis Parameters
- Axis Flags
- Object Parameters (includes analog inputs)
- Program Parameters
- Program Flags

In addition to the system bits and parameters, you have your own user bits and parameters that can be used within the controller and to interface to other devices like an HMI or PLC.

## User Bits and Parameters

Five groups of global parameters are available as user defined parameters.

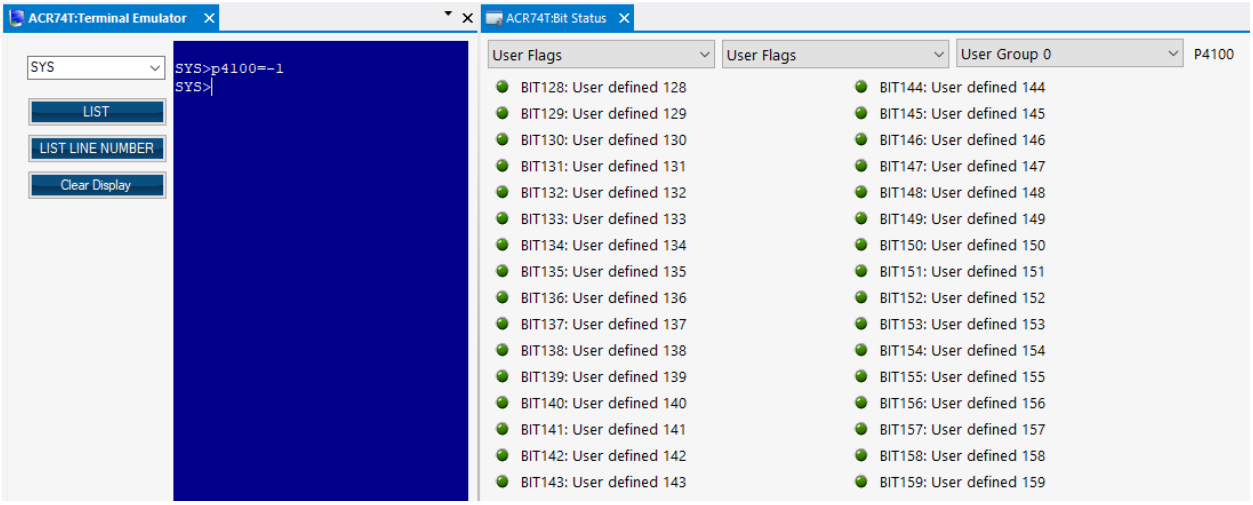
| Parameter Range | Data type | Retained | | Notes |
|---|---|---|---|---|
| | | ACR7xx0 | IPA | |
| P0-P4095 | 64-Bit Float | Flash | Flash | Must be dimensioned prior to use. Dimensioning included in PMM Configuration Wizard. FLASH IMAGE to save |
| P4100-P4103 | 32-Bit Long | No | No | Each parameter contains 32 user flags (BIT128-255) |
| P4156-P4159 | 32-Bit Long | No | No | Each parameter contains 32 user flags (BIT1920-2047) |
| P38912-P39167 | 32-Bit Long | Yes | Yes | Retained in Non-Volatile USER RAM |
| P39168-P39423 | 32-Bit Float | Yes | Yes | Retained in Non-Volatile USER RAM |
| BIT128-255 | Bit | No | No | Also accessible as P4100-P4103 |
| BIT256-511 | Bit | No | No | Also accessible as P4104-P4111 |
| BIT1920-2047 | Bit | No | No | Also accessible as P4156-P4159 |

Note that P4100 is 32 bits long, which are bits 128-159.

Each bit can be cleared individually, or more easily `P4100 = 0` would do the same thing and save a lot more programming space. `P4100 = -1` would set all the bits.

The range of a 32-bit Long is -2147483648 to 2147483647.

PMM's Bit Status now includes the User Flags. Note the *P4100* in the top-right indicating that parameter is made of bits 128-159.




# *Using Parameters and Bits*

You can specify parameters and bits in your programs or in the Terminal Emulator. Use the format `Px` or `BITx`, where x represents the parameter or bit number.

The following demonstrates how to format parameters and bits. Suppose your program refers to the current position for Axis 0 (see table P12288-P14199 Axis Parameters) and input 24 (see table Bit0-Bit31 Opto-Isolated Inputs).

```
P12288
BIT24
```

## Setting Binary Bits

You can use the `SET` command, or fix the bit value equal to 1. The following demonstrates how to set at bit. All methods are valid.

```
SET 32
Bit32=1
SET Bit32
SET BIT 32
```

`SET` is always used with a bit. Thus, the `BIT` in the line is redundant and optional. Note the space between `BIT` and the number is optional and both are valid syntax.

## Clearing Binary Bits

You can use the `CLR` command or fix the bit value equal to 0. The following demonstrates how to set at bit. All methods are valid.

```
CLR 32
Bit32=0
```

```
CLR Bit32
CLR BIT 32
```

CLR is also always used with a bit. Thus, the BIT in the line is redundant and optional. Note the space between BIT and the number is optional and both are valid syntax.

### Printing the Current Value

You can send the PRINT command followed by a parameter or bit whose value you want to see. Bits return the following values:

- -1 when set.
- 0 when clear.

You can use a question mark in place of the PRINT command. The question mark is a shortcut in the Terminal Emulator.

**NOTE: When printing a system parameter, the value returned is either an integer or a 32-bit floating point.**

When printing a user parameter (P0-P4095), the value returned is either an integer or a 64-bit floating point.

The following demonstrates how to view values stored in parameters and bits. Parameter 6144 provides the current position of Encoder 0; bit 24 provides the current state of input 24.

```
PRINT P6144
PRINT Bit24
? P6144
? Bit24
```

Note a PRINT statement can be used for parameters, bits or strings and thus querying a bit status will require "BIT" in the line.

### A Word on Aliases

Parameters and bits can use aliases. You only need to assign the alias once, and then can use it throughout user programs. The alias lets you provide a name that makes sense for programs and makes programs easier to read.

For more information, see Defines.

# *Programming Example*

The following program uses two orthogonal axes, X and Y, to draw a square. You can use PMM to set up the controller. Then, enter the program into Program 0 and download it to the controller.

```
RES X Y    : REM Reset encoder registers to 0 at startup.
_LOOP
ACC 50     : REM Set trajectory generator acceleration.
DEC 50     : REM Set trajectory generator deceleration.
STP 50     : REM Set trajectory generator stop ramp.
```

```
VEL 5      : REM Set target velocity.
X5         : REM Move axis to position.
Y5         : REM Move axis to position.
X0         : REM Move axis to position.
Y0         : REM Move axis to position.
GOTO LOOP
ENDP
```

Before running the program, make sure you are at the Program 0 prompt in the Terminal Emulator. The `LRUN` command lets you listen through a terminal to the `PRINT` statements and error messages. This is the only way to view program errors.

To run the program, type `LRUN`. When ready to exit the listening mode, press *Escape* (ASCII 27, top-left on most keyboards).

As the program runs, you can pause the program by setting the Feedhold Request bit or sending the `PAUSE` command. The Feedhold Request bit stops the axes using the deceleration value. To set the Feedhold Request bit, issue the command `SET 520`.

You can resume the program by setting Cycle Start Request bit or sending the `RESUME` command. The Cycle Start Request bit starts the axes using the acceleration value. To set the Cycle Start bit, issue the command `SET 521`.

While the program is in a feedhold, you can check the encoder position of each axis. To view the axis X encoder position, issue `PRINT P6144`. To view the axis Y encoder position, issue `?P6160`.

Note the space between "`?`" and "`P6160`" is optional.

# *Local Variables*

Users can also dimension and use different types of variables local in each program:

| Identifier | Type Description | Initialization |
|---|---|---|
| LV | Long (32 bit integer) | DIM LV(count) |
| SV | Single (32 bit floating point) | DIM SV(count) |
| DV | Double (64 bit floating point) | DIM DV(count) |
| $V | String (8 bit characters) | DIM $V(count) |
| LA | Long Array | DIM LAn(count) |
| SA | Single Array | DIM SAn(count) |
| DA | Double Array | DIM DAn(count) |
| $A | String Array | DIM $An(count) |

Local variables need to be dimensioned within program. Place a CLEAR command prior to a DIM statement to clear out any previously dimensioned variables.

Below is a sample program that extends and retracts and actuator to feed out material, printing the number of cycles, the length and the motor position.

```
PROGRAM
CLEAR                   : REM Clear any dimensioned variables.
DIM LV(1)               : REM Dimension 1 long variable.
DIM DV(2)               : REM Dimension 2 double variables.
LV0=0                   : REM Initialize cycle counter.
DV0=3.1412              : REM Set material feed distance.
_LABEL1
PRINT "Cycles=",LV0     : REM Comma inserts a tab character between.
PRINT "Length=";DV1     : REM Semicolon indicates there is no space.
PRINT "X",(P12290/P12375)
X(DV0)                  : REM Feed out material.
X0                      : REM Retract actuator.
LV0=LV0+1               : REM Increment the cycle count.
DV1=DV0*LV0             : REM Update the total length fed out.
GOTO LABEL1
ENDP
```

Use LRUN to execute the program in the Terminal Emulator and listen to PRINT commands. A PRINT statement that does not end with either a comma or a semicolon produces a carriage return and linefeed combination.

# *Defines*

From examples up to this point, AcroBASIC extensively uses bits, parameters and variables. These bits and parameters can be for controller status or the programmer's user data in programs. Alternative names, called defines, can be assigned to parameters, bits, constants and variables to make program code more readable. Defines are recognized globally (across user programs).

Observe the following rules when creating and using defines:

- Use a maximum of 23 letters.
- Defines are case sensitive.
- The first character must be a letter, but numbers can be used after that.
- Do not use spaces or special characters (such as _ and @).
- Use caution when using defines with local variables.

A define is recognized across programs, while local variables are limited to the program in which they are created. This can cause problems if you have created similar local variables in different programs. For example, if long variables are dimensioned in three programs, then the define "counter" is assigned to LV1 (long variable 1), the controller recognizes "counter" as a define in all three programs, though it represents a counter in only one program.

To assign defines, use the Defines editor in the Program Editor tree. This is a central location for defines that are global across all programs.

The `#DEFINE` command can be used within a program editor, at the top, before the `PROGRAM` line.

**NOTE: Aliases are reserved in memory. If changing an existing bit or parameter alias, redownload the configuration with the defines in PMM.**

Defines

| | Alias | Description | Type | Address/Value |
|---|---|---|---|---|
| 1 | bAutoRun | AutoRun pushbutton | Bit | 0 |
| 2 | bJogRight | Extend pushbutton | Bit | 1 |
| 3 | bJogLeft | Retract pushbutton | Bit | 2 |
| 4 | bXenabled | Motor energized status | Bit | 8465 |
| 5 | pCounter | Cycle counter | Parameter | 0 |
| 6 | bExtending | Output light/valve/relay | Bit | 32 |

Updated sample program to either run in Auto mode or Jog mode with defines:

```
PROGRAM
GOSUB STARTUP                  : REM Go subroutine to STARTUP label.
ACC 10 DEC 10 STP 10 VEL 1     : REM Set move parameters.
_MAIN
'AUTO MODE
IF (bAutoRun)                  : REM Input0 Auto Switch on.
X10                            : REM Move to 10 on X axis.
SET (bExtending)               : REM Turn on output bit32.
DWL 0.5                        : REM Dwell 0.5 sec.
CLR (bExtending)               : REM Turn off output bit 32.
X0                             : REM Return to start on X axis.
bCounter=bCounter+1            : REM Increment a part counter.
'JOG MODE
ELSE IF (bJogRight)            : REM If Jog+ switch.
    JOG FWD X
ELSE IF (bJogLeft)             : REM If Jog- switch.
    JOG REV X
ELSE
    JOG OFF X                  : REM If Auto and Jog are off, Stop Jog.
ENDIF
IF (bCounter>=3) THEN END      : REM end program after 3 cycles.
GOTO MAIN                      : REM continue program at MAIN label.
_STARTUP
DRIVE ON X                     : REM Enable the X axis motor.
INH bXenabled(3)               : REM Wait until Axis0 is enabled or 3seconds.
REM If drive does not enable, end program.
IF (NOT bXenabled) THEN ? "DRIVE NOT ENABLED" : END
RES X                          : REM Reset current X position to 0.
RETURN                         : REM Go back to GOSUB STARTUP and continue.
```

```
ENDP
```

Aliases thus make programs easier to read by allowing programmers to name bits and parameters.

# *Starting, Pausing, and Halting Programs*

PMM's Toolbar allows you to start and halt programs using buttons.


From the Terminal Emulator, you can also control programs from the `SYS` prompt, as well as any `PROG` prompt. You must include the program number when issuing the command from outside its program—for example, `RUN PROG0`. The commands described in this section provide immediate program control from PMM's Terminal Emulator.

## Running a Program

When the program starts, the controller returns to the `SYS` or `PROG` prompt. You can then enter immediate commands as the program runs.

To start a program, send the `RUN` command from the program prompt. Use PMM's buttons to change to the PROG0 prompt or type `PROG0` and press *Enter*. `P00>` is the PROG0 prompt. If the program does not run, issue the `LIST` command to see the program.

After a download, the terminal will be at the `P15` prompt. You could start Program 0 from here with `RUN PROG0`. This could also be done from the `SYS` prompt or any other program prompt such as PROG1 (`P01>` in the terminal).

## Running a Program at Power Up

You can set a specific program to automatically start after powering up or rebooting the controller. In the program editor, enter the `PBOOT` command as the first line in a program.

**NOTE: PBOOT must be the first line of the program. Any or all programs can be PBOOT.**

## Listening to a Program

While a program is running, you can "listen" to it. The listen mode displays data from the program `PRINT` statements and error messages.

To enable the listening mode on a running program, issue the `LISTEN` command. To exit the listening mode, press the *Escape* key (ASCII 27).

## Viewing a Running Program

You can also start and listen to a program using a single command. This is best used for development troubleshooting purposes. It is the only time you can view program syntax errors.

To start a program with the listening mode enabled, issue the LRUN command. To exit the listening mode, press the *Escape* key (ASCII 27).

### Halting a Program

You can stop program execution from the SYS or PROG prompts using the HALT command. This will not interrupt a move in progress.

**NOTE: HALT cannot be used inside a program. To terminate a program in the middle of execution inside a program, use the END command.**

### Pausing a Program

Pausing a program places a feed hold on the current move and suspends the program at the current command line.

To suspend a currently running program, send the PAUSE command.

### Resuming a Paused Program

Once paused, you can resume the program—motion and code execution continue from the places at which they paused.

To continue program operation, issue the RESUME command.

### Affecting Multiple Programs

You can control all programs simultaneously using the ALL argument. For example: RUN ALL, HALT ALL, PAUSE ALL, or RESUME ALL.

To control all programs, use the ALL argument in a command.

### Restart Controller

To test starting the controller, users could cycle power on the controller. Note this will cause the computer to lose connection to the controller temporarily. The REBOOT command is the same as cycling power.

To restart a controller, issue the REBOOT command.

### Running Startup Programs

You can run all startup programs without having to send individual RUN PROG commands.

To start all PBOOT programs, send the PBOOT command.

# *Parametric Evaluation*

Most commands take arguments. Often, those command-line arguments are literals—values that are interpreted as they are written. For example, axis numbers, bit index numbers, acceleration/deceleration speeds or positional values.

In addition to literals, you can use expressions (also called formulas). The ACR controller can solve complex integer or floating-point math. To use expressions, you must enclose them in parentheses. Expressions can use the following data sources:

- Constants
- Literals

- Variables
- Parameters
- Bits
- Aliases

An expression is comprised of at least one operand and one or more operators. Operands are values, whether literals or variables. Operators are symbols that represent specific actions. For example, the plus sign (+) represents addition, and the forward slash (/) represents division. In the expression:

```
A + 7
```

`A` and `7` are operands, and `+` is an operator.

**NOTE: For a complete list of operators available, see the Expression Reference section of the ACR Command Language Reference.**

Operations are performed in the following order:

- Powers
- Multiplication and division
- Addition and subtraction
- Relational operations (such as greater than, less than, not equal to)

The trigonometric (sine, cosine, tangent, etc.) and miscellaneous operators (absolute value, natural log, square root, etc.) require parentheses around their own expressions. The order of operations with such operators begins with the deepest nested parentheses.

## Parentheses and Operational Order

Using parentheses, you can group operations in an expression to change the order in which they are performed. For example, the expression:

```
4 + 6 / 2
```

Provides the answer 7, and not 5, because division performs before addition. When a mathematical expression contains operators that have the same rank, operations are performed left to right. For another:

```
2 + 6 / 3 * 5 - 9
```

Division and multiplication perform before addition and subtraction. The first operation is `6 / 3`; the second operation multiplies the result 2 by 5, which results as 10. In the third operation, add 2 to 10, which results as 12. In the fourth operation, subtract 9 from 12 to produce the final answer of 3.

By using parentheses, you can change the order of operations in an expression. That is, operations in parentheses are performed first, then operations outside the parentheses. For example, the expression:

```
(2 + 6 / 3) * 5 - 9
```

Results in an answer of 11, while the expression:

```
(2 + 6 / 3) * (5 - 9)
```

Results in -16 as the answer.

## Nested Parentheses

You can also embed parentheses, where operations in the deepest parentheses are performed first. For example, the expression:

```
((7 + 3) / 2) * 3
```

Contains embedded parentheses. From the example, the first operation is `7 + 3`, the second operation is `10 / 2`, and the third operation is `5 * 3`, which results in 15 as the answer.

## Examples

The following demonstrate some simple uses of expressions. The examples assume memory space is allocated for the variables.

### *Example 1*

The following causes axis X to move position to the resulting value of the expression.

```
X(P0 + P2 * P30)
```

### *Example 2*

When the following IF statement proves true, the message "OK" prints.

```
IF(P0=1234) THEN PRINT "OK"
```

### *Example 3*

The following concatenates strings $V1 and $V2 and sets string $V0 equal to the result.

```
$V0 = $V1 + $V2
```

### *Example 4*

The following program generates a random number from 0 to 999. As the program loops, it counts each loop. When the number equals 123, the program exits the loop and prints the count.

```
PROGRAM
DIM LV(2)          : REM dimension 2 long variables
LV0=0              : REM set LV0 equal to 0
_LOOP1
LV1=RND(1000)      : REM set LV1 equal to random number
LV0=LV0+1          : REM increment LV0 with each loop
IF (LV1<>123) THEN GOTO LOOP1
PRINT "Done in";lv0;"tries"
ENDP
```

To view the print statements and run this program, do `LRUN` from program prompt in the Terminal Emulator.

### *Example 5*

The following flashes the first 30 outputs in a random sequence.

```
PROGRAM
DIM DV(1)                : REM dimension 1 floating point variable
_LOOP2
```

```
DV0=RND(4294967295)      : REM set DV0 equal to random number
P4097= DV0               : REM set onboard outputs equal to DV0
GOTO LOOP2
ENDP
```

# *Example Code Conventions*

Examples that include code are provided throughout most of the ACR Series documentation to illustrate a concept, supply model code samples or to show multiple ways to employ the commands.

The example code may include the terminal prompt or configuration code if it is necessary for clarity. Example code is complete only as far as conveying information about the discussion, and configuration and other information may need to be added in order for the code to be of use in an actual application.

**NOTE: In ACR Series example code, Axis 0 is the X axis and Axis 1 is the Y axis unless otherwise specified.**

# ACR System

This section details the architectural layout of the ACR. Knowing the system architecture can help a developer better understand the product's strengths and limitations, allowing them to take full advantage of it.

## *ACR Architecture*

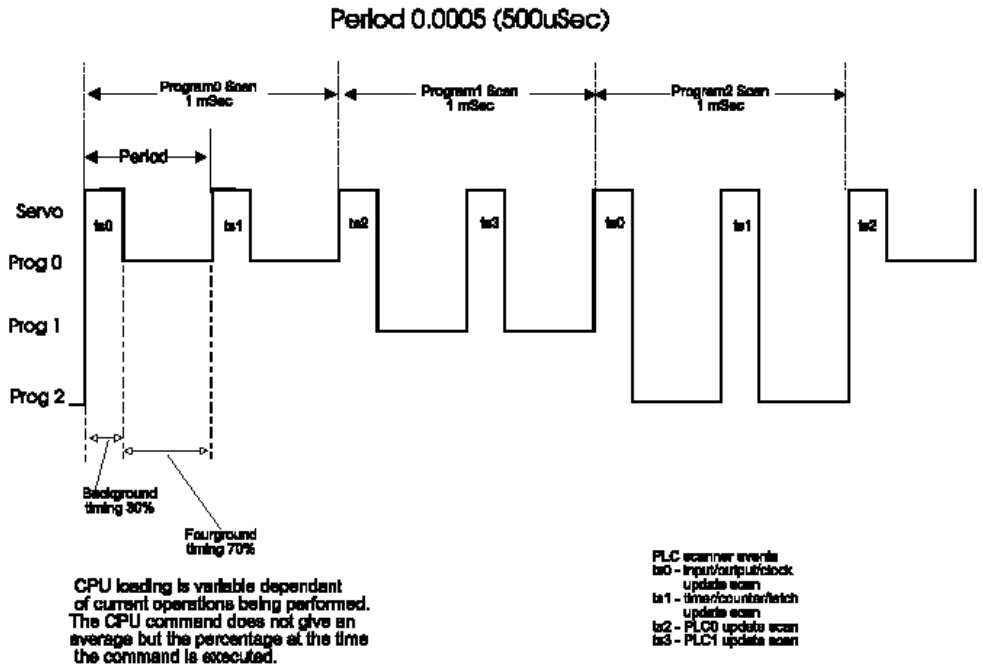
The ACR7000 uses a high-speed multitasking System on Module (SOM) processor for program and motion control. FPGAs are sent the position updates and output analog command signals or step and direction signals. The ACR7xV multiaxis servo platform uses one dedicated processor per axis to control current and close the position loop. The ACR7xT multiaxis stepper platform uses stepper ASICs to control current. The ACR7xV and ACR7xT use cutting edge high-power MOSFETs to drive current to the motors. For the ACR7xC controller, the FPGAs output the drive control signals for external drives.

The SOM processor is a pre-emptive multi-tasker. Users can use up to 15 programs, 8 high priority tasks (Motion programs Prog 0-Prog 7) and 7 low priority tasks (Non-Motion programs Prog 8-Prog 14). On the ACR7xV, the individual axis processors run a 500 µs (default) position loop and 31.25 µs current loop. On the ACR7xC, the position loop for all axes is run on the main SOM processor on a 500 µs (default) interval. Inputs and outputs are updated every 0.5 ms.

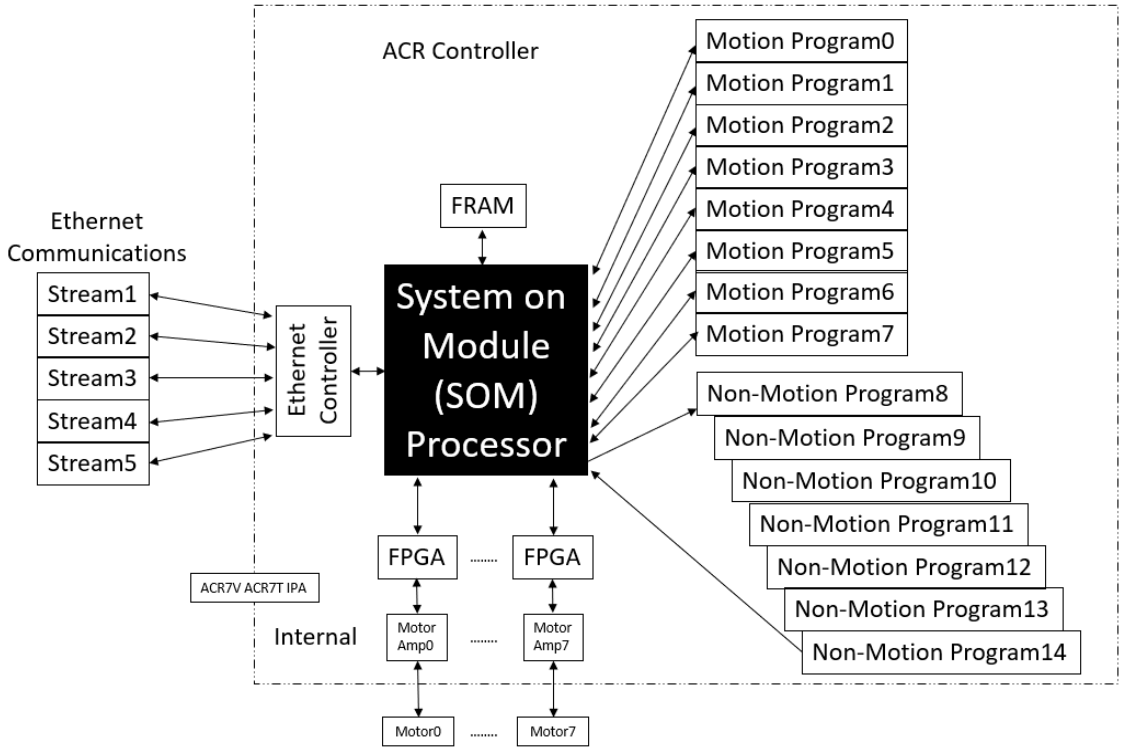
Prog 0-Prog 7 each have a 1 ms time slice and are used for Motion Programs. Each program has its own coordinated group of axes for motion and can run independently of the other programs. Each program has its own set of local variables (longs, floats, strings, arrays).

Prog 8-Prog 15 all share a 1 ms time slice and are used for non-motion programs. These programs are to be used for monitoring conditions, error recovery and handling communications to external devices.

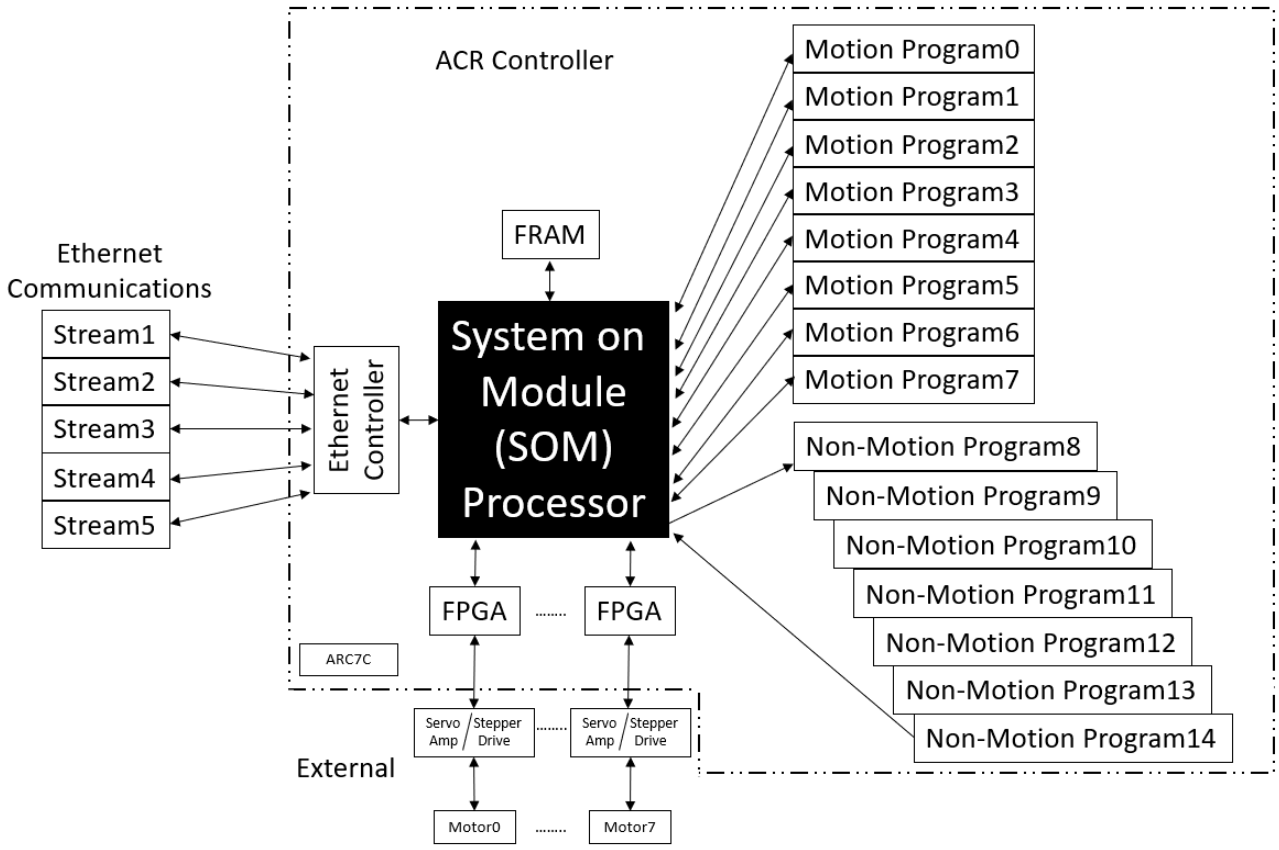
### *Program Execution Timing*



### *ACR7xV/ACR7xT/IPA Hardware Architecture*




### *ACR7xC Hardware Architecture*



# Ethernet

The ACR7000 and IPA's Ethernet implementation has 5 communication streams available, allowing 5 different connections to the controller. In addition to TCP/IP communications, both the ACR7000 and IPA controllers can be used in EtherNet/IP systems.

## Ethernet TCP/IP

You would first use the Ethernet port to configure and program the controller using Parker Motion Manager (PMM). PCs, HMIs and other machine components can also connect using Ethernet TCP/IP. The ACR7000 and IPA allow both ASCII communications through port 5002 and binary communication through port 5006.

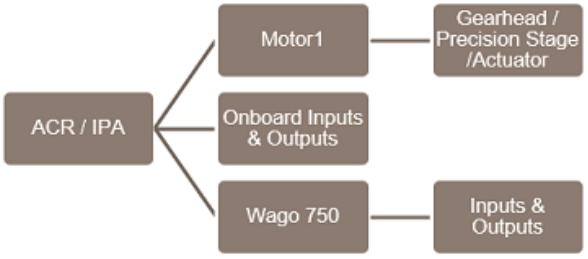
For Windows PC communications, ComACRServer6 is a 32 bit OLE automation server providing communications between ACR controllers and compatible PC software applications such as LabVIEW, Visual Basic.NET, Visual C++, C#, etc. Samples are available here.

ComACRserver6 is installed and used with Parker Motion Manager (PMM), running in the background. It is based on the Microsoft Component Object Model (COM) and allows for reading/writing bits and parameters, initiating motion and uploading/downloading programs. It is a wrapper for the Binary Host Interface. For further details see ComACRServer6 User's Guide.


For PC-based applications for OEM machinery using a non-Windows PC for control, or for users wanting faster communications, the Binary Host Interface can be used to connect to and control ACR or IPA controllers, explained here. This option is not recommended for most users.

## EtherNet/IP Scanner

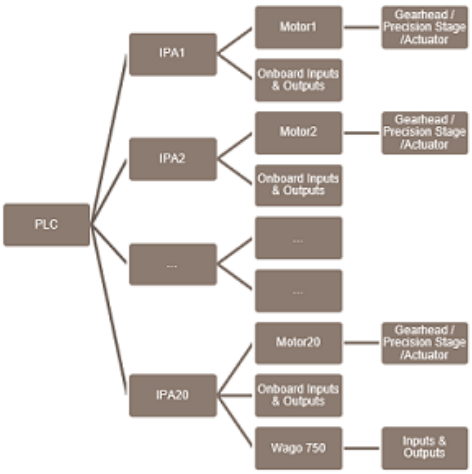
For applications needing more I/O than the controller's onboard inputs and outputs, the ACR7000 and IPA can be configured as a scanner for up to 16 Wago 750 series EtherNet/IP bus couplers. This allows for larger numbers of inputs and outputs, analog I/O and other types of I/O (higher current relay modules, temperature/thermistor modules, etc.) and distributed I/O on the machine. Each node can have max of 512 bits of inputs and 512 bits of outputs, 32 analog inputs and 32 analog outputs. For further information, see IPA Ethernet I/O User Guide (same applies to ACR7000). A sample ENIP Scanner program is available in Application Examples.



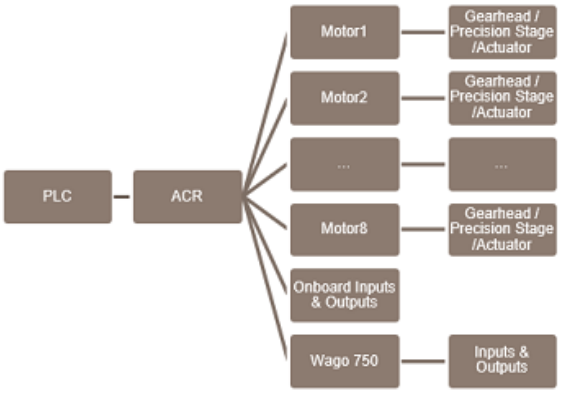
## EtherNet/IP Node

The ACR7000/IPA can also be a node on an EtherNet/IP network for use with Allen Bradley and Omron PLCs (others too). Both support class 1 and class 3 messaging. Further information can be found in IPA Ethernet/IP Programmer's Guide.

Add-On Instructions (AOIs) are available for the IPA for control from Allen-Bradley ControlLogix and CompactLogix PLCs. Further details can be found here.




PLC with multiple IPAs. IPA is a node on EtherNet/IP and can simultaneously be a scanner.



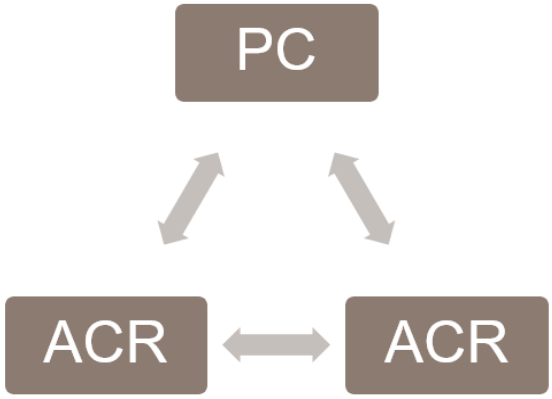

PLC with a multiaxis ACR. ACR is a node on EtherNet/IP and can simultaneously be a scanner.

## Ethernet/IP Peer-to-Peer

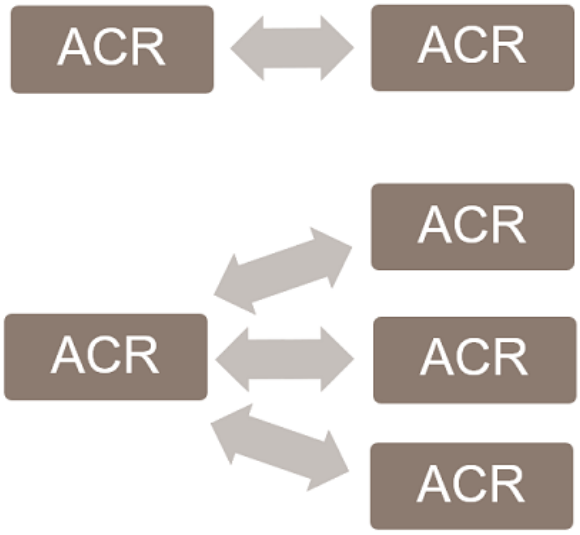
The ACR7000 and IPA's EtherNet/IP implementation can be a scanner for Peer-to-Peer communications with up to four other ACR7000 or IPA controllers. Though not for expansion of interpolated motion, this can be used to coordinate between controllers using only an Ethernet cable. Users can have systematic exchange up to 128 longs or floats in both directions. For further information, see IPA Ethernet I/O User Guide (ACR7000 is the same). A sample Peer-to-Peer program is available in Application Examples.

An ACR7000 or IPA can only be a scanner (client) for one type of device; it can be a scanner for a Wago 750 or another ACR7000/IPA, but not both. The peer ACR7000/IPA however can be a server to the first ACR7000/IPA and a scanner for a Wago 750 series.

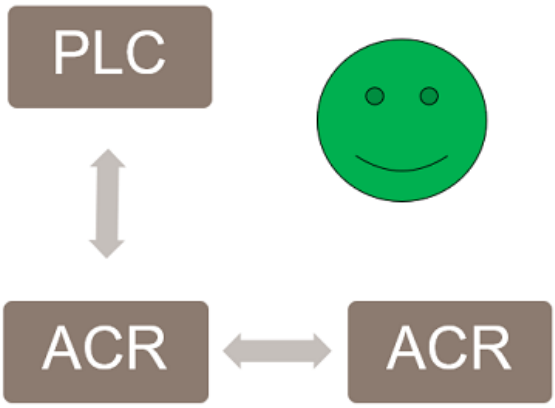
### *ACR EtherNet/IP Architecture Examples*



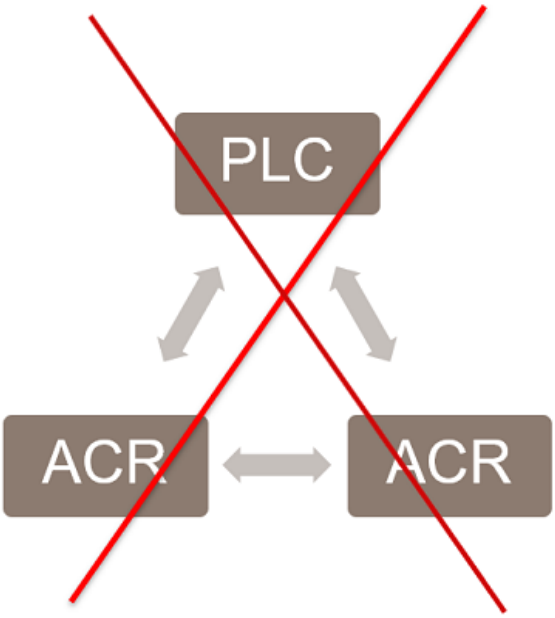

PC control via TCP/IP and two ACRs with Peer-to-Peer.




ACR standalone Peer-to-Peer (up to 4).




PLC control via EtherNet/IP with two ACRs Peer-to-Peer.




ACR cannot be an EtherNet/IP slave to both a PLC and another ACR.

# Command Syntax

The AcroBASIC programming language accommodates a wide range of needs by providing basic motion control building blocks, as well as sophisticated motion and program flow constructs.

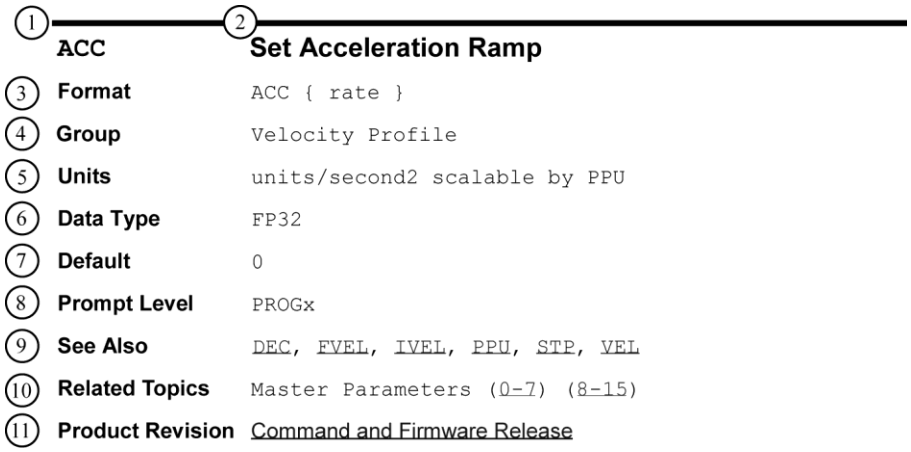
The language comprises simple ASCII mnemonic commands, with each command separated by a command delimiter (carriage return, colon, or line feed). The command delimiter indicates that a command is ready for processing.

The AcroBASIC programming language uses a parent-daughter approach. A parent can have daughter statements; a daughter statement is considered a sub-statement of the parent.

You can issue many parent statements alone—some provide the current status related to that particular command, others perform an action. For example, issuing the `ATTACH` command at the program level provides you with a report of the axis attachments to the master. Conversely, issuing the `CLEAR` command at the system level frees the memory allocated to all programs.

You can only issue some parent commands in conjunction with a daughter statement. For example, the `DRIVE` command has the `ON`, `OFF`, and `RES` daughter statements. Therefore, you can issue the `DRIVE ON (axis)`, `DRIVE OFF (axis)` and `DRIVE RES (axis)` commands, but not `DRIVE` by itself.

## *Description of Format*




1. **Mnemonic Code:** The ASCII command.
2. **Name:** A short description of the command.
3. **Format:** Indicates the proper syntax and arguments for the command.
4. **Group:** The functional group to which the command belongs.
5. **Units:** Indicates the units of measurement required by the argument(s) in the command syntax.
6. **Data Type:** Indicates the class of data required by the argument(s).
7. **Default:** Indicates the setting or value automatically selected unless you specify a substitute.
8. **Prompt Level:** Indicates the communication level at which you can use the command. For more information, see Communication Levels.
9. **See Also:** Indicates commands related or similar to the command you are reviewing.
10. **Related Topics:** Indicates parameter and bit tables related to the command you are reviewing.

11. **Product Revision:** To determine whether the command applies to your specific ACR series controller and firmware revision, see the Command and Firmware Release table.

# *Arguments and Syntax*

The syntax of an AcroBASIC command shows you all the components necessary to use it. Commands can contain required and optional arguments. They also contain a number of symbols:

- Braces { }—arguments that are optional. Do not type the braces in your code.
- Parentheses ( )—arguments that are optional, and must appear within the parentheses in your code. Also used to indicate variables and expressions. If replacing a constant with a variable or parametric equation, use parentheses to "contain" the variable/equation. Signed (-) or (+) constants must be in parentheses.
- Commas (,)—delimiters between arguments in specific commands. In addition, select commands use commas to control spacing and line feeds. To understand the separator's specific use in a command, refer to the command's format and description.
- Semicolons (;)—delimiters between arguments in specific commands. In addition, select commands use semicolons to control spacing and line feeds. To understand the separator's specific use in a command, refer to the command's format and description.
- Slash mark (/)—signifies an incremental move in select commands.
- Quotes (" ")—arguments within the quotes must appear within quotes in your code.
- Number sign (#)—device arguments following number signs must include the number sign in your code.
- Ellipsis (…)—arguments can be given for multiple axes.

The following examples illustrate how to interpret common syntax:

***Example 1***

```
ACC {rate}
```

In the `ACC` command, the lower-case word rate is an argument. Arguments act as placeholders for data you provide. If an argument appears in braces or parentheses, the argument is optional.

For example, the following sets the acceleration ramp to 10,000 units per second$^2$.

```
ACC 10000
```

When you issue a command without an optional argument, the controller reports back the current setting. Not all commands report back, and some require you to specify an axis. For example, the following reports the current acceleration rate in Program 0.

```
P00>ACC
10000
```

***Example 2***

```
PGAIN {AXIS {value}} {AXIS {value}} ...
```

Optional arguments can nest. This provides the flexibility to set data for or receive reports on multiple axes. For example, the following sets the proportional gain for axes X and Y to 0.0001 and 0.0002 respectively.

```
PGAIN X 0.0001 Y 0.0002
```

Because the PGAIN command can report on multiple axes, you specify at least one axis on which the controller is to report back.

```
P00>PGAIN X
0.0001
P00>PGAIN X Y
0.0001
0.0002
```

***Example 3***

```
IPB {AXIS {value}} {AXIS {(value1, value2)}} …
```

The AcroBASIC language provides programming shortcuts. You can set positive and negative values for commands using one argument. If the values differ, you can use two arguments. The command format illustrates when this is possible. For example, the following sets the in-position band for axis X to ±0.05 and for axis y to 3 and –1.

```
IPB X 0.05 Y(3, -1)
```

Or:

```
IPB AXIS0 0.05 AXIS1(3,-1)
```

Notice that the two values for axis Y are given inside parentheses and separated by a comma, as shown in the format of the command.

Note the X and Y aliases are only valid within the program to which the axes are attached (see ATTACH). Using AXIS0 or AXIS1 is valid in any program.

***Example 4***

```
HALT {PROGx | ALL}
```

The vertical bar indicates a choice between arguments. For example, the HALT command lets you stop a user program or all programs.

```
HALT PROG0
HALT ALL
```

# *Variable Substitution Syntax*

AcroBASIC commands can be used with parameters instead of numeric values. However, variable substitution requires use of parentheses.

***Example 5***

Modifying Example 3 to do the same thing but with variable substitution:

```
P0 = 0.05
P1 = 3
P2 = -1
IPB X (P0) Y((P1),(P2))
```

Or:

```
IPB AXIS0 (P0) AXIS1((P1),(P2))
```

Same if using Aliases with variable substitution:

```
#DEFINE ipbx P0
#DEFINE ipbypos P1
#DEFINE ipbyneg P2
IPB X (ipbx) Y((ipbypos),(ipbyneg))
```

Or:

```
IPB AXIS0 (ipbx) AXIS1((ipbypos, ipbyneg))
```

Note that `NOT` is not an operation and does not need to be in parentheses with a bit. This is valid syntax:

```
IF (BIT 0 AND NOT BIT 1) THEN P100=6
```

***Example 6***

Program sample using variables for move parameters:

```
PROGRAM
DIM SV10
SV2=100 : SV3=6
ACC (SV2) VEL (SV3) DEC (SV2) STP (SV2)
X/100
? SV2 : ? SV3 : INH 516
ENDP
```

# *Nested Commands Syntax*

Parametric evaluation can be used within commands, but as it is a command within a command, it needs to be within parentheses. This is a great way to condense program code.

```
JOG HOME X1         : REM Start homing X positive
REM Infinite WHILE statement while X is trying to HOME
WHILE ((NOT BIT 16134) AND (NOT BIT 16135))
WEND
REM Prints Information regarding "X" Axis homing
IF (BIT 16134) THEN PRINT "X HOMING SUCCESSFUL"
IF (BIT 16135) THEN PRINT "X HOMING UNSUCCESSFUL"
```

Axis X starts homing. The program continues into a `WHILE` loop. This is looping while axis X is homing, that is, while it has not found home (bit 16134, Axis 0 Found Home) and not failed (bit 16135 Axis 0 Failed to Find Home). We know either one or the other will happen. After homing, the `WHILE` loop is exited. This `WHILE` loop shows how to evaluate multiple bits within a command.

# Commands Lists

The tables in this section list commands according to the following command groups:



The ACR Command Reference and PMM online help gives full syntax and explanation of these commands with example code. The more common commands are covered within this Programmer's Guide. Many of the commands are settings and are part of the System Code generated by PMM's Configuration Wizard. See Configuration.

## Axis Limits

| Command | Description |
|---|---|
| ALM | Set stroke limit 'A' |
| BLM | Set stroke limit 'B' |
| EXC | Set excess error band |
| HLBIT | Set hardware limit/homing input |
| HLDEC | Hardware limit deceleration |
| HLIM | Hardware limit enable |
| IPB | Set in-position band |
| ITB | Set in-torque band |
| JLM | Set jog limits |
| MAXVEL | Set velocity limits |
| PM | Position maintenance |

| | |
|---|---|
| SLDEC | Software limit deceleration |
| SLIM | Software limit enable |
| SLM | Software positive/negative travel range |
| TLM | Set torque limits |

## Character I/O

| Command | Description |
|---|---|
| CLOSE | Close a device |
| INPUT | Receive data from a device |
| OPEN | Open a device |
| PRINT | Send data to a device |
| TALK TO | Talk to device |

## Drive Control

| Command | Description |
|---|---|
| DRIVE | Drive report-back |

## Feedback Control

| Command | Description |
|---|---|
| HSINT | High speed interrupt |
| INTCAP | Encoder capture |
| MSEEK | Marker seek operation |
| MULT | Set encoder multipliers |
| NORM | Normalize current position |
| OOP | High speed output |
| PPU | Set axis pulse/unit ratio |
| REN | Match position with encoder |
| RES | Reset or preload encoder |

| | |
|---|---|
| ROTARY | Set rotary axis length |

## Global Objects

| Command | Description |
|---|---|
| ADC | Analog input control |
| AXIS | Direct axis access |
| CIP | Ethernet/IP status |
| DAC | Analog output control |
| ENC | Quadrature input control |
| FSTAT | Fast status setup |
| LIMIT | Frequency limiter |
| MASTER | Direct master access |
| PLS | Programmable limit switch |
| RATCH | Software ratchet |
| SAMP | Data sampling control |

## Interpolation

| Command | Description |
|---|---|
| CIRCCW | Counter clockwise circular move |
| CIRCW | Clockwise circular move |
| INT | Interruptible move |
| INVK | Inverse kinematics |
| MOV | Define a linear move |
| NURB | NURBs interpolation mode |
| SINE | Sinusoidal move |
| SPLINE | Spline interpolation mode |
| TANG | Tangential move mode |

| | |
|---|---|
| TARC | 3-D circular interpolation |
| TRJ | Start new trajectory |

## Logic Function

| Command | Description |
|---|---|
| CLR | Clear a bit flag |
| DWL | Delay for a given period |
| IHPOS | Inhibit on position |
| INH | Inhibit on bit high or low |
| MASK | Safe bit masking |
| SET | Set a bit flag |
| TRG | Start move on trigger |

## Memory Control

| Command | Description |
|---|---|
| CLEAR | Clear memory allocation |
| DIM | Allocate memory |
| MEM | Display memory allocation |

## Non-Volatile

| Command | Description |
|---|---|
| ELOAD | Load system parameters |
| ERASE | Clear the EEPROM |
| ESAVE | Save system parameters |
| FIRMWARE | Firmware upgrade/backup |
| FLASH | Create user image in flash |
| PARTNUMBER | Displays controller part number |

| | |
|---|---|
| PBOOT | Auto-run program |

## Operating System

| Command | Description |
|---|---|
| ATTACH | Define attachments |
| BOOTREV | Displays boot revision |
| CONFIG | Hardware configuration |
| CPU | Display processor loading |
| DEF | Display the defined variable |
| #DEFINE | Define variable |
| DETACH | Clear attachments |
| DIAG | Display system diagnostics |
| ECHO | Character echo control |
| HELP | Display command list |
| IP | IP address |
| MODE | Binary data formatting |
| PASSWORD | Block uploading programs |
| PERIOD | Set base system timer period |
| PROG | Switch to a program prompt |
| REBOOT | Reboot controller |
| STREAM | Display stream name |
| SYS | Return to system prompt |
| VER | Display firmware version |

## Program Control

| Command | Description |
|---|---|
| AUT | Turn off block mode |

| | |
|---|---|
| BLK | Turn on block mode |
| HALT | Halt an executing program |
| LIST | List a stored program |
| LISTEN | Listen to program output |
| LRUN | Run and listen to a program |
| NEW | Clear out a stored program |
| PAUSE | Activate pause mode |
| REM | Program comment |
| RESUME | Release pause mode |
| RUN | Run a stored program |
| STEP | Step in block mode |
| TROFF | Turn off trace mode |
| TRON | Turn on trace mode |

## Program Flow

| Command | Description |
|---|---|
| BREAK | Exit a program loop |
| END | End of program execution |
| ENDP | End program without line numbers |
| FOR / TO / STEP / NEXT | Relative program path shift |
| GOSUB | Branch to a subroutine |
| GOTO | Branch to a new line number |
| IF/ELSE IF/ELSE/ENDIF | Conditional execution |
| IF / THEN | Conditional execution |
| PROGRAM | Beginning of program definition |
| RETURN | Return from a subroutine |

| | |
|---|---|
| WHILE/WEND | Loop execution conditional |

## Servo Control

| Command | Description |
|---|---|
| DGAIN | Set derivative gain |
| DIN | Dead zone integrator negative value |
| DIP | Dead zone integrator positive value |
| DWIDTH | Set derivative sample period |
| DZL | Dead zone inner band |
| DZU | Dead zone outer band |
| FBVEL | Set feedback velocity |
| FFACC | Set feedforward acceleration |
| FFVC | Feedforward velocity cutoff region |
| FFVEL | Set feedforward velocity |
| FLT | Digital filter move |
| IDELAY | Set integral time-out delay |
| IGAIN | Set integral gain |
| ILIMIT | Set integral anti-windup limit |
| KVF | PV loop feedforward gain |
| KVI | PV loop integral gain |
| KVP | PV loop proportional gain |
| LOPASS | Setup lopass filter |
| NOTCH | Setup notch filter |
| PGAIN | Set proportional gain |

## Setpoint Control

| Command | Description |
|---|---|

| | |
|---|---|
| BKL | Set backlash compensation |
| BSC | Ballscrew compensation |
| CAM | Electronic cam |
| GEAR | Electronic gearing |
| HDW | Hand wheel |
| JOG | Single axis velocity profile |
| LOCK | Lock gantry axis |
| UNLOCK | Unlock gantry axis |

## Transformation

| Command | Description |
|---|---|
| FLZ | Relative program path shift |
| OFFSET | Absolute program path shift |
| ROTATE | Rotate a programmed path |
| SCALE | Scale a programmed path |

## Velocity Profile

| Command | Description |
|---|---|
| ACC | Set acceleration ramp |
| DEC | Set deceleration ramp |
| F | Set velocity in units/minute |
| FOV | Set feedrate override |
| FVEL | Set final velocity |
| IVEL | Set initial velocity |
| JRK | Set jerk parameter (S-curve) |
| LOOK | Lookahead mode |
| MBUF | Multiple move buffer mode |

| | |
|---|---|
| ROV | Set rapid feedrate override |
| SRC | Set external time base |
| STP | Set stop ramp |
| SYNC | Synchronization mode |
| TMOV | Set time-based move |
| TOV | Time override |
| VECDEF | Define automatic vector |
| VECTOR | Set manual vector |
| VEL | Set target velocity for a move |

# Startup Programs

You can set a program to automatically run on powering up or rebooting the controller. The PBOOT command provides that ability.

The PBOOT command must appear as the first statement in a program. From a terminal, sending the PBOOT command starts all PBOOT programs. Every program can use PBOOT.

***Example***

The following program runs on power-up, flashing output 32.

```
PROGRAM
PBOOT : REM PBOOT must appear as first line
REM Beginning of loop
_LOOP1
BIT 32 = NOT BIT 32
DWL 0.25
GOTO LOOP1
ENDP
```

# Resetting the Controller

When you reset the controller, it shuts down communications, turns off outputs, and kills all programs. For controllers with non-volatile memory, the controller stores all conditions.

There are three ways to reset the ACR series controller:

- Cycle power.
- Send the REBOOT command from the Terminal Emulator.
- Send a binary reboot request, typically done via ComACRServer.

# Memory

Memory allocation is completely customizable on the ACR series controllers. The `DIM` commands allocate memory to program, global and local variables, and communication streams.

Once you have allocated memory, you cannot change it without first clearing the memory space. Otherwise, you receive a "re-dimensioned block" error.

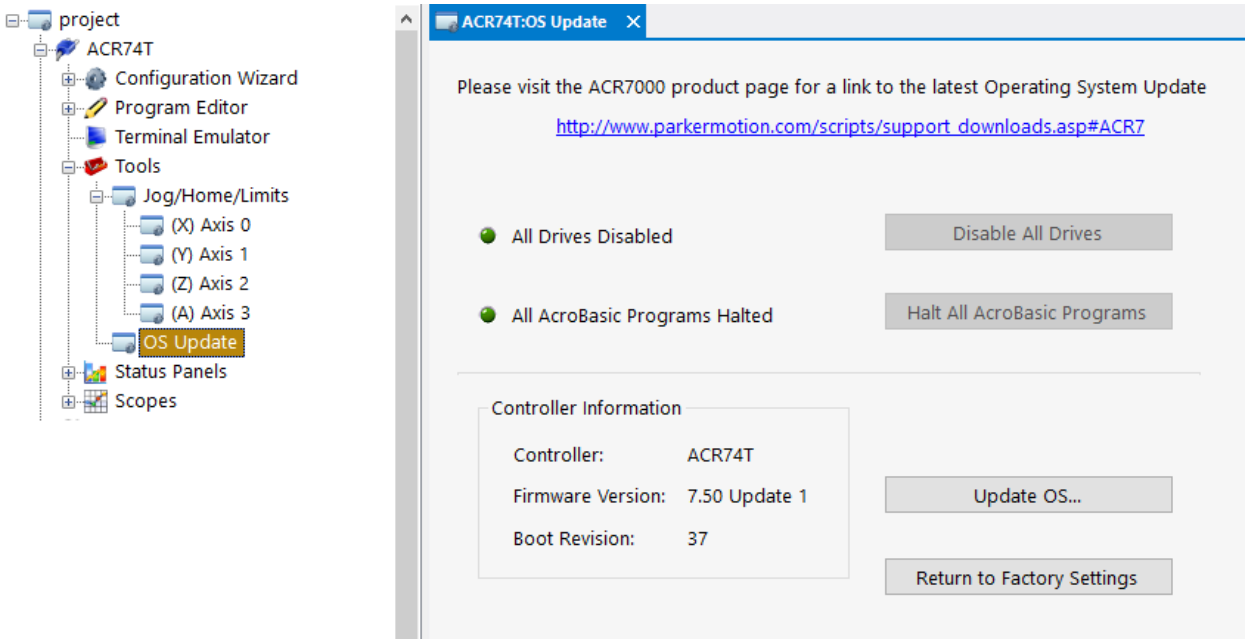
To change memory allocations, use PMM's Configuration Wizard and download to the controller.

# Return to Factory Default

To erase the controller's programs and settings and reset back to factory default:

1. Open Parker Motion Manager
2. Connect to the controller.
3. Open the Terminal Emulator.
4. Type `FLASH RES` and press *Enter*.

Or, from PMM's OS Update screen, click *Return to Factory Settings*.




Before starting a new project on any ACR controller, ensure the controller is set to factory default settings and up to date with the latest operating system.

This will reset the IP address to the default of 192.168.100.1 for the ACR7000 or, for the IPA, to 192.168.100.x where "x" is the rotary dial value on the front of the IPA.

New controllers will always ship with the latest OS.

# Configuration

Because the ACR series controller is powerful and flexible, it requires configuration for your particular application. There are two methods: you can manually write the configuration code or use the Configuration Wizard in the Parker Motion Manager (PMM) software.

As the number of axes increase, the code required to configure a controller can be extensive. The Configuration Wizard helps ensure all constituent devices are configured quickly and correctly.

The configuration code for different models of ACR series controllers varies—dependent on each model's distinct feature set and options, as well as various drives, motors and encoders connected to it. In addition, the firmware revision you have for a controller can affect which features and AcroBASIC commands are available to you.

The wizard makes some choices for you behind the scenes. The ACR7000 controllers are available as an integrated multi-axis stepper drive and controller ACR7xT, integrated multi-axis servo drive and controller ACR7xV, and multi-axis controller ACR7xC. The Configuration Wizard is slightly different for each, showing stepper motors for the ACR7xT, servo motors for the ACR7xV and asking for the drive info for the controller.

PMM generates the system code when *Finish* is selected at the end of the Configuration Wizard. The project is also saved.

**NOTE: The wizard does not collect data in the same order in which code is written.**

# What is Configuration Code?

To get a sense of what configuration code looks like—the requirements and order of items, as well as information that goes into the program space—the following example looks at the code resulting from the PMM Quick Start.

**NOTE: The application is controlled by a 4-axis ACR74T integrated stepper controller.**

## The Code

The wizard generates the Primary System Settings automatically and does not collect data for this. If you are writing your own configuration code, it is good coding practice to include the following at the beginning. The controller is switched to the `SYS` prompt. From there, all program execution is halted (`HALT ALL`), all user programs and PLC programs are deleted (`NEW ALL`), all memory allocations are cleared (`CLEAR`), and all slaves are detached from their respective masters (`DETACH ALL`).

```
REM ------------------------------
REM --- Primary System Settings
REM ------------------------------
SYS
HALT ALL
NEW ALL
CLEAR
DETACH ALL
```

If you do not make any changes to the Memory defaults, the wizard allocates memory to all programs with a large size for Program 0, 10 kB for motion programs Prog 1-Prog 7 and 1 kB for non-motion programs Prog 8-Prog 13. Program 14 is used for PMM's graphing tools (Servo Tuner and Oscilloscope) and thus has a large memory set for onboard data collection. In addition, the wizard allocates memory to Program 15, which stores wizard data. User global parameters P0-P4095 are dimensioned and 100 defines are allocated by default.

```
REM -----Allocate system memory-----
DIM PROG0(100000)
DIM PROG1(10000)
DIM PROG2(10000)
DIM PROG3(10000)
DIM PROG4(10000)
DIM PROG5(10000)
DIM PROG6(10000)
DIM PROG7(10000)
DIM PROG8(1000)
DIM PROG9(1000)
DIM PROG10(1000)
DIM PROG11(1000)
DIM PROG12(1000)
DIM PROG13(1000)
DIM PROG14(200000)
DIM PROG15(100000)
DIM P(4096)
DIM DEF(100)
```

Then begins axis-specific configuration. The axis feedback and signal output information comes from the Axes and Feedback dialogs. The `PPU` (pulses per programming unit) is computed from data provided through the Feedback and Scaling dialogs and units selected. The excess error band data comes from the Fault dialogs. `PM SCALE` is set when a stepper motor with an encoder is used.

```
REM ------------------------------
REM      AXIS 0
REM ------------------------------
ATTACH AXIS0 ENC0 STEPPER0
AXIS0 MULT 4
AXIS0 PPU 30720.000000
AXIS0 EXC (1,-1)
AXIS0 PM SCALE 12.8
SET BIT8469    : REM Enable EXC Response
SET BIT17163   : REM Enable step motor to encoder scaling
```

The Extended I/O section sets and clears bits related to enabling the axis control, drive enable output (DEO) polarity, fault input polarity.

```
REM ACR Extended IO Settings
SET BIT8468  : REM  Enable Drive I/O
CLR BIT8464  : REM Enable CW CCW vs StepDir
CLR BIT8470  : REM DEO Serves Shutdown Function
CLR BIT8453  : REM Invert Drive Fault Input Level
```

The next section is the Axis Gains values. Servo Gains for a stepper axis are set by default and used internally by the controller. Default tuning gains are set for the IPA, ACR7xV and ACR7xC servo axes and can be tuned with the Servo Tuner.

```
REM Axis Gains values
AXIS0 PGAIN 0.00244141
AXIS0 IGAIN 0
AXIS0 ILIMIT 0
AXIS0 IDELAY 0
AXIS0 DGAIN 0
AXIS0 DWIDTH 0
AXIS0 FFVEL 0
AXIS0 FFACC 0
AXIS0 TLM 10
AXIS0 FBVEL 0
```

The Axis Limits section sets the homing, hardware and software limits based on user input on the Fault screen in the Configuration Wizard.

```
REM Axis Limits
AXIS0 HLBIT (0,1,10)
AXIS0 HLDEC 500.000000
SET BIT16144  : REM Positive EOT Limit Level Invert
SET BIT16145  : REM Negative EOT Limit Level Invert
CLR BIT16146  : REM Home Limit Level Invert
SET BIT16148  : REM Positive EOT Limit Enable
SET BIT16149  : REM Negative EOT Limit Enable
AXIS0 SLM (10,0)
AXIS0 SLDEC 500.000000
SET BIT16150  : REM Positive Soft Limit Enable
SET BIT16151  : REM Negative Soft Limit Enable
The ACR7xT has step motor parameters and are set based on the motor
selection.
REM Axis Stepper Motor Settings
P7938=2.38     : REM Max amps peak (user)
P7946=256         : REM Micro Steps (Power 2)
BIT15618=1        : REM Standby Enable flag
P7944=50       : REM Standby Percentage
P7945=0           : REM Standby Delay
BIT8455=0         : REM Invert Motor and Encoder direction
BIT15616=1        : REM Assert Config flag
AXIS0 ON
```

Axis 1 is set the same but note Axis 1 has different addresses for the bits and parameters.

```
REM ------------------------------
REM      AXIS 1
REM ------------------------------
ATTACH AXIS1 ENC1 STEPPER1
AXIS1 MULT -4
```

```
AXIS1 PPU 30720.000000
AXIS1 EXC (1,-1)
AXIS1 PM SCALE 12.8
SET BIT8501    : REM Enable EXC Response
SET BIT17195   : REM Enable step motor to encoder scaling
REM ACR Extended IO Settings
SET BIT8500  : REM  Enable Drive I/O
CLR BIT8496  : REM Enable CW CCW vs StepDir
CLR BIT8502  : REM DEO Serves Shutdown Function
CLR BIT8485  : REM Invert Drive Fault Input Level
REM Axis Gains values
AXIS1 PGAIN 0.00244141
AXIS1 IGAIN 0
AXIS1 ILIMIT 0
AXIS1 IDELAY 0
AXIS1 DGAIN 0
AXIS1 DWIDTH 0
AXIS1 FFVEL 0
AXIS1 FFACC 0
AXIS1 TLM 10
AXIS1 FBVEL 0
REM Axis Limits
AXIS1 HLBIT (0,1,10)
AXIS1 HLDEC 500.000000
SET BIT16176  : REM Positive EOT Limit Level Invert
SET BIT16177  : REM Negative EOT Limit Level Invert
CLR BIT16178  : REM Home Limit Level Invert
SET BIT16180  : REM Positive EOT Limit Enable
SET BIT16181  : REM Negative EOT Limit Enable
AXIS1 SLM (10,0)
AXIS1 SLDEC 500.000000
SET BIT16182  : REM Positive Soft Limit Enable
SET BIT16183  : REM Negative Soft Limit Enable
REM Axis Stepper Motor Settings
P7954=2.38      : REM Max amps peak (user)
P7962=256   : REM Micro Steps (Power 2)
BIT15650=1  : REM Standby Enable flag
P7960=50        : REM Standby Percentage
P7961=0     : REM Standby Delay
BIT8487=0   : REM Invert Motor and Encoder direction
BIT15648=1  : REM Assert Config flag
AXIS1 ON
```

Axis 2 and Axis 3 are not shown for length but have similar settings to Axis 0 and Axis 1. The firmware is structured for multiaxis though the hardware defines the number of axis possible, so we turn off unused axes.

```
REM Turn off any unused Axes
AXIS4 OFF
AXIS5 OFF
AXIS6 OFF
```

```
AXIS7 OFF
AXIS8 OFF
AXIS9 OFF
AXIS10 OFF
AXIS11 OFF
AXIS12 OFF
AXIS13 OFF
AXIS14 OFF
AXIS15 OFF
```

Program 0, a motion program, is attached to Master 0 which is the multiaxis motion trajectory calculator. Axes 0-3 are attached based on the Axes settings in the Config Wizard and the default profiles values are also set.

```
REM ------------------------------
REM --- Program Level setup
REM ------------------------------
 PROG0
DETACH
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH SLAVE2 AXIS2 "Z"
ATTACH SLAVE3 AXIS3 "A"
 REM the desired master acceleration
ACC 10
REM the desired master deceleration ramp
DEC 10
REM the desired master stop ramp (deceleration at end of move)
STP 10
REM the desired master velocity
VEL 1
REM the desired acceleration versus time profile.
JRK 0
```

# *Resources Reserved for Generated Code*

When you click *Finish*, the Configuration Wizard generates the System Code. On download it is saved as XML in Program 15. This allows the Configuration Wizard to be populated when uploading. No changes from the user need to be made.

**NOTE: Do not edit the source files generated by the Configuration Wizard.**

The PMM project file (.pprj) includes the Configuration Wizard settings, the AcroBASIC programs and defines, the servo tuning settings, scope settings, terminal user button settings, oscilloscope motion test code and watch window settings. The project file can have multiple controllers and subsequent controller programs and settings that are also saved within the one project file.

# Flash Memory

The table below describes an overview of the Flash Memory for the ACR Controllers.

| Memory Usage | | | | |
|---|---|---|---|---|
| **SAVE** | **IPA** | **90x0PxUxMx** | **90x0PxUxBx** | **ACR7000** |
| System Parameters (Attachments, Masters) | ESAVE | ESAVE | ESAVE | FLASH IMAGE |
| IP Address | Retained in Non-Volatile USER RAM | ESAVE | ESAVE | ESAVE |
| Axis Set-up Parameters (Gains, PPU, etc.) | ESAVE | ESAVE | ESAVE | ESAVE |
| Motor Configuration | Retained in Non-Volatile USER RAM | N/A | N/A | ESAVE |
| User Programs | FLASH SAVE or FLASH IMAGE | FLASH SAVE or FLASH IMAGE | Programs retained in USER RAM with BBRAM | FLASH IMAGE |
| Local Variables and Arrays | FLASH IMAGE | FLASH IMAGE | Retained in USER RAM with BBRAM | FLASH IMAGE |
| User Global Double Variables P0-4095 | FLASH IMAGE | FLASH IMAGE | Retained in USER RAM with BBRAM | FLASH IMAGE |
| User Global Float Variables P39168-P39423 | Retained in Non-Volatile USER RAM | N/R* | Retained in USER RAM with BBRAM | Retained in Non-Volatile USER RAM |
| User Global Long Variables P38912-P39167 | Retained in Non-Volatile User RAM | N/R* | Retained in USER RAM with BBRAM | Retained in Non-Volatile User RAM |

# CHAPTER 3
# Making Motion

# Making Motion

Now that the controller is configured, it is ready to make motion. The ACR controller can perform linear, circular, or more complex motion with a single axis or multiple axes.

## *Four Basic Categories of Motion*

There are four basic categories of motion used in motion control: coordinated, jog, gear, and cam.

- **Coordinated Moves Profiler (Multi-Axis Profile):** Use the `MOV` command for linear-interpolated incremental and absolute moves. It also allows circular interpolation (`CIRCW`, `CIRCCW`, `SINE`, and `TARC`). The trajectory values are "path" values.
- **Jog Profiler (Single-Axis Profile):** Use the `JOG` commands for incremental, absolute, or continuous moves. The Jog Profiler is axis-independent, meaning that each axis uses its own trajectory values independent of other axes.
- **Gear Profiler (Electronic Gear):** Use the `GEAR` commands to control motion based on an external source—such as a linespeed encoder for feed-to-length, electronic gearbox, trackball, follower axis, or changes of ratio related to position.
- **Cam Profiler (Electronic Cam):** Use the `CAM` commands to control irregular motion using data tables. The Cam Profiler provides control of complex motion and is best used in situations where the desired motion is non-linear using an external source.

Regardless of the type of motion or number of axes used, the controller must always be set up for coordinated motion. This may be done by using the Configuration Wizard or by writing custom configuration code, and including master, slave, and axis attachment statements. The attachment statements make the basic connections to a coordinated motion profiler. For more information, see Attachments.

After making the necessary attachments, a motion profile can be defined. The following sections examine the different move types and motion profilers.

## *Move Types*

To command motion, use a command appropriate to the desired type of motion, such as `JOG` (single-axis profile), `CIRCW` (two-dimensional clockwise circle), `CIRCCW` (two-dimensional counter clockwise circle), `SINE` (sinusoidal move), or `TARC` (3-D arc). The `MOV` (define a linear move) command activates linear-interpolated motion.

When the user includes several axes in a single statement, the controller coordinates the moves, meaning the axes complete their respective moves at the same time. Whereas, if each axis is written as an independent statement, the controller treats them as independent moves and they are performed one at a time.

The `MOV` command is not necessary for coordinated motion because the controller recognizes an axis name and a value as commanded motion, such as `X500`. When multiple axes are written in a single statement, such as `X500 Y100`, the motion is coordinated.

**NOTE: When commanding motion, you must use the axis name; the axis number is not a valid way to indicate an axis. For more information on axis names, see Slaves and Axis Names.**

# Absolute Motion

Absolute motion is commanded with respect to the established "home" or reference location.

To make a linear-interpolated move with the `MOV` command, use the arguments axis target, specifying the axis name followed by the target position.

***Example 1***

The following moves the X axis to the absolute position of 10 units.

```
MOV X10
```

***Example 2***

To command linear-interpolated motion without `MOV`, the axis and position must be designated. The following also moves the X axis to the absolute position of 10 units in an identical manner as Example 1.

```
X10
```

***Example 3***

If motion is commanded for multiple axes on a single line, the controller treats it as coordinated motion. The X and Y axes complete their respective moves at the exact same time.

```
X20 Y-30
```

# Incremental Motion

Incremental motion is commanded relative to the current position.

To move an incremental distance (a distance "relative" to the current position), use a slash mark ( / ) following the axis.

**NOTE: The slash mark is only applicable in linear-interpolated motion.**

***Example 1***

In this example, the X axis moves an incremental distance of 20 units from its current position. Then, the Y axis moves a decremental distance of 30 units from its current position.

```
X/20
Y/-30
```

***Example 2***

The X axis makes an incremental move, Y axis makes an absolute move and Z axis makes a decremental move. Written on the same line, this is a coordinated move; all axes complete their moves at the same time.

```
X/2 Y2 Z/-2
```

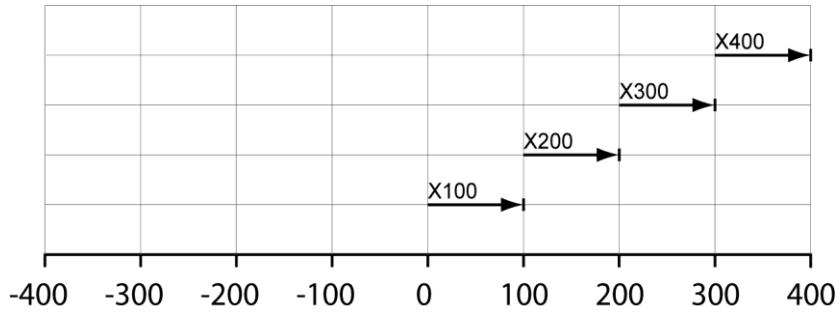
# Comparing Absolute and Incremental Motion

Different types of motion can be used to achieve the same result. The following examples show absolute and incremental motion, and a combination of the two. All three examples end at the absolute position of 400 units.

### *Example—Absolute Motion*

The X axis is commanded to the following absolute positions:

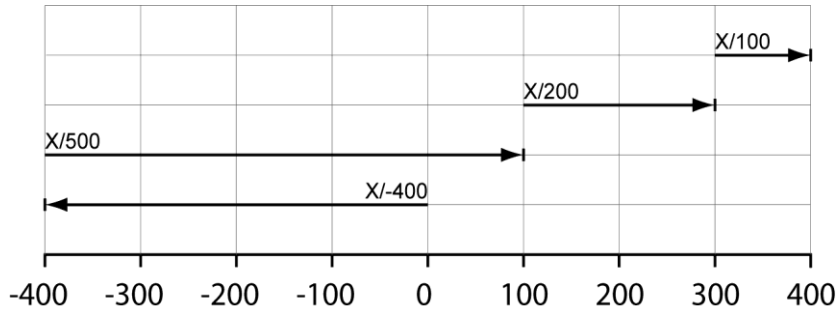
```
X0
X100
X200
X300
X400
```




### *Example—Incremental Motion*

The X axis is commanded to the following relative positions:

```
X0
X/-400
X/500
X/200
X/100
```

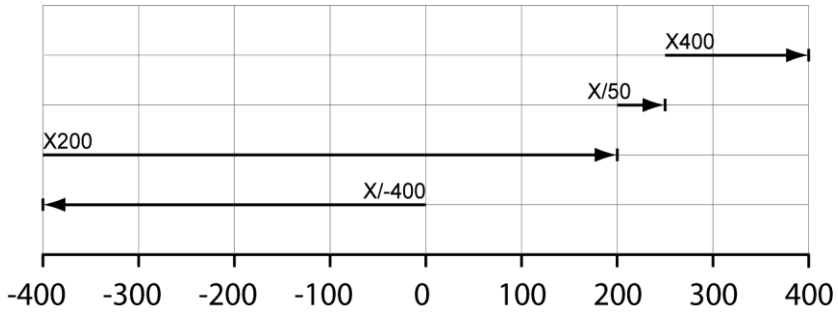



### *Example—Absolute and Relative*

The X axis is commanded to the following absolute and incremental positions.

```
x/-400
x200
```

```
x/50
x400
```




# Combining Types of Motion

The user can command multiple types of motion (linear, circular or sinusoidal) in a single statement. The controller coordinates the motion of all axes in the statement regardless of the type of motion.

***Example***

The following illustrates a coordinated move where the X axis performs linear interpolation and the Y axis performs sinusoidal interpolation.

```
X2 SINE Y(0,90,90,100)
```

# Immediate Mode

While a program is running, the path velocity can be changed for a master and all axes attached to it. The change is instantaneous and takes effect even if the axis or axes are moving.

Use the `FOV` (set feedrate override) command to set a floating-point scaling factor to adjust the master velocity. If a move is in progress, the master uses the established acceleration or deceleration ramp to adjust to the new velocity.

**NOTE: The FOV command does not change the master velocity permanently and the change is not saved. To make a permanent change, adjust the master velocity in the program code either manually or through the Configuration Wizard.**

For more information about feedrate override, see the `FOV` command in the ACR Command Language Reference.

***Example***

The following is typed in at the prompt by the user. It reduces the master velocity for all attached axes to 75%, then 50%, and then returns the velocity to 100%.

```
FOV 0.75
FOV 0.50
```

```
FOV 1.00
```

# Differences Between FOV and VEL

While a program is running, both the FOV and VEL (set target velocity for a move) commands can be set, but each affects motion differently:

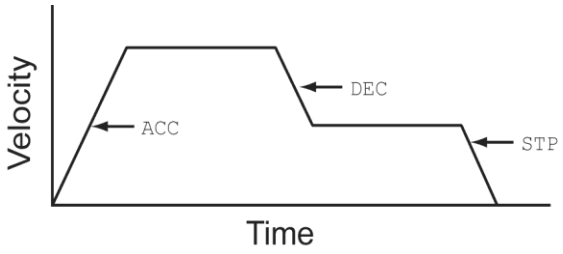
- FOV immediately affects all axes attached to the master.
- VEL is buffered in memory. The newly commanded velocity does not take effect until current motion is completed.

# What are Motion Profiles?

To make motion, the user must define the motion profile. The acceleration, deceleration, stop ramps, velocity, and distance (ACC, DEC, STP, VEL and MOV commands, respectively) set the motion profile values.

- **Acceleration:** The ACC (set acceleration ramp) command sets the master acceleration. The master acceleration is used to ramp from lower to higher speeds. The value is in units per second$^2$.
- **Velocity:** The VEL (set target velocity for a move) command sets the target velocity for subsequent moves. The value is in units per second.
- **Deceleration:** The DEC (set deceleration ramp) command sets the deceleration used to ramp from higher to lower speeds. The value is in units per second$^2$. The deceleration ramp is only used when the stop ramp is zero. Use the DEC ramp to blend moves.
- **Stop:** Use the STP (set stop ramp) command to set the master deceleration ramp used at the end of the next move. The value is in units per second$^2$. When the stop ramp is set to zero, the move ends without ramping down. This allows you to merge back-to-back moves. The final velocity of the first move becomes the initial velocity of the second move.

Motion profiles can be graphically represented. The following illustrates the ACC, DEC, and STP values as a typical trapezoidal motion profile.




All motion profile values are entered in user-based units (inches, millimeters, degrees, revolutions or other units). Use the PPU (set axis pulse-per-unit ratio) command to relate the feedback pulse to the unit of measure. The PPU command sets the ratio of pulses per programming unit. The controller computes the motion trajectory from the motion profile data.

Motion profile values for each master can be set in two ways:

- Through the Configuration Wizard.
- In a program using the appropriate motion profile statements (ACC, DEC, STP or VEL).

In either case, the program continues to use those motion profile values until new values are commanded.

**NOTE: Motion profile values in a specific program can be changed from within a different program using the MASTER (Direct Master Access) command. A master must be attached to each program and is usually the same number as the program number. For more information about masters, see Master/Slave Attachments. For example, to change the velocity in program zero to 500, send the following: MASTER0 VEL500.**

***Example***

The following example assumes a 1000 line encoder attached to a motor. The MULT (set encoder multiplier) command brings the value to 4000. Then, PPU X4000 sets the programming units to revolutions (4000 pulses/rev) for the rest of the program. The X axis moves 200 revolutions at 20 revs/second using 10 revs/second² ramps.

```
MULT X4
PPU X4000
ACC 10
DEC 10
STP 10
VEL 20
MOV X200
```

# *Interaction Between Motion Profilers*

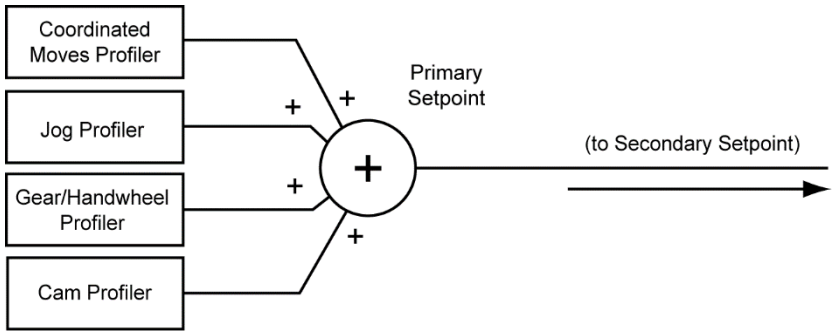
Any combination of motion profilers can be used to carry out motion for an application. As stated previously, the controller must be set up for coordinated motion. Once this is done, the other motion profilers can be accessed through the JOG, GEAR, and CAM commands.

Before writing code, it is important to understand how the motion profilers interrelate:

1. Each motion profiler calculates its own commanded position—the absolute and relative moves for an axis or axes.
2. No motion profiler supersedes another—there is no hierarchy among the profilers.

# *Primary Setpoint*

All profilers feed their commanded positions to a summation point, and the result is the Primary Setpoint for each axis.




In effect, the Jog, Gear and Cam profilers act as offsets to the Coordinated Motion Profiler. The example below demonstrates the offset concept.

***Example***

Suppose an application cuts four diamond shapes from sheets of stock. The program commands motion of axes X, Y, and Z. For simplicity, this example focuses only on the X and Y axes.

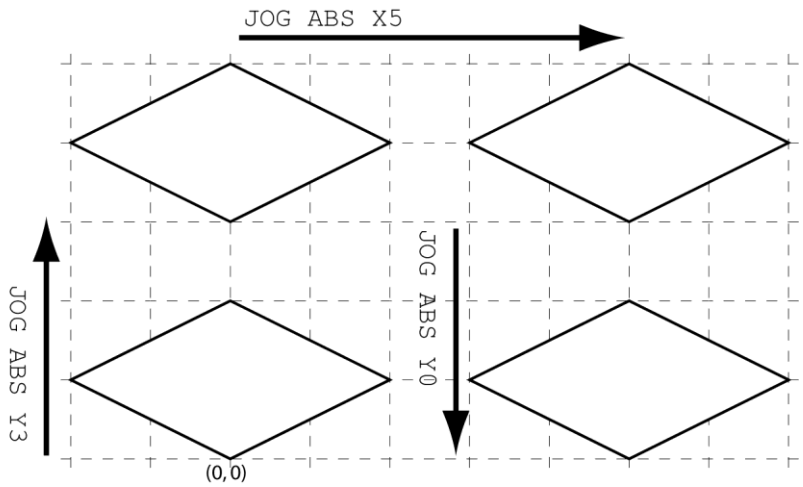
Rather than plotting the cutting motion by providing the coordinates for each diamond, the code in this example provides the coordinates for one diamond and uses the Jog Profiler to offset the coordinates for the remaining diamonds.

The axes are attached to a Coordinated Moves Profiler (see Master/Slave Attachments). The cutting tool starts at coordinates (0, 0) in the lower left quadrant of the stock. Subsequent diamonds are cut in sequence from upper left, upper right, and lower right quadrants. The first shape is cut based on the following moves:

```
X-2 Y1
X0 Y2
X2 Y1
X0 Y0
```

For the second shape, instead of providing a new set of X and Y coordinates, a jog statement is used to shift the Y axis 3 units. You can then provide the same coordinates used to cut the first shape. The new starting position becomes coordinates (0, 3).

```
JOG ABS Y3
X-2 Y1
X0 Y2
X2 Y1
X0 Y0
```




To cut the third and fourth diamond shapes, jog statements again shift the starting positions for axes X and Y. After each jog statement, the coordinates of the first shape are reused.

```
JOG ABS X5
X-2 Y1
X0 Y2
X2 Y1
X0 Y0
JOG ABS Y0
X-2 Y1
X0 Y2
X2 Y1
X0 Y0
```

So what is happening? Each motion profiler calculates its own commanded position, which is sent to a summation point. The coordinated move, jog, gear, and cam data is combined for each axis to create a setpoint.

The Coordinated Moves Profiler always starts and ends at coordinates (0, 0). With the first shape, there are no `JOG`, `GEAR` or `CAM` commands, so the setpoint for the X and Y axes is (0,0):

| Summation Point | X Axis | Y Axis |
|---|---|---|
| Coordinated Moves Profiler | 0 | 0 |
| Jog Profiler | 0 | 0 |
| Gear Profiler | 0 | 0 |
| Cam Profiler | 0 | 0 |
| SETPOINT | 0 | 0 |

For the second shape, the jog statement tells the Jog Profiler to start the Y axis at 3 units. At the summation point, this data is added to the values from the other profilers to yield a Y-axis setpoint of +3:

| Summation Point | X Axis | Y Axis |
|---|---|---|
| Coordinated Moves Profiler | 0 | 0 |
| Jog Profiler | 0 | +3 |
| Gear Profiler | 0 | 0 |
| Cam Profiler | 0 | 0 |
| SETPOINT | 0 | +3 |

For the third shape, the jog statement adjusts the starting point again, this time changing the X axis to 5. The Y axis has not been jogged so it stays at its previous value of +3:

| Summation Point | X Axis | Y Axis |
|---|---|---|
| Coordinated Moves Profiler | 0 | 0 |
| Jog Profiler | +5 | +3 |
| Gear Profiler | 0 | 0 |
| Cam Profiler | 0 | 0 |
| SETPOINT | +5 | +3 |

For the fourth shape, the jog statement adjusts the starting point for the Y axis back to 0. The X axis has not been jogged so it stays at its previous value of +5:

| Summation Point | X Axis | Y Axis |
|---|---|---|
| Coordinated Moves Profiler | 0 | 0 |
| Jog Profiler | +5 | 0 |
| Gear Profiler | 0 | 0 |
| Cam Profiler | 0 | 0 |
| SETPOINT | +5 | 0 |

Without offsets, coordinates for each shape would have to be calculated (and debugged). Instead, one set of coordinates can be reused and the starting point shifted through an offset.

# *Velocity Profile Commands*

A basic motion profile for coordinated motion, controlled by an attached master, consists of acceleration, deceleration, stop ramps and a velocity. You can further control coordinated motion using additional velocity profile commands.

Axis motion with gear, cam or jog offsets are controlled solely by their associated commands—for example, `CAM OFFSET`, `CAM SCALE`, `GEAR ACC`, `GEAR RATIO`, `JOG DEC` or `JOG JRK`.

**NOTE: To check the setting of a specific motion profile command, enter the command without any arguments.**

**NOTE: To disable a command, set its value to zero.**

Use the `ESAVE` command to save coordinated motion and feedback control values in the controller. Otherwise, the system parameters, motion profiles, and master and axis attachments are retained by the controller only until the controller is rebooted or its power cycled. Then all data reverts to its default values.

# *Velocity Profile Setup*

The following commands further shape and refine the coordinated motion profile. For more information about each command, see the ACR Command Language Reference.

F (set velocity in units per minute)—sets a move velocity in units/minute. The F command otherwise functions the same as the VEL command.

FOV (set federate override)—sets the move velocity manually, without changing the current VEL value. Use FOV to superimpose an additional move onto existing motion. Typically, the FOV provides a manual way to change velocity from a terminal. You can also assign the FOV to an input, providing users a manual way to initiate the superimposed move. For more information, see Immediate Mode.

FVEL (set final velocity)—sets a final velocity value. When a STP value has been set, FVEL can be used to set a target final velocity value. The value is used to slow down between moves, but not stop. Moreover, a move only ramps down to the FVEL value, never up to the value.

JRK (set jerk parameter)—sets the slope of acceleration versus time profile. An S-curve profile provides a smoother motion control by reducing the jerk (rate of change) in acceleration and deceleration portions of the move profile. Because S-curve profiling reduces jerk, it improves position tracking performance.

ROTARY (set rotary axis length)—sets a rotary axis length used in a shortest-distance calculation. The resulting move is never longer than half the rotary axis length.

TMOV (time-based move)—sets the time (in seconds) in which the move is completed. The controller calculates a new master motion profile to complete the move in the specified time. The new motion profile values for acceleration, deceleration, stop ramps and velocity are no greater than the user-specified values.

VECDEF (define automatic vector)—controls how the Coordinated Moves Profiler calculates the master move vector. The VECDEF command defines the weight each axis receives in the vector calculation. The default value is 1 for every axis.

In some applications, it is not desirable to include an axis in the motion profile calculation. Suppose there is an application with coordinated motion for axes X, Y, and Z and rotary axis A. Setting the axis A value to zero removes it from the vector calculation. Axis A makes its move within the defined motion profile, but is not part of the calculation itself.

VECTOR (set manual vector)—sets an independent vector value for an axis removed from the motion profile calculation through the VECDEF command. Because the axis is no longer part of the motion profile calculation, it has no master velocity with which it can make independent moves. The VECTOR command provides that value so the axis can make independent moves.

# *Feedback Control Commands*

The feedback control commands affect the velocity profiles and define the encoder feedback used by axes in the current program. Values must be set for each axis.

MULT (set encoder multipliers)—sets the count direction and the hardware multiplication for the encoder of a given axis. This command affects tuning gains, directions, distances, velocities and accelerations.

**Caution:** **Damage to equipment and/or serious injury to personnel may result if MULT is changed to a value inappropriate to the application. Carefully consider the effects throughout the application before applying a new value and perform a test without the load or mechanics attached.**

PPU (set axis pulse per unit ratio)—sets the pulses per programming unit for an axis, allowing convenient units for motion profiles such as inches, millimeters or degrees. The PPU for each axis is independent of that of other axes.

**Caution:** **Damage to equipment and/or serious injury to personnel may result if MULT is changed to a value inappropriate to the application. Carefully consider the effects throughout the application before applying a new value and perform a test without the load or mechanics attached.**

REN (match position with encoder)—sets the commanded position equal to the actual position for a given axis, thus removing the following error.

RES (reset or preload encoders)—sets the commanded position and actual encoder position to zero for a given axis. It also allows the user to preload an axis with a position.

## REN Details

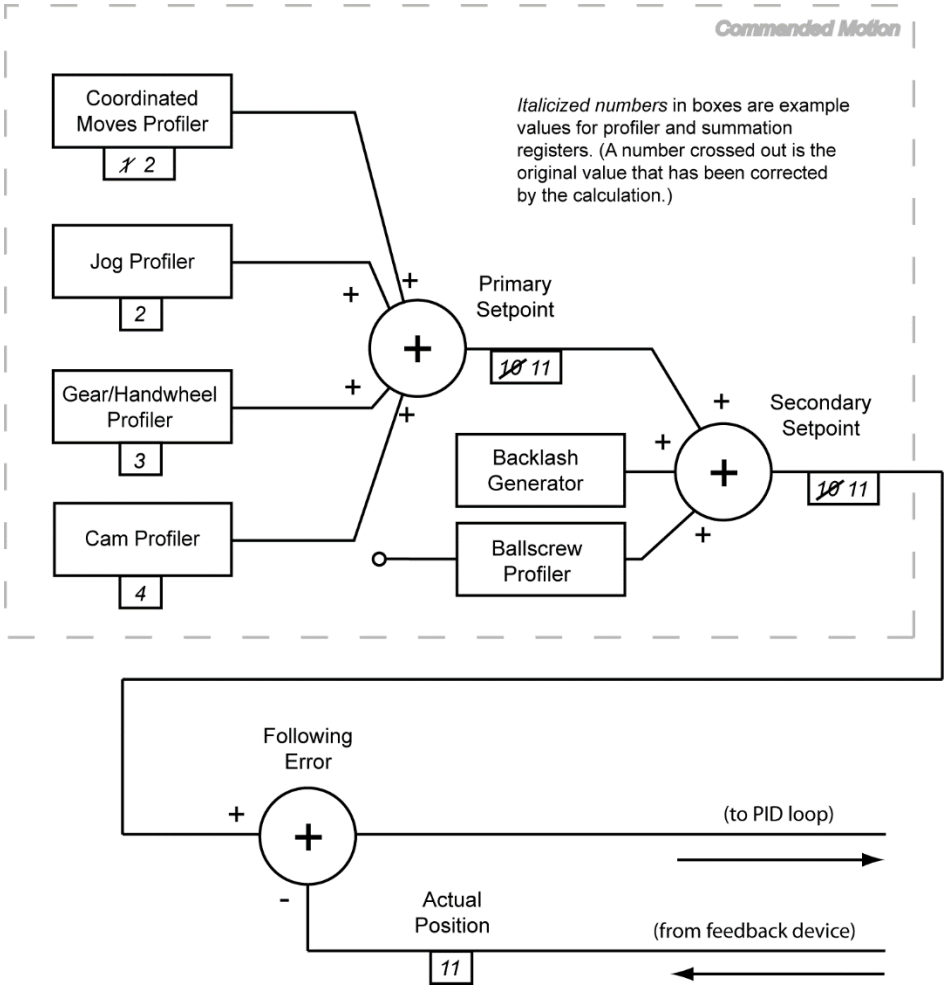
The REN command copies the actual position from the encoder into the Secondary Setpoint of the servo loop. The values for the Primary Setpoint register and for the Coordinated Moves Profiler's offset are then calculated backwards from the Secondary Setpoint. This action removes the following error.

In the example below, the actual position is 11. That number is copied into the register for the Secondary Setpoint, and the Primary Setpoint is then calculated (11).

The Jog, Gear and Cam profilers' offsets do not change. The values in their registers are subtracted from the Primary Setpoint to get the offset for the Coordinated Moves Profiler:

11 – (2 + 3 + 4) = 2

*Calculations for the REN Command*

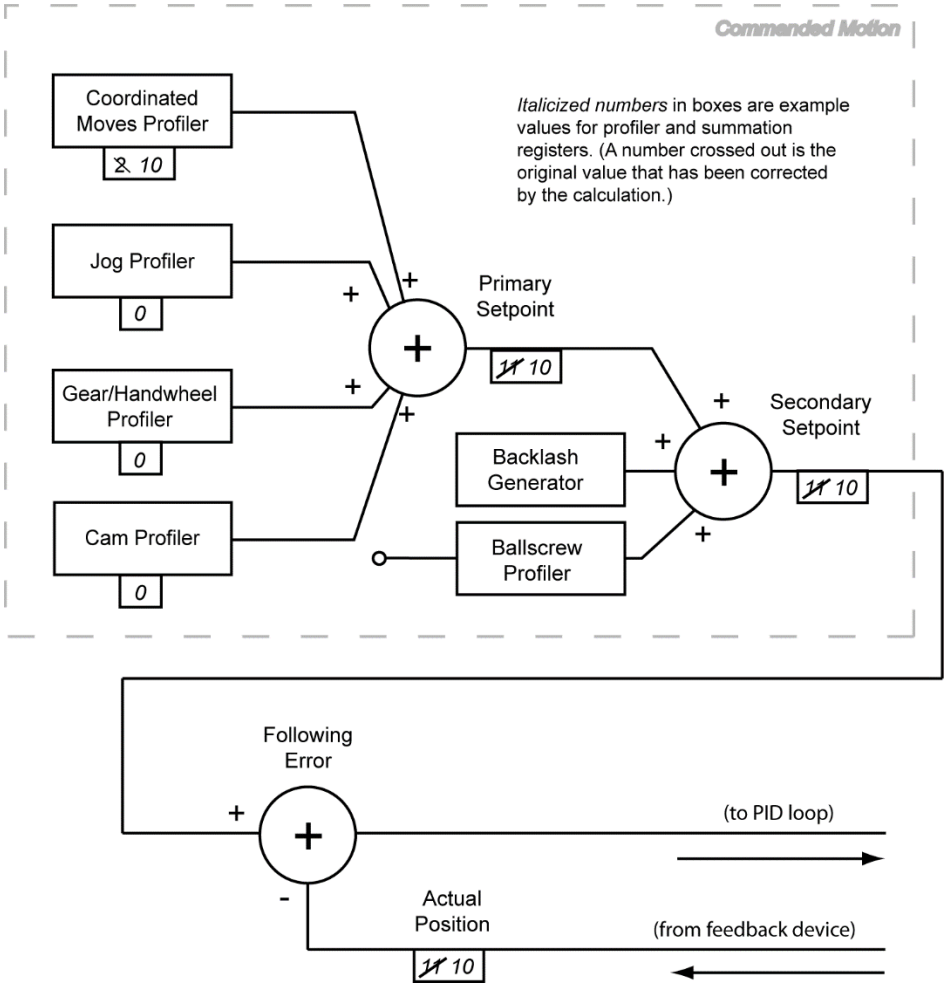



## RES Details

The RES command is used to zero out the primary setpoint (`RES`) or to preload positions into the Coordinated Moves Profiler and Actual Position registers (example: `RES X10`).

See below for a diagram of the profiler and summation registers for the command `RES X10`. The values of the Coordinated Moves Profiler, Primary and Secondary Setpoints and Actual Position registers have been changed to 10. The remaining profilers have been changed to zero.

*Register Values for RES X 10*




If `RES` is used without an axis and preload value, all the registers shown in the above figure would be zero (0).

# *Coordinated Moves Profiler*

The Coordinated Moves Profiler (formerly called the current position profiler) controls motion for multiple axes using a single set of motion profile values. The `MOV` command (define a linear move) commands absolute and incremental motion.

**NOTE: The MOV command is not necessary for coordinated motion. The controller recognizes the axis name and a value as commanded motion, such as X500. Multiple axes can be commanded in a single code statement, such as X500 Y100; the motion is coordinated.**

No matter what the designed application is, the controller must first be configured for coordinated (linear interpolated) motion. This does not limit the user from simultaneously using the other motion profilers—jog, gear or cam. Information regarding which elements are involved is provided to the Coordinated Moves Profiler by the master, slave and axis attachment statements. The other motion profilers look to the Coordinated Moves Profiler for the configuration data. For more information about making attachments, see Attachments.

When multiple axes are moving, the Coordinated Moves Profiler computes the vector based on all the axes' target points. The vector moves at the values set through the motion profile (ACC, DEC, STP, and VEL) and is scaled for each axis. Therefore, all axes start, accelerate, decelerate and stop at the same time.

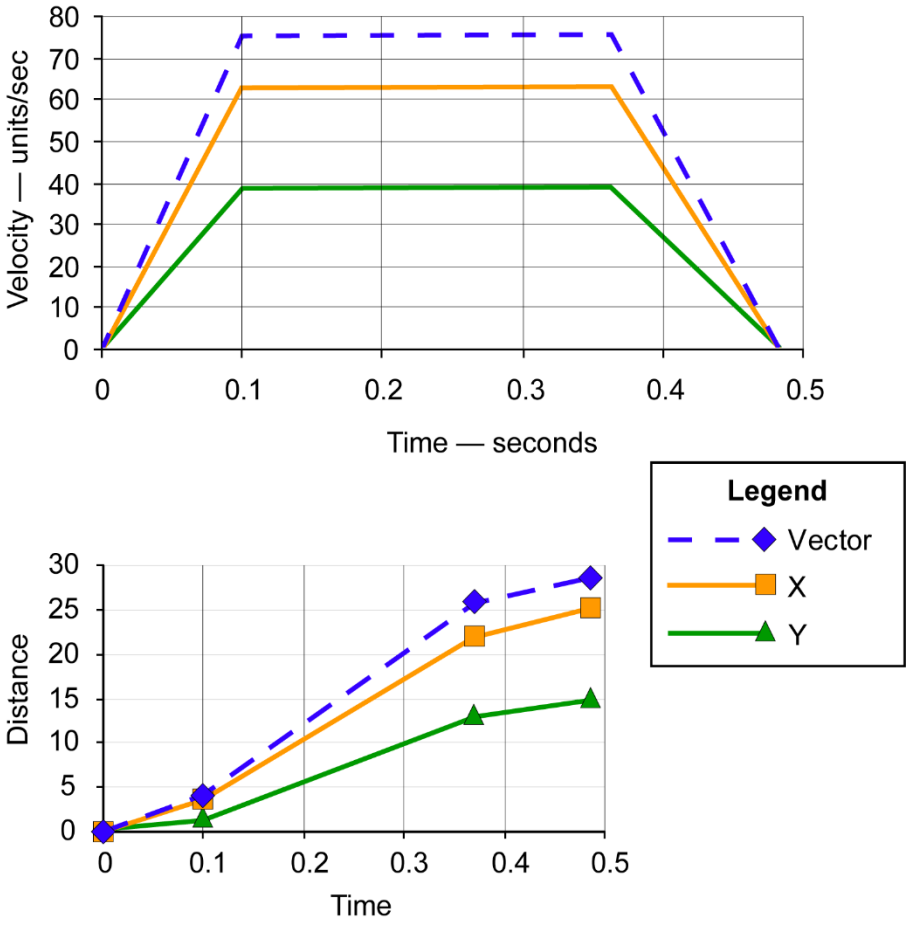
When only one axis is moving, the ACC, VEL and STP are the same as the master.

**NOTE: The Coordinated Moves Profiler typically uses the clock as its timebase.**

***Example 1***

Two axes are attached to the same master and instructed to move to absolute positions: axis X to 25 millimeters and axis Y to 15 millimeters. Both axes start, accelerate, decelerate and stop together.

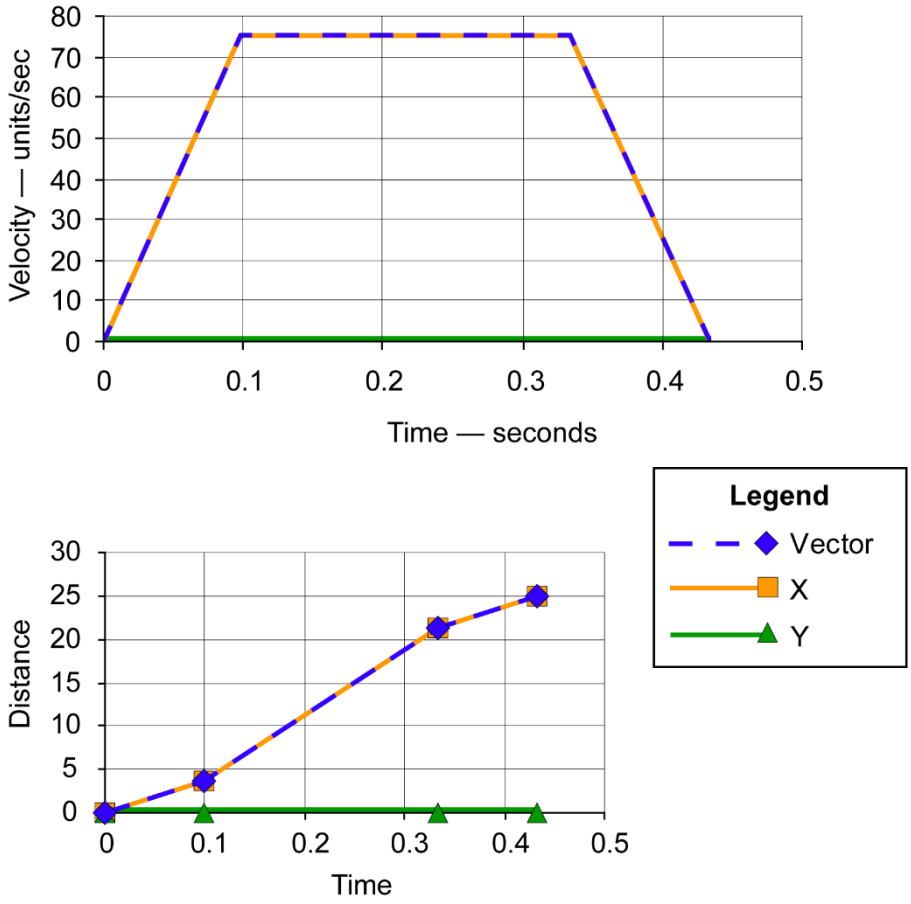
```
ACC 750 DEC 750 VEL 75 STP 750
X25 Y15
```




***Example 2***

Two axes are attached to the same master and the program moves one axis to an absolute position: axis X to 25 millimeters. As only axis X is commanded to move, axis Y is not included in the motion trajectory calculation.

```
ACC 750 DEC 750 VEL 75 STP 750
X25
```




# *Jog Profiler*

Each axis has a dedicated Jog Profiler which can, using a set of motion profile values, control absolute, incremental, or continuous motion for that axis. It can do this independently or in conjunction with the other profilers (Cam, Gear and Coordinated Moves).

**NOTE: Multiple axes may be commanded in a single jog statement, such as JOG ABS X500 Y100. The motion is not coordinated.**

For any application, the controller is first configured for coordinated motion. This does not exclude simultaneously using the other motion profilers.

The Jog Profiler looks to the Coordinated Moves Profiler for its configuration data (master, slave, and axis attachment statements). For more information about making attachments, see Attachments.

The Jog Profiler computes motion based on axis target positions and on the motion profile values (`JOG ACC`, `JOG DEC`, `JOG JRK` and `JOG VEL`). The motion profile is scaled by the `PPU` (pulses per programming unit) for each axis. All axes may start, accelerate and decelerate at different times.

**NOTE: The Jog Profiler typically uses the clock as its timebase.**

**NOTE: The ACR controller uses the Jog Profiler for jogging and homing routines. If the acceleration, deceleration, velocity and jerk values are set for jogging, those values are also used for homing. Therefore, it is a good programming practice to declare the motion profile at the beginning of every jog subroutine. Doing so ensures the correct motion values are used for a jogging or homing routine, regardless how the program branches to a subroutine.**

**NOTE: The Configuration Wizard contains a Jog/Home Commissioning dialog. The dialog only allows the user to test the setup of an axis—it does not produce jogging or homing code.**

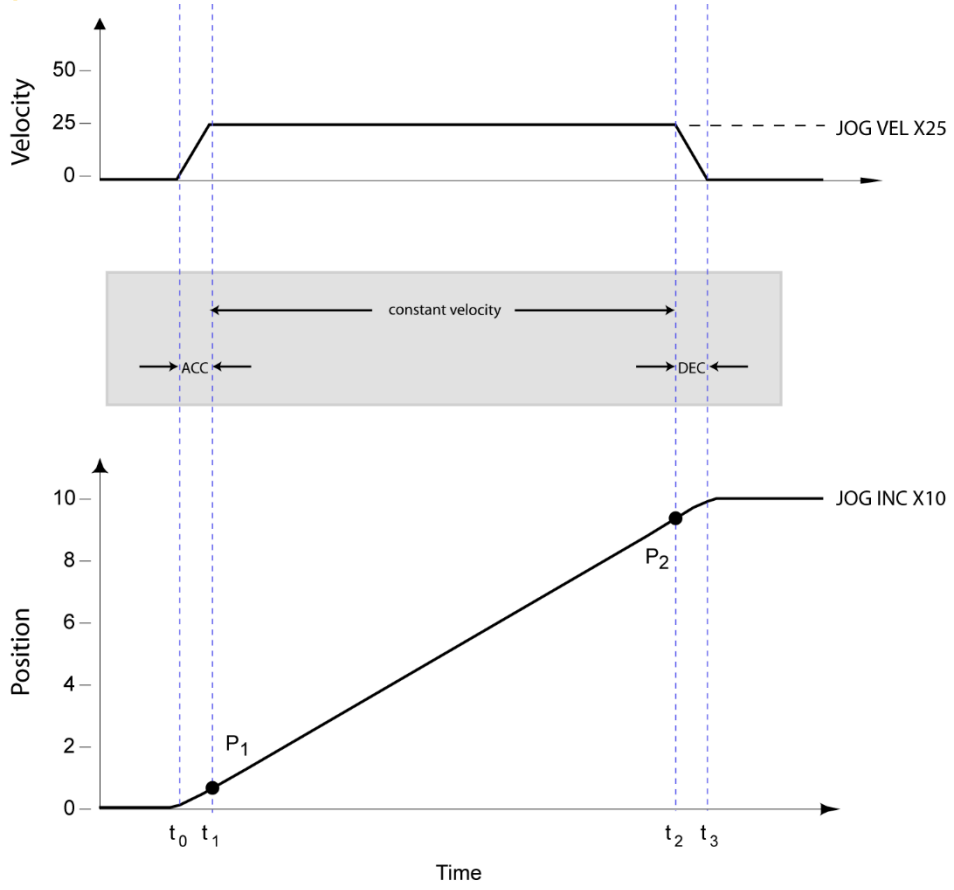
### *Example 1*

Two axes are set to different acceleration, deceleration and velocities, and are moved the same distance.

```
JOG ACC X1000 Y500
JOG DEC X1000 Y500
JOG VEL X25 Y50
JOG INC X10 Y10
```

The figure below looks at the commanded motion of the X axis. In the upper graph (velocity motion profile), `JOG ACC` and `JOG DEC` determine the acceleration and deceleration values, which always graph as ascending and descending slopes, respectively. `JOG VEL` always graphs as a horizontal line once the axis is up to speed. The area under the velocity profile graph is the distance traveled.

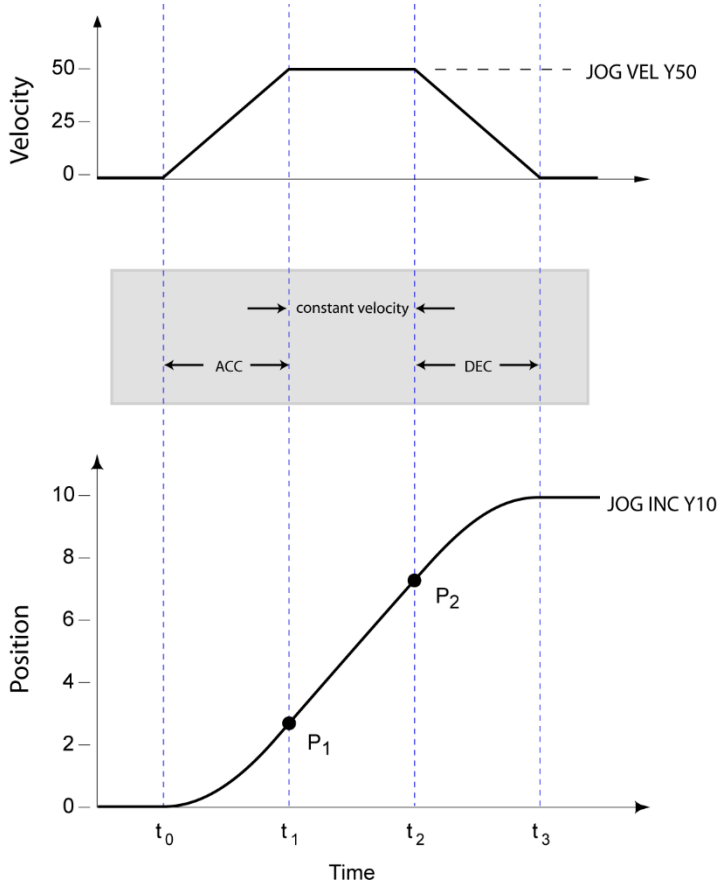
### *X Axis Velocity and Position Profiles*



In the lower graph (position motion profile) of the previous figure, the curve between $t_0$ and $t_1$ shows the change in position during the time it takes for the X axis to accelerate from zero to the target velocity. Likewise, the curve between $t_2$ and $t_3$ shows the change in position during deceleration to zero. The actual acceleration and deceleration curves shown are approximated due to the resolution of the graph. The straight line between points $P_1$ and $P_2$ is where the X axis movement is a constant velocity.

The next figure looks at the movement for the Y axis, characterized by more gradual slopes for acceleration and deceleration values of 500 in the velocity motion profile (as compared to the X axis' values of 1000).

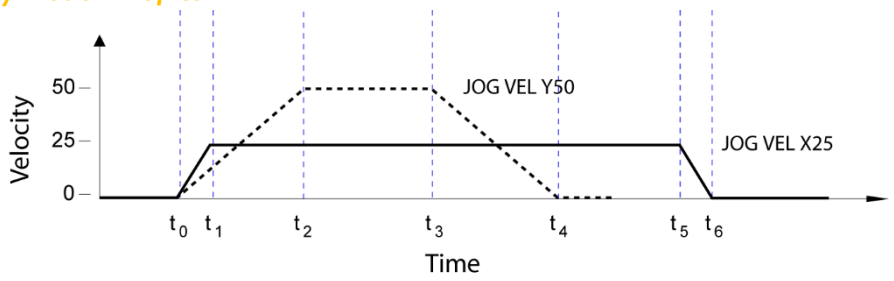
### *Y Axis Velocity and Position Profiles*




Again, the straight line between points $P_1$ and $P_2$ on the position motion profile is where the Y axis movement is at a constant velocity.

The figure below shows the velocity motion profiles for both the X and Y axes superimposed. The Y axis is dashed. Due to a higher `JOG VEL` value, the Y axis finishes its commanded motion in less time than the X axis.

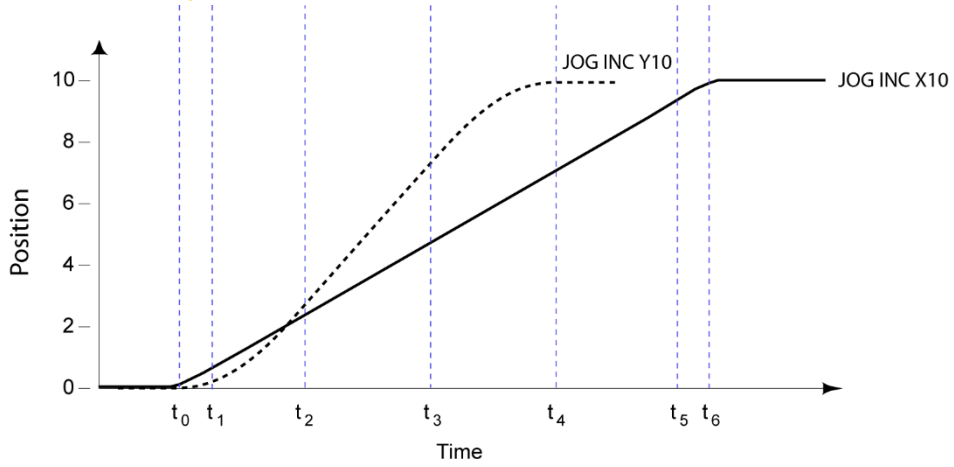
### *X and Y Velocity Motion Profiles*




The following figure graphs the change in position for the X and Y axes. The Y axis is dashed. The overall slope of the position curve for the Y axis is steeper, reflecting its higher `JOG VEL` value (`JOG VEL X25 Y50`).

Comparing the first curve after $t_0$ for the axes show that a higher acceleration value presents as a more gradual curve (`JOG ACC X1000 Y500`).

### *X and Y Position Motion Profiles*



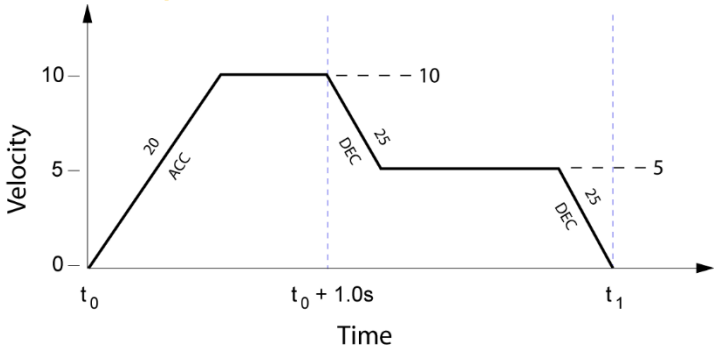

### *Example 2*

The `JOG VEL` value is changed while a single axis is in motion (on the fly—OTF).

```
JOG ACC X20
JOG DEC X25
JOG VEL X10
JOG INC X10
DWL 1.0
JOG VEL X5
```

At one second ($t_0$ + 1.0 s), the axis is commanded to decrease speed to the new velocity. See below for the velocity profile. Motion ends at $t_1$.

### *Change in JOG VEL Value "On the Fly"*

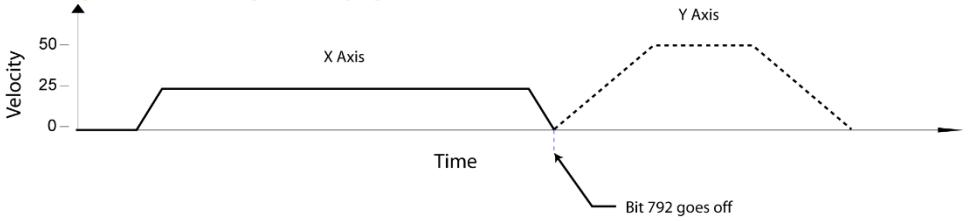



### *Example 3*

To illustrate sequential jog moves, two axes are attached to the same program. The program moves each axis an incremental distance of 10 units using two separate moves. The program waits until the Jog Active Bit (bit 792) is off, indicating that Axis X has finished its move, after which time the Y axis is commanded to move to its incremental position. The figure below shows the velocity profile of this example.

```
JOG ACC X1000 Y500
JOG DEC X1000 Y500
JOG VEL X25 Y50
JOG INC X10
INH -792
JOG INC Y10
```

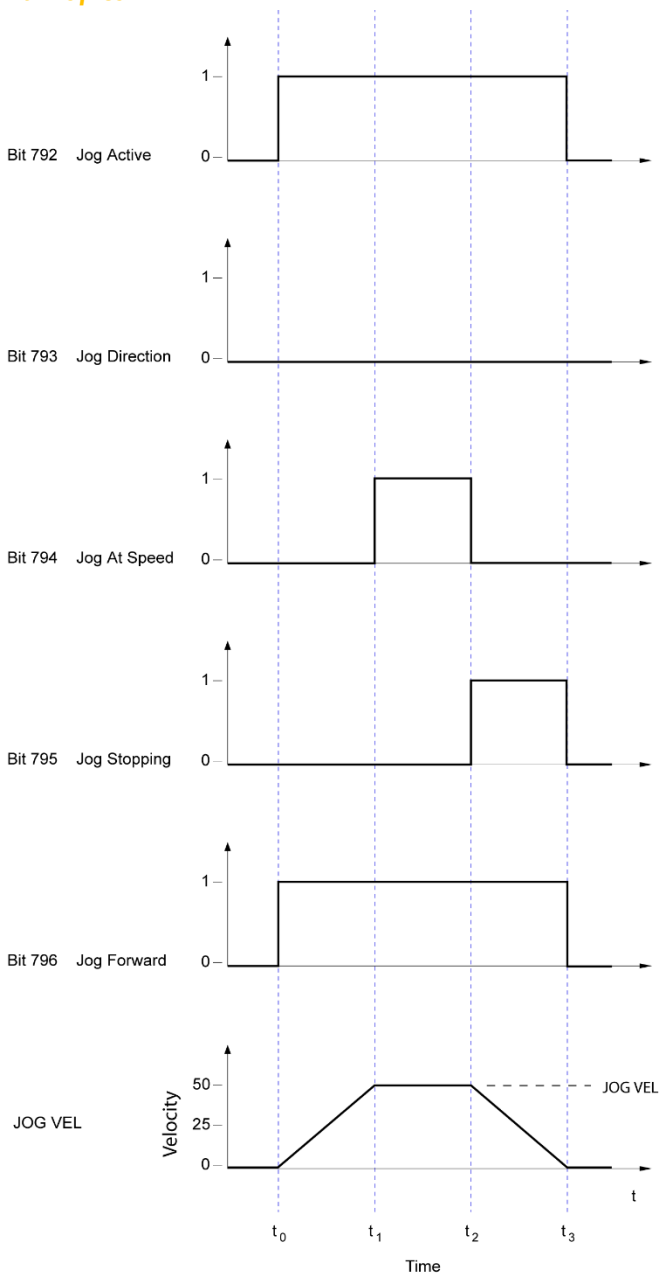
### *Velocity Profile of Sequential Jog Moves*




# *JOG VEL Details*

The next figure shows the bit profiles for the Jog Flags (bits 792 through 796) as a `JOG VEL` command is executed.

### *JOG VEL Command and Bit Profiles*




# *JOG Commands*

See the ACR Command Language Reference for detailed information, including necessary arguments, on `JOG` (single axis velocity profile) and its associated commands:

- `JOG ABS` (jog to absolute position)—uses the current jog settings to jog an axis to an absolute jog offset.
- `JOG ACC` (set jog acceleration)—sets the programmed jog acceleration for an axis.
- `JOG DEC` (set jog deceleration)—sets the programmed jog deceleration for an axis.
- `JOG FWD` (jog axis forward)—initiates a ramp to the velocity programmed by the `JOG VEL` command.

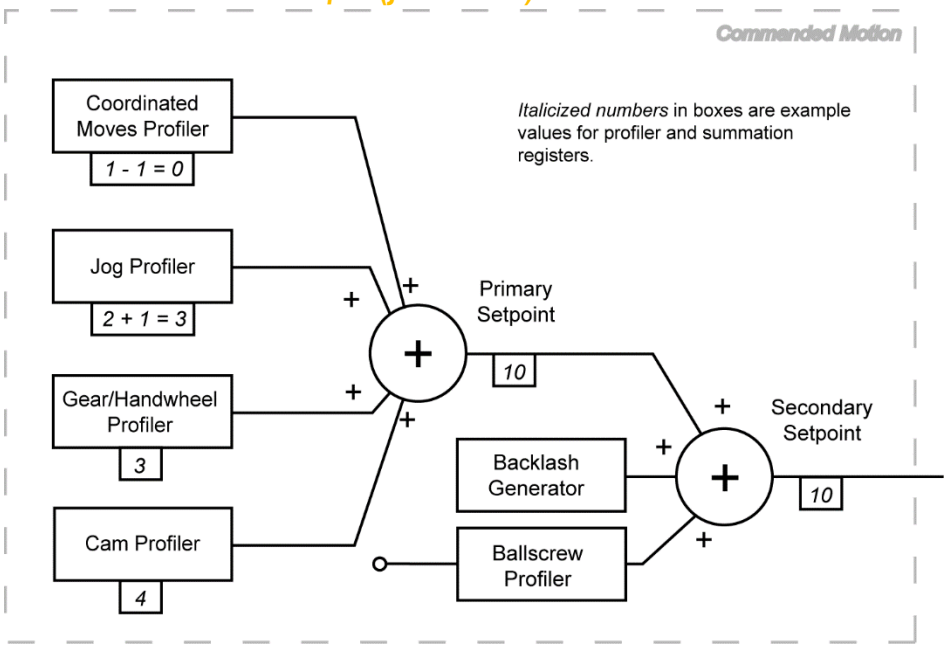
- JOG HOME (go home)—instructs the controller to search for the home position in the direction and on the axes specified.
- JOG HOMVF (home final velocity)—specifies the velocity to use when the homing operation makes the final approach.
- JOG INC (jog an incremental distance)—uses the current jog settings to jog an axis an incremental distance from the current jog offset.
- JOG JRK (set jog jerk (S-curve))—controls the slope of the acceleration versus time profile.
- JOG OFF (stop jogging axis)—initiates a ramp down to zero speed.
- JOG REN (transfer current position into jog offset)—either clears or preloads the current position of a given axis and adds the difference to the jog offset parameter.
- JOG RES (transfer jog offset into current position)—either clears or preloads the jog offset of a given axis and adds the difference to the current position.
- JOG REV (jog axis backward)—initiates a ramp in the negative direction to the velocity programmed with the JOG VEL command.
- JOG SRC (set external timebase)—specifies the timebase for jogging.
- JOG VEL (set jog velocity)—sets the programmed jog velocity for an axis.

# JOG REN Details

The JOG REN command (transfer current position into jog offset) clears the Coordinated Moves Profiler of a given axis and adds the difference to the Jog Profiler offset (example: JOG REN X). It can also be used to preload a position into the Coordinated Moves Profiler, adjusting the Jog Profiler to make up the difference (example: JOG REN X2). In either case, the Gear and Cam profilers and the Primary and Secondary setpoints do not change.

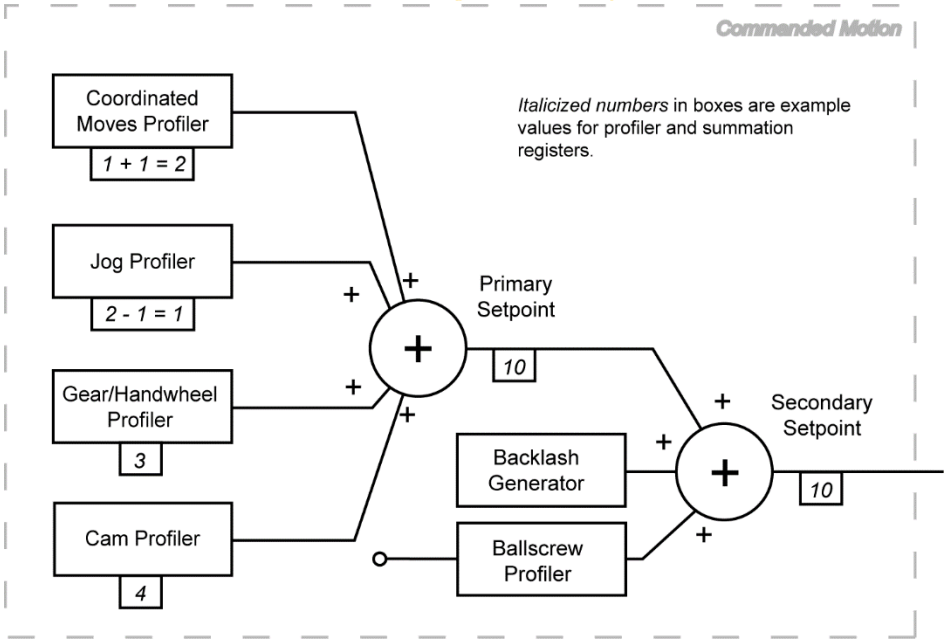
The drawing below illustrates JOG REN as it clears the Coordinated Moves Profiler.

*JOG REN Clears Coordinated Moves Profiler (JOG REN X)*



The drawing below illustrates `JOG REN` as it preloads the Coordinated Moves Profiler.

*JOG REN Preloads the Coordinated Moves Profiler (JOG REN X2)*

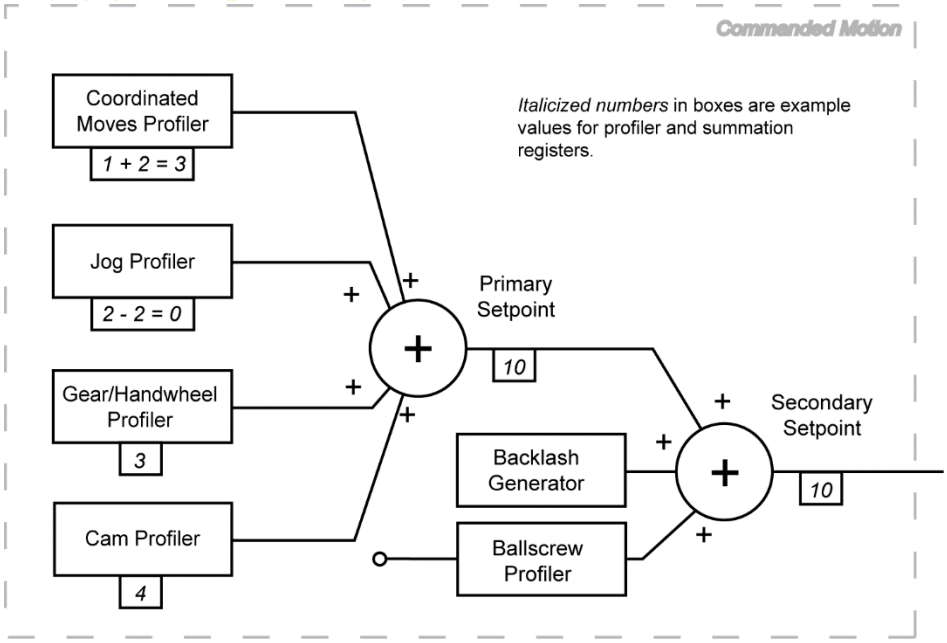



# JOG RES Details

The `JOG RES` command (transfer jog offset into current position) clears the Jog Profiler offset of a given axis and adds the difference to the Coordinated Moves Profiler (example: `JOG RES X`). It can also preload the Jog Profiler offset, and, again, adjusts the Coordinated Moves Profiler to make up the difference (example: `JOG RES X2`). In either case, the Gear and Cam profilers and the Primary and Secondary setpoints do not change.

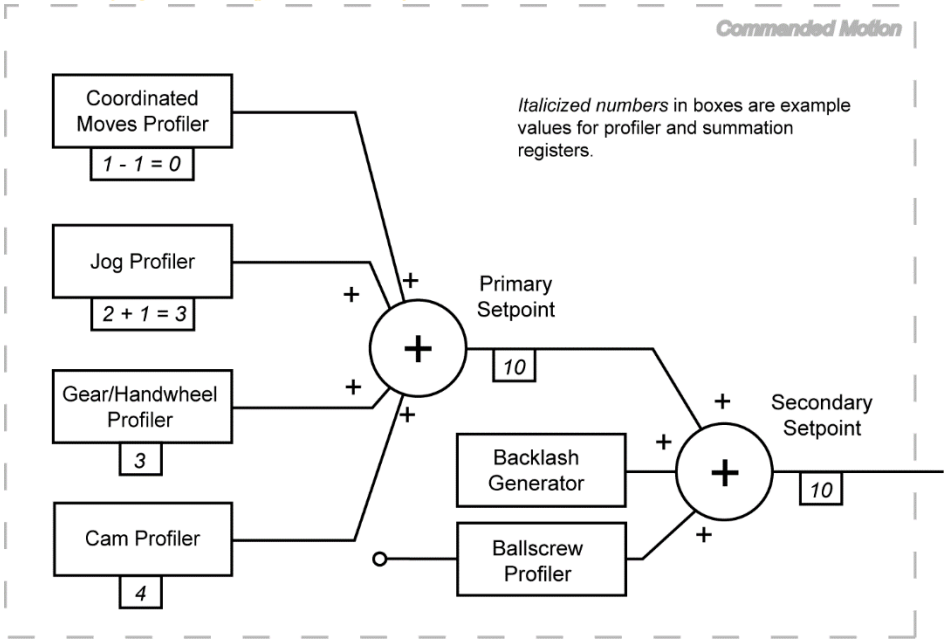
The drawing below illustrates JOG RES as it clears the Jog Profiler.

### JOG RES Clears the Jog Profiler (JOG RES X)




The drawing below illustrates `JOG RES` as it preloads the Jog Profiler.

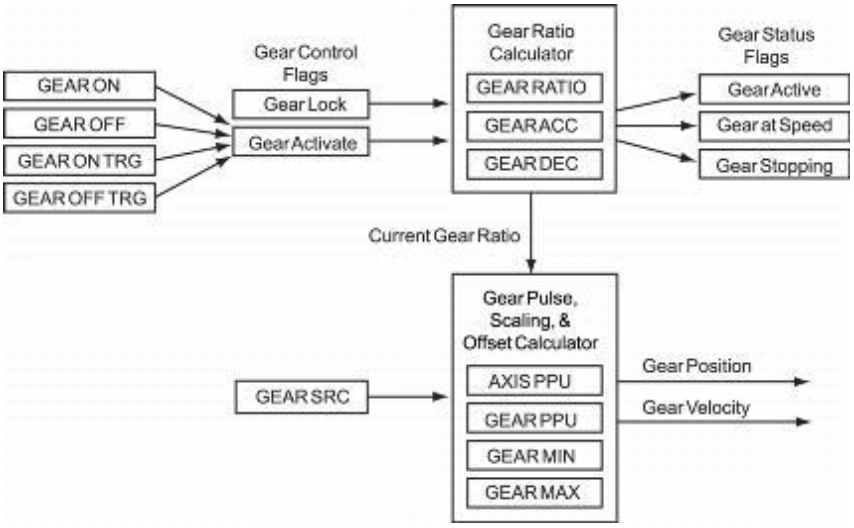
### JOG RES Preloads the Jog Profiler (JOG RES X2)




# Gear Profiler

The Gear Profiler controls motion for axes needing to match their motion output to some form of input. The input source is usually external, such as an electronic gearbox, trackball, follower axis or changes of ratio related to position. In electronic gearing, pulses are fed from a selected source into the gear offset of a slave axis. These

pulses are scaled by a ratio that is equivalent to a gear ratio on a mechanical system. The rate at which the ratio changes is controlled by a ramping mechanism similar to a clutch or a variable speed gearbox.




### *Simple Gear Example—Gearing to an Axis*

```
GEAR SRC X P12546 : REM Gear X to Actual Position of Axis 1.
GEAR PPU X51200   : REM Master is 51200 pulses per rev.
GEAR RATIO X.25   : REM Set gear ratio at 1/4 master.
GEAR ON X         : REM Turn electronic gearing on.
```

For each revolution of Y, X would move 0.25 inch. An external encoder could be used (Encoder 8 or, for IPA, Encoder 1). The above `PPU` is for an ACR7xT stepper axis. The master's axis `PPU` could be used as `GEAR PPU` instead to set similar units.

The external encoder input can also be a gear source:

```
GEAR SRC X ENC1   : REM Gear X to Actual Position of Axis 1.
GEAR PPU X 4000   : REM Master is 4000 pulses per rev.
GEAR RATIO X-1.5  : REM Set gear ratio at -1.5 of master.
GEAR ON X         : REM Turn electronic gearing on.
```

No external encoder wired to the controller? Use the global clock:

```
P6916=0           : REM Reset Global System Clock to 0.
GEAR SRC X P6916  : REM Tie slave gearbox to Global System Clock.
GEAR PPU X1000    : REM Master is 1000 counts (1 second).
GEAR RATIO X.25   : REM Set gear ratio at 1/4 (0.25 in/rev).
GEAR ON X         : REM Turn electronic gearing on.
```

The above simple examples do not set `GEAR ACC` or `GEAR DEC`. As soon as gearing is enabled, X will use infinite acceleration to match speed with its gear source. If the source to which you are gearing is already moving at high speed or you have a high gear ratio, this can cause high jerk in the system or cause the axis to accelerate at a very high rate; hence `GEAR ACC` and `GEAR DEC` can be used to limit the acceleration and deceleration ramps.

`GEAR RATIO` can be changed while gearing is active, but programmers should be careful; large changes in the ratio can lead to abrupt changes in velocity.

*Gearing Example—Start Gearing on High-Speed Input*

```
GEAR ACC X10000
GEAR DEC Y10000
GEAR SRC Y0       : REM Gear Y axis to ENC0.
GEAR RATIO Y1     : REM Gear ratio of 1/1.
X/ 200000         : REM X axis move.
GEAR ON Y TRG(2, 0) OFFSET 3000
REM Mode 2, Rising Primary external.
REM Capture Register 0, gear source is ENC0.
REM Offset is positive, X-axis is moving
REM in positive direction.
INH 2344
REM Wait, capture register is shared by GEAR TRG ON
REM and GEAR TRG OFF.
GEAR OFF Y TRG(2,0) OFFSET 6500
REM The gear will turn off 6500 pulses after the
REM trigger is received.
INH 2348
```

For `GEAR SRC` see the `SRC` command (set external timebase) for available sources. Parameters can be used but care should be taken that these do not change abruptly (mistakenly written to from another program, PLC or HMI) or are subject to noise corruption.

**NOTE: The Gear Profiler typically uses a source other than the clock as its timebase.**

Gantry Lock is a special application as compared to gearing. See Lock.

# *Cam Profiler*

An electronic cam is primarily used as a replacement for a mechanical cam. The Cam Profiler controls motion for axes needing precise motion. It uses an array of target points in relation to an externally sourced timebase. By breaking the motion into discrete target points, the cam arrives at the exact point needed. The source can be the position of another axis, an external encoder or any parameter within the controller.

The Cam Profiler provides linear interpolation between points, regardless of how many points are necessary for the move. All changes in motion are real time. The Cam Profiler does not compile motion.

Cam uses an arbitrary source to generate an index into a table of offset values. If this index falls between two table entries, the cam offset is linearly interpolated between the entries. This offset is then scaled, shifted by the output offset, and then multiplied by the `PPU` for the given axis.

A cam table can be composed of more than one segment with each segment having different distances between table entries. The data for each segment of the table resides in separate long integer arrays, possibly of different sizes. This allows some parts of the table to be defined coarsely and others to be defined in more detail. Each point of the cam table is scaled by `PPU` of the cam axis.

You can only use long integer arrays in a cam table. The table index automatically tracks which segment it is in and where it is within that segment. It also wraps around if it goes off either end of the table. The wraparound point is determined by the total length of the table that is equal to the summation of the individual segment lengths.

**NOTE: The Cam Profiler typically uses a source other than the clock as its timebase.**

**NOTE: The cam table is stored within an array of long integers, not real numbers. Thus, the position data in the cam table would be in counts; convert real positions to counts by multiplying by PPU.**

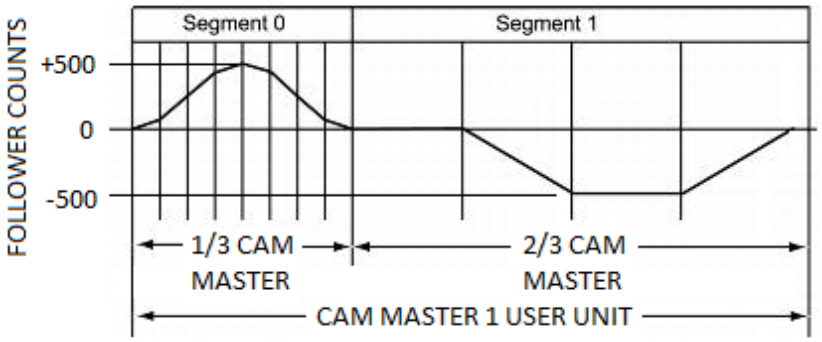
For `CAM SRC`, see `SRC` command (set external timebase) for available sources. Parameters can be used but care should be taken that these do not change abruptly (mistakenly written to from another program, PLC or HMI) or are subject to noise corruption.

`CAM RES` (transfer cam offset)—this command either clears or preloads the cam offset of a given axis and adds the difference to the current position. It also clears out any cam shift that may have been built up by an incremental cam.

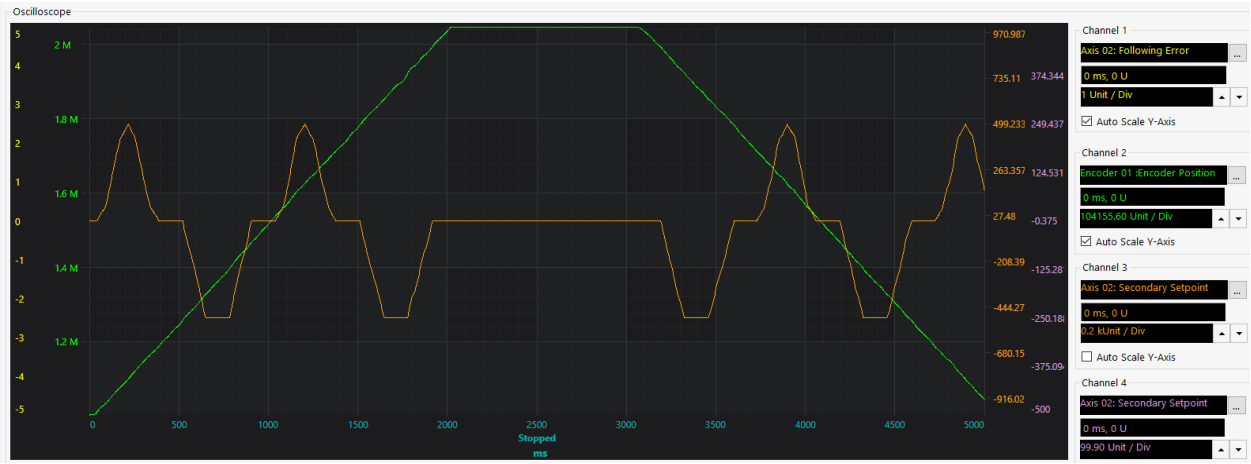
### *Cam Example Program—CAM X to Y Axis*

```
DRIVE ON X Y
REM   START CAM TABLE ARRAY***********
DIM LA(2)    : REM Dimension 2 long arrays
DIM LA0(9)   : REM LA0 has 9 elements
LA0(00)=0    : REM Start defining cam table segment1
LA0(01)=73
LA0(02)=250
LA0(03)=427
LA0(04)=500
LA0(05)=427
LA0(06)=250
LA0(07)=73
LA0(08)=0
DIM LA1(6)
LA1(00)=0    : REM Start defining cam table segment2
LA1(01)=0
LA1(02)=-500
LA1(03)=-500
LA1(04)=0
LA1(05)=0
REM   END CAM TABLE ARRAY***********

CAM DIM X2         : REM Define 2 cam segments
CAM SEG X(0, (P12631*1/3),LA0)
REM Cam segment0, range (counts of src), data table
CAM SEG X(1, (P12631*2/3),LA1)
REM Cam segment1, range (counts of src), data table
CAM SRC X1         : REM Define cam source as ENC1
CAM SCALE X(1/P12375)
REM Set Cam Scaling
```

```
REM Set to 1/(PPU X) for 1/1 relation between cam scale and axis units
CAM SRC X RES      : REM Reset cam source to 0
CAM RES X          : REM Reset cam to 0
CAM ON X           : REM Start camming
Y/2
DWL 1
Y0
```




For each unit of Y moved, X would progress through the cam table, repeating as it moved and reversing if Y is reversed:




Outputs can be set to turn on position automatically with Programmable Limit Switch (PLS). See PLS sample.

# *Homing*

The homing operation is a sequence of moves that position an axis using the Home Limit inputs. The goal of the homing operation is to return the load to a repeatable starting location.

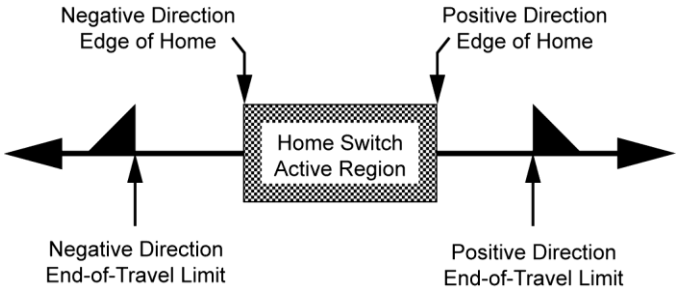
When the homing operation successfully completes, the controller sets the absolute position register to zero, establishing a zero reference position. For servo axes using analog feedback, the controller sets the voltage register to zero.

The Jog Profiler controls homing operations. If the acceleration, deceleration, velocity, and jerk values are set for jogging, those values are also used for homing.

**NOTE: It is a good programming practice to declare the motion profile at the beginning of every jog subroutine. Doing so ensures the correct motion values are used for a jogging or homing routine, regardless how the program branches to a subroutine.**

**NOTE: A homing routine cannot be started for an axis that is already in motion.**

The relevance of positive and negative direction with respect to limit switches is shown below.




If an end-of-travel limit is encountered during the homing operation, motion is reversed and the home switch is sought in the opposite direction. If a second limit is encountered, the homing operation is terminated, stopping motion at the second limit.

**NOTE: For homing operations, always use the clock as the source of the Jog Profiler.**

The controller uses the following guidelines for all backup-enabled profiles:

- Search for the selected edge at the velocity set with the `JOG VEL` command (set jog velocity).
- Use the direction given in the `JOG HOME` command (go home). If the home input is already active, start toward the selected edge. On finding the selected edge, decelerate.
- Return to the selected edge at the velocity set with the `JOG HOMVF` command (home final velocity). If the returning direction is the same as the selected final direction, the profile is complete. Otherwise, find the edge again in the selected final direction using the velocity set with the `JOG HOMVF` command.

***Example***

The homing routine sets the conditions for homing: a motion profile, the inputs related to homing and homing velocity. In addition, specific bit conditions are set out. The `JOG HOME` command then starts the homing process.

The `WHILE/WEND` statement (loop execution conditional) causes the program to wait until the homing conditions it contains are met. In the first `AND` statement, Axis 0 cannot have found home and cannot have failed to find home. The second `AND` statement does the same for Axis 1. Once conditions are met, the code within the `WHILE/WEND` statement is executed.

Finally, the program prints that the Y axis homing is successful and initiates Z channel homing (MSEEK command—marker seek operation) for axis X. When axis X has successfully completed the Z channel homing, the program prints that X axis homing is successful.

```
PROGRAM
JOG VEL X10 Y10          : REM Set axes jog parameters used during homing
JOG ACC X100 Y100
JOG DEC X100 Y100
HLBIT X0 Y3              : REM X uses 1Home (input2), Y uses 2Home (input5)
HLIM X3 Y3               : REM enable EOT limit checking for box axes
JOG HOMVF X0.1 Y0.1      : REM Set backup to home velocity
SET 16144 SET 16145      : REM Invert axis0 level of limit inputs
SET 16176 SET 16177      : REM Invert axis1 level of limit inputs
CLR 16152 CLR 16184      : REM Disable backup to home
CLR 16153 CLR 16185      : REM Look for positive edge of sensor
CLR 16154 CLR 16186      : REM Final homing direction will be positive
JOG HOME X-1 Y1          : REM start homing x negative, y positive
REM The WHILE/WEND statement uses Boolean logic to define homing
REM conditions. Bits 16134 and 16166 are the Found Home bits for axes
WHILE (((NOT BIT 16134) AND (NOT BIT 16135)) OR ((NOT BIT 16166) AND (NOT BIT
16167)))
WEND
IF (BIT 16166) THEN PRINT "Y HOMING SUCCESSFUL"
IF (BIT 16134)
    MSEEK X(1,0)
    INH -516
    IF (BIT 777)
        PRINT "X HOMING SUCCESSFUL"
    ENDIF
ENDIF
ENDP
```

← Line Wraps

# *Homing Subroutines*

Typically, the homing code is a subroutine in a program. The Jog commands define the motion (JOG ACC, JOG DEC, JOG HOME, JOG HOMVF, JOG JRK and JOG VEL) and three bits in the Quinary Axis Flags (bit 16128-16639) control other aspects of a homing routine:

- Home Backup Enable (bit index 24).
- Home Negative Edge Select (bit index 25).
- Home Final Direction (bit index 26).

The JOG HOME command simultaneously homes multiple axes. The arguments for this command, axis and direction, allow the user to specify an axis and the direction in which it seeks the homing region. For example, JOG HOME X1 Y-1 homes the X axis in the positive direction, and the Y axis in the negative direction.

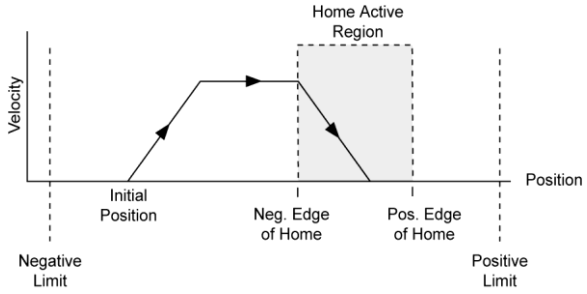
The following diagrams illustrate the combinations and interactions of the three homing bits (above) and the JOG HOME command.

## Basic Homing (Homing Backup Disabled)

When the Home Backup Enable bit (Bit 24) is clear, the controller ignores the Home Negative Edge Select bit (bit 25) and Home Negative Final Direction bit (bit 26). Consequently, when the controller finds any homing edge (positive or negative), the move decelerates. The controller does not attempt to back up to the found edge.

Figures A and B show the homing operation when the Home Backup Enable, Home Negative Edge Select, and Home Negative Final Direction bits are clear (Quinary Axis Flags, bit 16128-16639).

***Figure A***




Homing Profile Attributes:

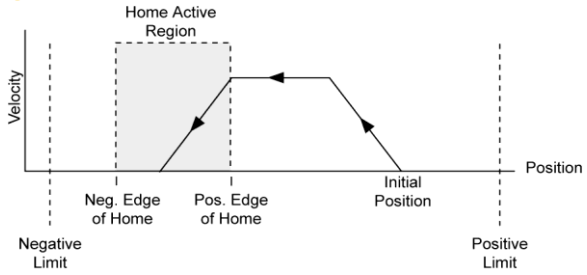
`JOG HOME X1`

Home Backup Enable (bit index 24) is clear.

Home Negative Edge Select (bit index 25) is clear.

Home Negative Final Direction (bit index 26) is clear.

***Figure B***




Homing Profile Attributes:

`JOG HOME X-1`

Home Backup Enable (bit index 24) is clear.

Home Negative Edge Select (bit index 25) is clear.

Home Negative Final Direction (bit index 26) is clear.

## Positive Homing (Homing Backup Enabled)

Figures C through F show the homing operation when the Home Backup Enable bit is set (parameters 4600-4615).

The seven steps below describe a sample homing operation, as illustrated in Figure C. Figures D through F show the homing operation for different values of the Home Negative Edge Select and Home Negative Final Direction bits—the Home Backup Enable bit is set.

A positive home move is started with the `JOG HOME X1` command at the `JOG ACC` and `JOG JRK` accelerations. Default `JOG ACC` is 10 revs (or volts or inches) per sec$^2$.

The `JOG VEL` velocity is reached (move continues at that velocity until home input goes active).

The negative edge of the home input is ignored and the move continues until the positive edge is detected. At this time, the move is decelerated at the `JOG DEC` and `JOG JRK` command values.

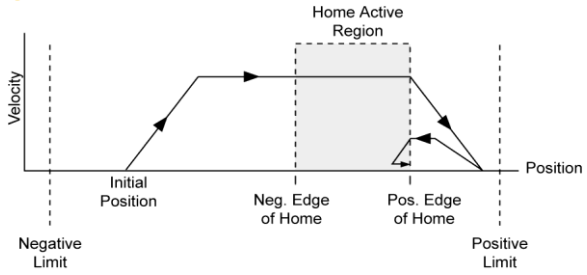
After stopping, the direction is reversed and a second move with a peak velocity specified by the `JOG HOMVF` value is started.

This move continues until the positive edge of the home input is reached.

Upon reaching the positive edge, the move is decelerated at the `JOG DEC` and `JOG JRK` command values, the direction is reversed, and another move is started in the positive direction at the `JOG HOMVF` velocity.

As soon as the home input positive edge is reached, this last move is immediately terminated. The load is at home and the absolute position register is reset to zero.

***Figure C***




Homing Profile Attributes:

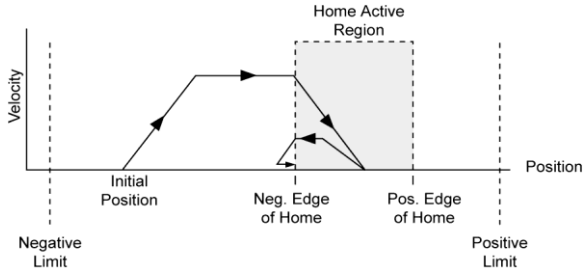
`JOG HOME X1`

Home Backup Enable (bit index 24) is set.

Home Negative Edge Select (bit index 25) is clear.

Home Negative Final Direction (bit index 26) is clear.

***Figure D***




Homing Profile Attributes:

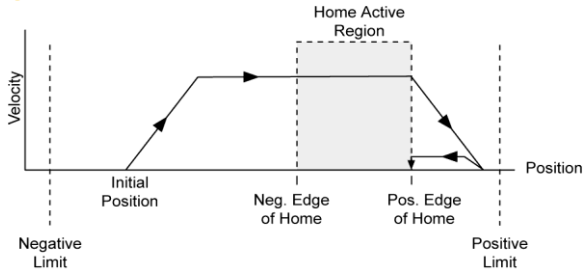
`JOG HOME X1`

Home Backup Enable (bit index 24) is set.

Home Negative Edge Select (bit index 25) is set.

Home Negative Final Direction (bit index 26) is clear.

***Figure E***




Homing Profile Attributes:

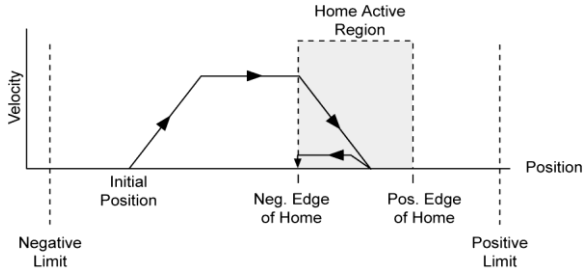
`JOG HOME X1`

Home Backup Enable (bit index 24) is set.

Home Negative Edge Select (bit index 25) is clear.

Home Negative Final Direction (bit index 26) is set.

***Figure F***




Homing Profile Attributes:

`JOG HOME X1`

Home Backup Enable (bit index 24) is set.

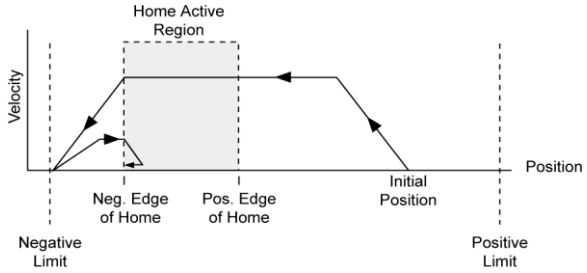
Home Negative Edge Select (bit index 25) is set.

Home Negative Final Direction (bit index 26) is set.

## Negative Homing (Homing Backup Enabled)

Figures G through J show the homing operation for different values of the Home Negative Edge Select and Home Negative Final Direction bits—the Home Backup Enable bit is set.

*Figure G*




Homing Profile Attributes:

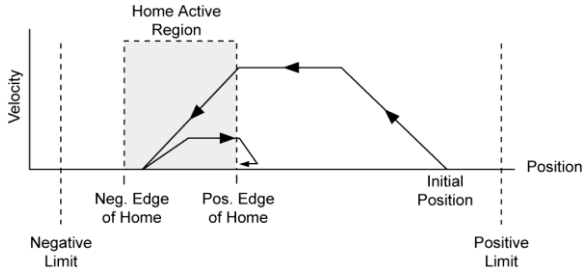
```
JOG HOME X-1
```

Home Backup Enable (bit index 24) is set.

Home Negative Edge Select (bit index 25) is set.

Home Negative Final Direction (bit index 26) is set.

*Figure H*




Homing Profile Attributes:

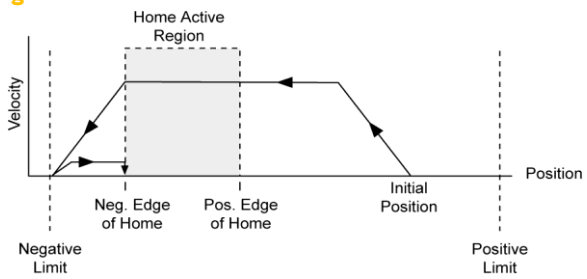
```
JOG HOME X-1
```

Home Backup Enable (bit index 24) is set.

Home Negative Edge Select (bit index 25) is clear.

Home Negative Final Direction (bit index 26) is set.

*Figure I*




Homing Profile Attributes:

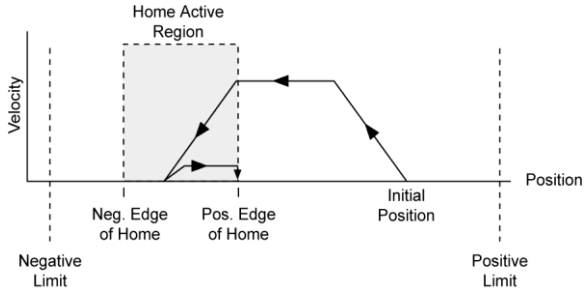
```
JOG HOME X-1
```

Home Backup Enable (bit index 24) is set.

Home Negative Edge Select (bit index 25) is set.

Home Negative Final Direction (bit index 26) is clear.

*Figure J*




Homing Profile Attributes:

```
JOG HOME X-1
```

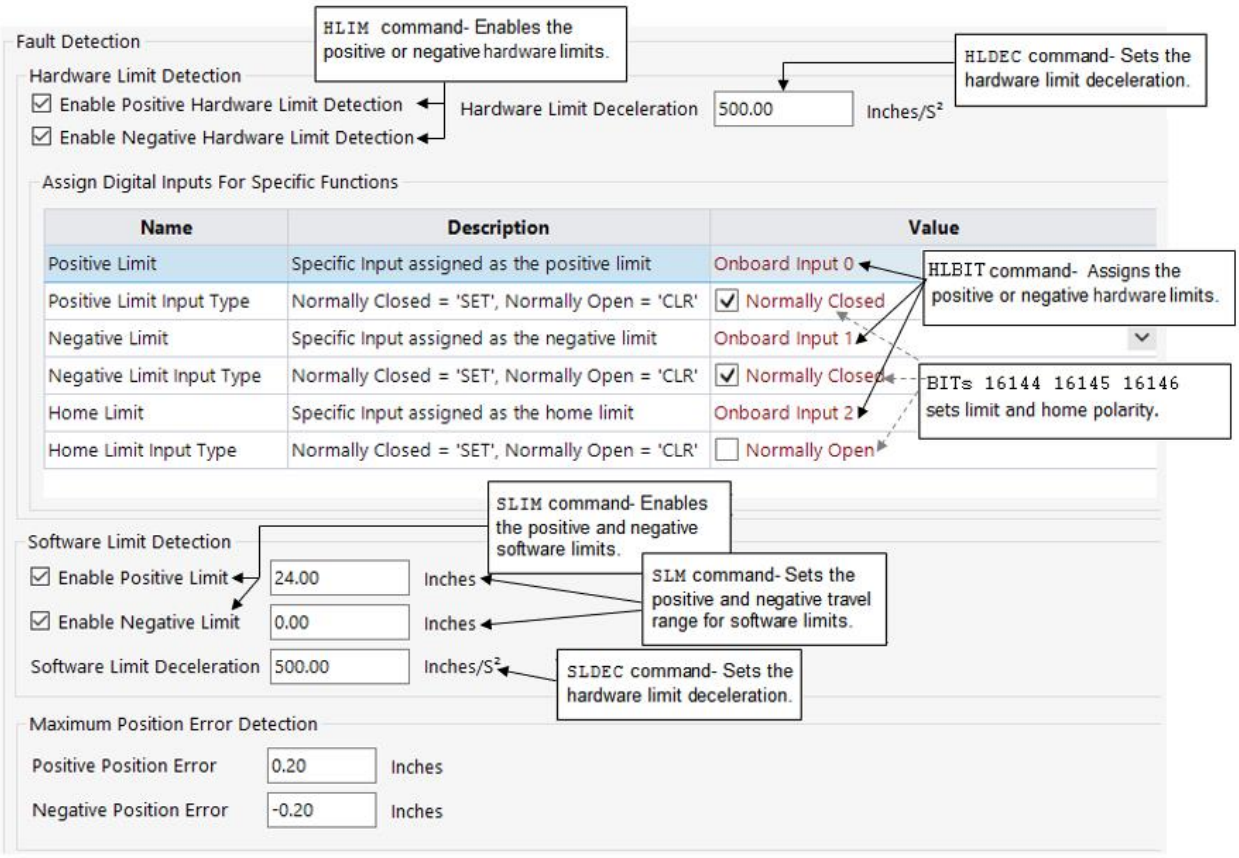
Home Backup Enable (bit index 24) is set.

Home Negative Edge Select (bit index 25) is clear.

Home Negative Final Direction (bit index 26) is clear.

# *Limit Detection*

The Configuration Wizard assists with setting up the Hardware and Software Limits Detection.




If limits are enabled, motion stops when the load encounters a limit. If the load hits a hardware limit, motion stops at the rate set by the `HLDEC`; if the load hits a software limit, motion stops at the rate set by the `SLDEC`.

## Dedicated I/O for Homing

For each axis, the user can assign which inputs are used for positive and negative hardware limits as well as the input used for homing. The inputs can be assigned or changed using the `HLBIT` command (no corresponding parameter exists). Use the `HLBIT` command to set the inputs for the positive hardware limit, negative hardware limit and homing sensor input.

```
HLBIT X (0, 1, 2)
```

For legacy systems or upgrading from ACR9000:

When using HLBIT without the parentheses, the number specifies the first input and the controller sets the next two contiguous inputs for the negative hardware limit and home limit.

```
HLBIT X0 : REM Input 0 is Pos. Limit, 1 is Neg. Limit and 2 is Home Limit.
```

For example, `HLBIT X0` assigns input 0 as the positive hardware limit and then the next two inputs. Input 1 becomes the negative hardware limit and input 2 becomes the home limit.

This syntax is still supported in ACR7000 and IPA firmware. However, you need to exercise caution with that syntax as the controller does not roll the assignment to the next block of 32 bits. For example, if `HLBIT X31` is issued, the negative hardware limit and homing input are not assigned and they become imaginary inputs with a value of zero.

# *Stopping Motion and Moves*

When an axis' KAMR is activated (by the user or automatically by the controller) the controller will:

- Attempt to stop the axis using the current setting for hardware limit deceleration, `HLDEC`. This is set within the Fault screen in the Configuration Wizard.
- Use the jog profiler to generate the setpoints necessary to bring the axis to a controlled stop. This may result in a Jog Offset. Use the `JOG RES` command to transfer the Jog Offset to the Current (coordinated) Position register. Or home the axis to re-establish the desired zero position.
- Stop jog, cam, gear or ballscrew motion on the axis by clearing those flags (gear activate, cam activate, jog active, jog forward and jog reverse).
- Set the Kill All Moves flag for the master that is assigned to that axis. This will stop and prevent any coordinated motion.
- Set the Kill All Motion Request flag for any other axes on that same master.

Any motion command issued while this flag is set will result in an error message "Associated Slave Kill Motion Request Active" in the Terminal Emulator. This is true if any axis assigned to the same master is commanded to move.

The user is responsible for clearing this flag.

Within a program, to resume motion, first clear the Kill All Motion Request flag for the axis (and any other axis on the same Master) and then clear the Kill All Moves flag in the master.

Enabling a drive using the `DRIVE ON` command will clear the Kill All Motion Request (KAMR) and Kill All Moves flag if the drive is not currently enabled.

Within the terminal emulator in PMM, the KAMR and Kill All Motion Request flags may be cleared for all axes by issuing a CTRL-Y.

The KAMR flag does not halt any programs. However, if the program encounters a new command to move while this bit is set, the program will halt. Non-motion programs can be running, monitoring motion program status for error recovery.

## Kill All Moves versus Kill All Motion Request

The Kill All Moves bits are for the interpolated motion moves. If you had an 3-axis X/Y/Z system, setting the Kill All Moves flag would immediately kill any `MOV` (single axis `X10` move, or interpolated `X5 Y/3` for example). Master 0 Kill All Moves bit 522 and Stop All Motion bit 523 would kill or stop all interpolated moves for the Master 0.

Setting the master's Kill All Moves or Stop All Motion bits will have no effect on other types of single axis moves like `JOG`, `GEAR` or `CAM`.

Setting the axis' Kill All Motion Request bit (bit 8467 for Axis 0) will kill all motion, including jogging for that axis and all other axes that are associated to that master.

***Example***

Axis 0 and Axis 1 are attached to Master 0—Axis 2 and Axis 3 are attached to Master 1. When all axes are jogging, setting bit 522 and bit 523 will have no effect on jogging. But when setting bit 8467, Kill All Motion Request for Axis 0, Axis 0 and Axis 1 will stop but Axis 2 and Axis 3 will continue to jog.

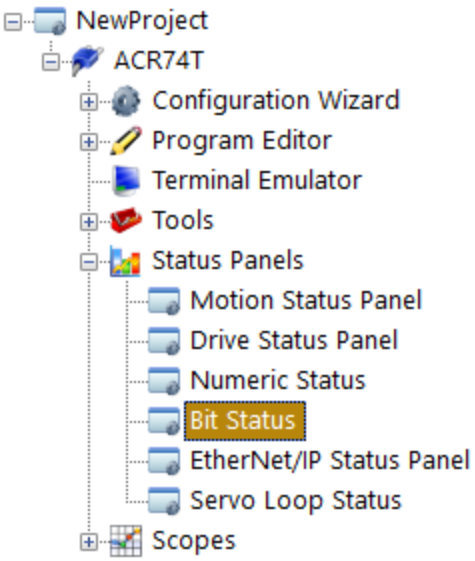
## Flag Comparison

The following table shows the bit numbers for Kill All Motion Request axis flags and the bit numbers for Kill All Moves master flags.

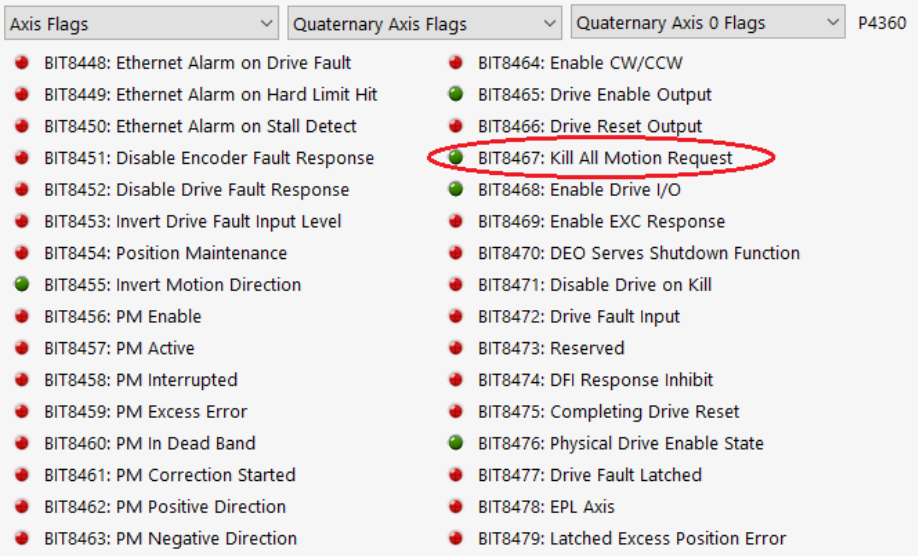
| Kill All Motion Request | | | | | | | |
|---|---|---|---|---|---|---|---|
| Quaternary Axis Flags | | | | | | | |
| Axis Number | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8467 | 8499 | 8531 | 8563 | 8595 | 8627 | 8659 | 8691 |
| Axis Number | | | | | | | |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 8723 | 8755 | 8787 | 8819 | 8851 | 8883 | 8915 | 8947 |
| | | | | | | | |
| Kill All Moves | | | | | | | |
| Master Flags | | | | | | | |
| Master Number | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 522 | 554 | 586 | 618 | 650 | 682 | 714 | 746 |
| Master Number | | | | | | | |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 7434 | 7466 | 7498 | 7530 | 7562 | 7594 | 7626 | 7658 |

## Bit Status Window Comparison

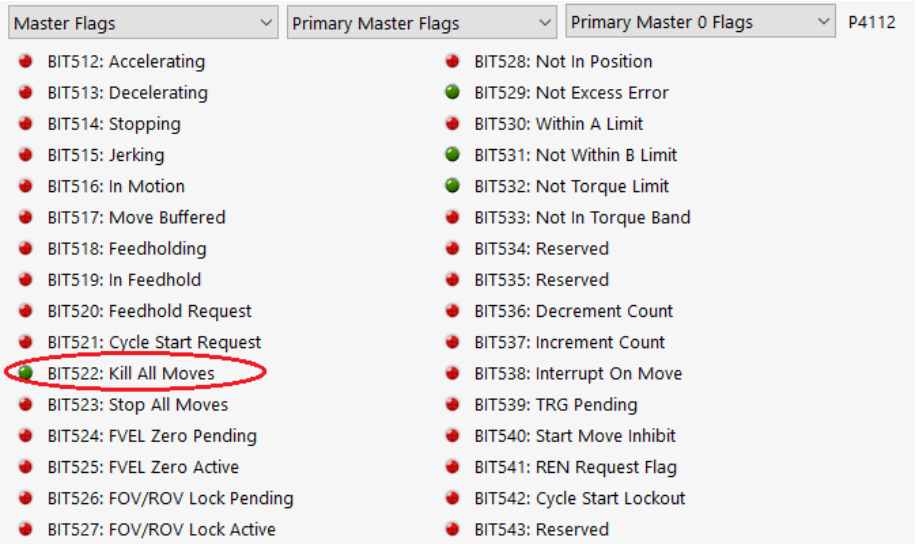
Locate the Bit Status Panel by clicking on the plus sign (+) next to Status Panels on the Explorer in PMM and clicking on *Bit Status*.




Select *Axis Flags* in the first pull-down menu, *Quaternary Axis Flags* in the second pull-down menu and *Quaternary Axis 0 Flags* in the third pull-down menu to display the Kill All Motion Request bit for Axis 0. A green LED, as circled in red below, indicates that the flag is set. All motion is stopped for this axis and all other axes on the same master.




Select *Master Flags* in the first pull-down menu, *Primary Master Flags* in the second pull-down menu and *Primary Master 0 Flags* in the third pull-down menu to display the Kill All Moves Request bit for Master 0. A green LED, as circled in red below, indicates that the flag is set.




***Example:***

This example uses terminal commands.

```
P00>ATTACH
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
```

The `ATTACH` command will reply with information about which axes are part of the master group.

```
P00>JOG FWD X
```

`JOG FWD X` starts a continuous jog move on X axis.

```
P00>SET 8467
```

`SET 8467` sets the KAMR for the X axis. It would decelerate at the `HLDEC` rate.

```
P00>JOG FWD Y
P00>Associated Slave Kill Motion Request is active
```

Y-axis motion is prevented due to the X-axis KAMR flag being active.

```
P00>CLR 8467 CLR8499
P00>JOG FWD Y
```

Y-axis motion is now allowed.

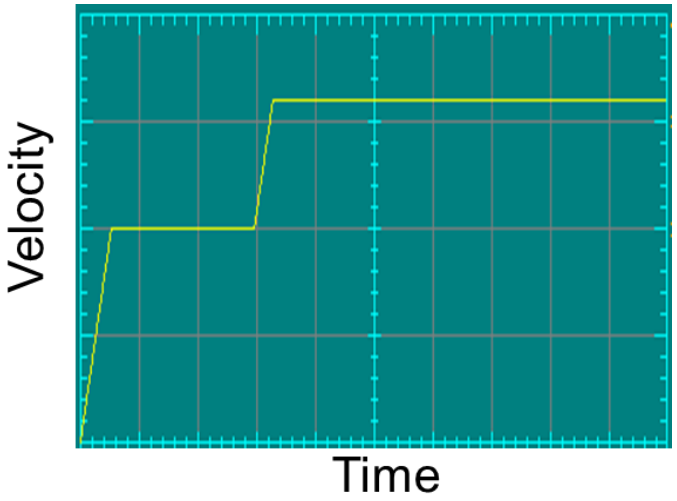
**NOTE: Enabling drives using the DRIVE ON command will clear the Kill All Motion Request (KAMR) and Kill All Moves bits if the drive is not currently enabled.**

# Contoured (Tiered) Profiles

Changes to jog velocity take effect immediately (velocity moves `JOG FWD` or `JOG REV`).

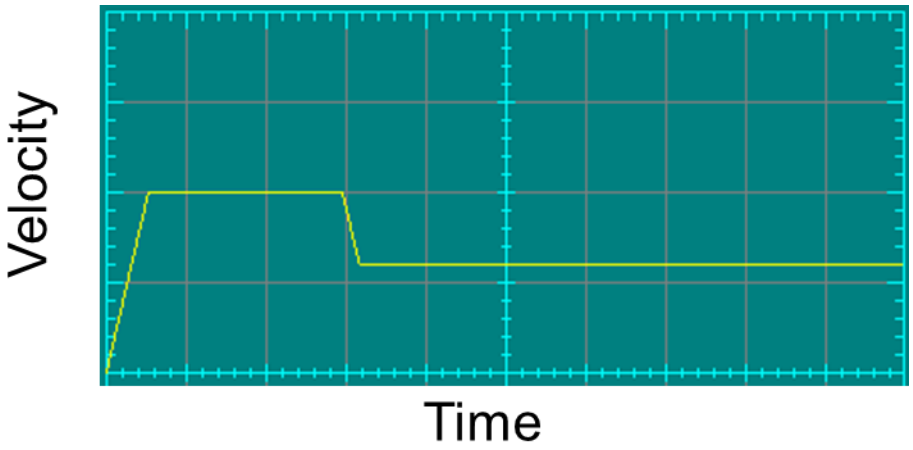
***Terminal Emulator Sample:***

```
DRIVE ON X
JOG VEL X5
JOG FWD X
JOG VEL X8
JOG OFF X
```




Or decelerating:

```
JOG VEL X5
JOG FWD X
JOG VEL X3
JOG OFF X
```



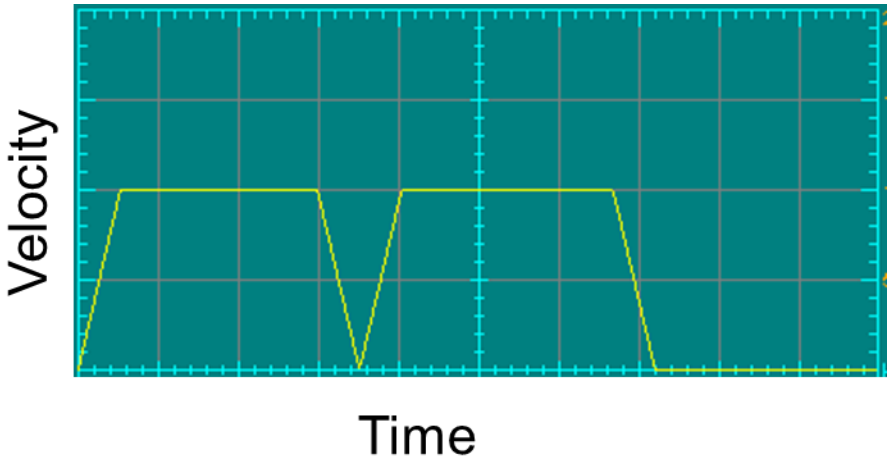
If a Jog move is in progress, another Jog move command (`JOG INC` or `JOG ABS`) will cause the current move to abort and ramp to zero velocity before starting the next move.

***Terminal Emulator Sample:***

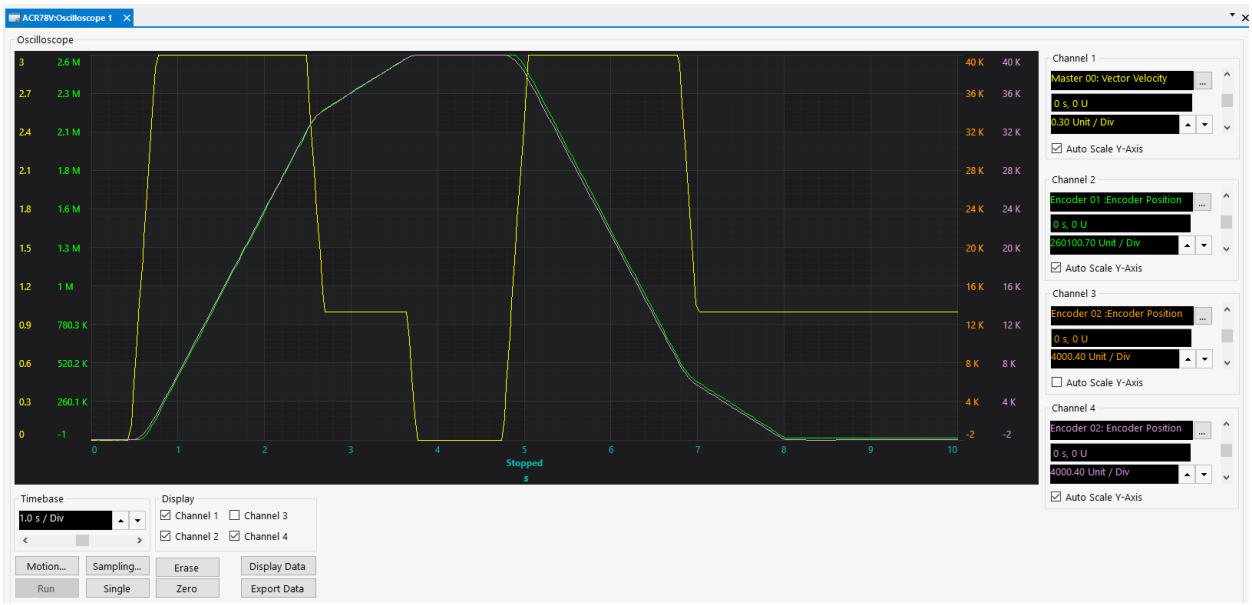
```
DRIVE ON X
JOG VEL X5
JOG INC X20
JOG INC X3
```

Or:

```
DRIVE ON X
JOG FWD X
JOG INC X3
```



# Blended (Tiered) Interpolated Moves




With an interpolated move, it would be programmed as two moves but with the stop ramp `STP` set to 0 so it would start the second move after completing the first move.

***Example:***

```
ACC 10 DEC 10 STP 0 VEL 3
Y/4 Z/4      : REM Start incremental move for Y and Z at speed of 3.
VEL 1 STP 10
Y/1 Z/1      : REM Toe-in with another move for Y Z at speed of 1.
INH-516      : REM Wait until move stops.
DWL 1        : REM Wait 1 second.
VEL 3 STP0
Y1 Z1        : REM Return move same but in absolute values.
VEL 1 STP10
Y0 Z0
```

# High-speed Position Capture (INTCAP)

`INTCAP` allows you to capture an axis position when one of the controller's high-speed trigger inputs or the encoder reference (Z channel) turns on. The position is stored in the capture register. The ACR7000 stepper has four capture registers, one for each stepper axis. The IPA has two, one for its servo axis and another for its auxiliary encoder input (ENC 1). The ACR7000 servo has one for each servo axis. The ACR7000 controller has one for each axis and any of the first four can also be used for the auxiliary encoder input (ENC 8).

PMM's online help for `INTCAP` has charts for the different ACR controllers (ACR7xT stepper, IPA single axis servo, ACR7xV servo and ACR7xC controller). Capture modes marked with (+) capture a rising edge while (-) is a falling edge. *INP* is a trigger input and *Z ENC* is the encoder reference mark. These are necessary as they arm the specific hardware at the chip level to capture the encoder position very precisely (1 µs latency). Multiple captures can be armed at the same time. No motion is initiated by `INTCAP`—it is simply a mechanism to arm the

capture to take place when the source is triggered. `INTCAP` is also used in other AcroBASIC commands such as `HSINT` (high speed interruptible move) and `MSEEK` (marker seek).

### *ACR7xV Capture Modes*

| | ACR7000 Servo (ACR74V, ACR78V) | | | |
|---|---|---|---|---|
| *SRC* | ENC0 | ENC1 | ENC2 | ENC3 |
| *capture_ register* | CAP0 | CAP1 | CAP2 | CAP3 |
| *Mode* | | | | |
| **0** | +Z-ENC 0 | +Z-ENC 1 | +Z-ENC 2 | +Z-ENC 3 |
| **1** | | | | |
| **2** | +INP 24 | +INP 26 | +INP 28 | +INP 30 |
| **3** | +INP 25 | +INP 27 | +INP 29 | +INP 31 |
| **4** | -Z-ENC 0 | -Z-ENC 1 | -Z-ENC 2 | -Z-ENC 3 |
| **5** | | | | |
| **6** | -INP 24 | -INP 26 | -INP 28 | -INP 30 |
| **7** | -INP 25 | -INP 27 | -INP 29 | -INP 31 |

Note for the stepper axis, PMM already attaches the encoder whether selected or not and the correct settings will be applied to the ACR7000 stepper controller. If there is no encoder (open loop stepper), the current position register value is used.

A Capture Complete Flag indicates when the capture is complete and then the Hardware Capture register will have the position information:

| Axis | Capture Complete Flag | Hardware Capture Position |
|---|---|---|
| 0 | 777 | 12292 |
| 1 | 809 | 12548 |
| 2 | 841 | 12804 |
| 3 | 873 | 13060 |
| 4 | 905 | 13316 |
| 5 | 937 | 13572 |
| 6 | 969 | 13828 |
| 7 | 1001 | 14084 |

### *Example Encoder Reference Trigger*

```
REM Y axis use capture register 1, Mode0, rising edge of Encoder1
REM reference marker.
INTCAP Y0    : REM arms capture for Y Axis
JOG FWD Y    : REM initiate jogging move on X axis
INH 809      : REM wait until Axis1 flag "Capture Complete" is set
JOG OFF Y    : REM stop jogging
REM capture parameter was not specified in INTCAP command, defaults to
REM Axis1
```

```
PRINT P12548       : REM Print Axis1 hardware Capture position
RETURN
```

***ACR7xC Example Capture—Two Axis Positions with One Trigger Input***

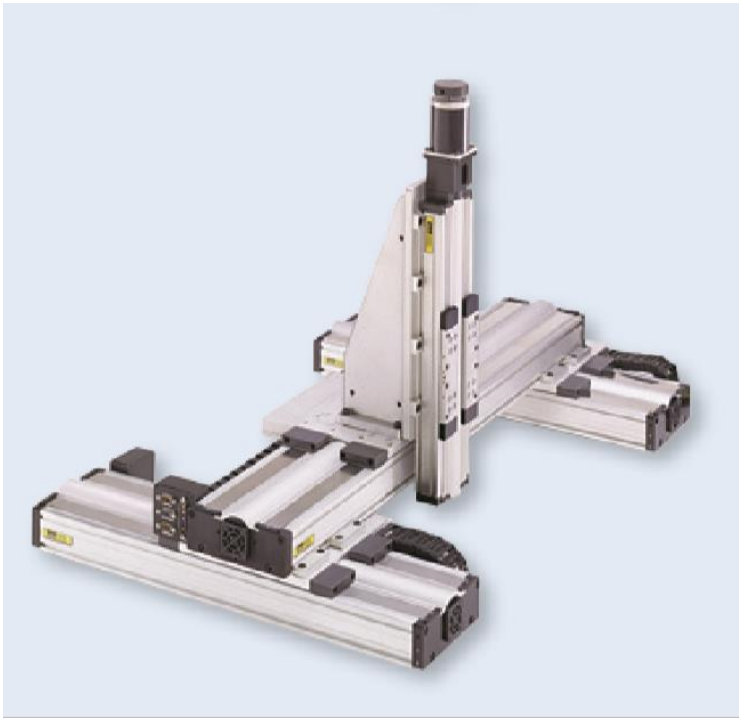
```
AXIS0 INTCAP 10 CAP2 P12804
REM Mode10, CAP2  : Rising 3rd External, CAP2 uses Input24
AXIS1 INTCAP 11 CAP3 P13060
REM Mode11, CAP3  : Rising 4th External, CAP3 uses Input24
AXIS0 JOG FWD     : AXIS1 JOG FWD
INH 841           : INH 873
PRINT "Axis0 Capture Position", P12804
PRINT "Axis1 Capture Position", P13060
```

Instead of a hardware capture, software capture is available with SET 113. It captures all encoder positions at the next period for the controller and stores them within the software capture parameters (P12293 for Axis 0 and so on).

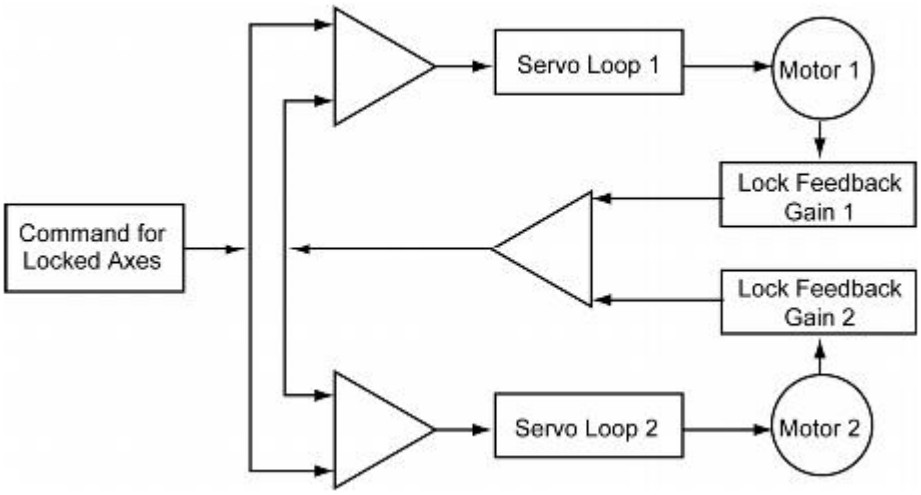
# Lock

The LOCK command redirects one axis to follow the primary setpoint of a second axis. This can be used to have multiple axes receive the same setpoint in the same servo cycle rather than following another axis (one servo period behind).

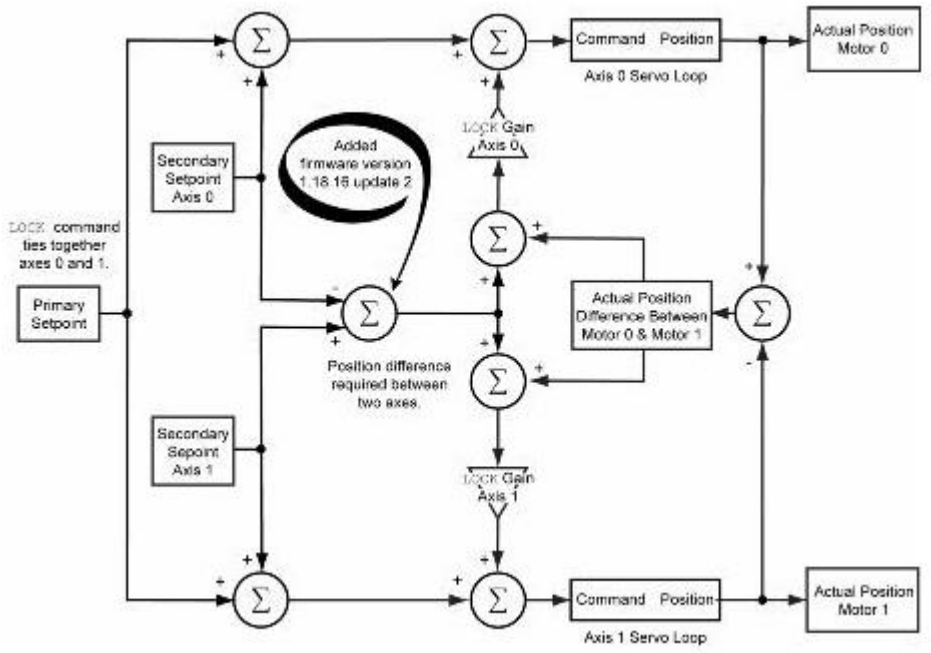
LOCK is essential for controlling a gantry system where two mechanical systems need to be coupled. Once the two axes are locked, a special control loop will minimize the error between them assuring perfect coordination. This is important for an XX' (X/X prime) so that one axis is not fighting the other, leading to crabbing or having to detune one of the axes.



When two axes are locked together using the LOCK command, their primary setpoints become the same. In other words, the two axes will get exactly the same command signal. However, in the real world, the response of the two physical motors/actuators will be slightly different. To compensate for this error, the user can turn on a feedback loop by setting some gain values for the "Lock Feed Back Gain" parameters of the locked axes. Thus, if one axis has a disturbance and corrects for the position error, the locked axes will also correct. The default value is zero, which turns this feedback loop off.




With feedback gain:




***Example***

```
P12376 = 3.5       : REM Set lock gain axis 0.
P12632 = 3.5       : REM Set lock gain axis 1.
LOCK Y X           : REM Lock axis Y to axis X's primary setpoint.
```

```
X/20                : REM Start motion axis X, axis Y also moves due to lock.
UNLOCK Y            : REM Unlock axis Y.
```

When the UNLOCK command is issued, that axis' position will be 0 and will need to be reset. The difference between the two positions should be stored and the unlocked axis should be reset to the main axis position less the offset.

# *Rotary Axis*

The ROTARY command allows a rotary axis to take the shortest path to a position, whether for a precision rotary stage, standalone motor or motor with gearhead. ROTARY sets the rotary axis length used for the shortest-distance calculations.

If the rotary length of an axis is non-zero, a MOD function is performed on absolute moves and the result is run through a shortest-distance calculation. The resulting move will never be longer than half the rotary axis length. Incremental moves are not affected by the rotary axis length.

This command only affects MOV absolute moves. JOG moves are not affected. Before enabling the ROTARY command, it may be useful clear the JOG offset register by issuing the JOG RES command, transferring the Jog Offset to the Coordinated Position register.

The NORM command can be used to return the current position to within the bounds of the rotary length. Issuing a ROTARY command without an argument will display the current setting. To disable, set ROTARY length to 0.

To increase the accuracy of the rotary motion, use degrees for the units. This can be selected within the Configuration Wizard.

***Example***

The following example sets the rotary length of the A axis to 360 units:

```
ROTARY A360
A120  : REM Move to 120 units results in positive motion.
A0    : REM Go back to 0 position.
A275
REM Move to 275 units results in negative motion as this is the
```

REM shortest distance.

# External Time Base

By default, motion's time base is set to the servo clock. The SRC command can be used to change to an external timebase, such as an encoder or parameter. This is done with the SRC (source) command. This is similar to CAM SRC or GEAR SRC, but those are only for CAM motion or GEAR motion. During each servo interrupt, the change in source pulses is multiplied by the servo period and the resulting delta time is fed into the velocity profile mechanism. Redirecting the source allows the controller to use an external time base for coordinated motion. Note when using P parameter, do not use a source that could be changed abruptly or have discontinuities. The encoder inputs would be good choices, either with ENC1 syntax or the corresponding P parameter P6272 for ENC8. Ratchets are also available as a source.

***Example***

```
SRC P6272
```

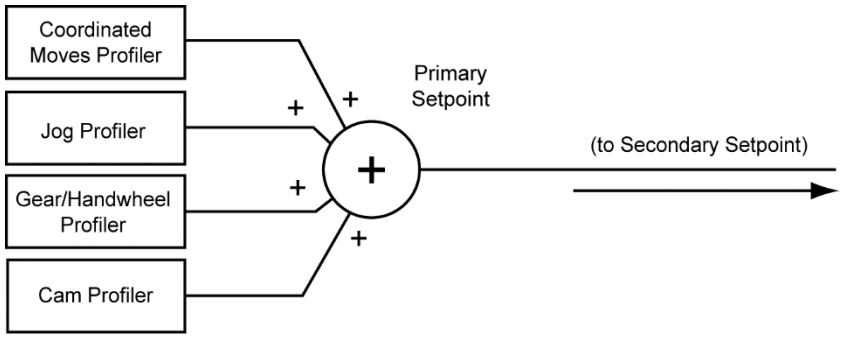
# Servo Loop Fundamentals

Each of the profilers contains a register with a value of the current offset. These values are added together and the summation is called the Primary Setpoint (PSP).

PSP = Coordinated Moves + Jog + Gear + Cam

See below for a diagram of the Primary Setpoint summation.

***Primary Setpoint Summation***




# Setpoint Compensation

There are two mechanical characteristics that the controller takes into consideration and compensates for: hysteresis losses and non-linear position error, which are processed by the Backlash Generator and Ballscrew Profiler, respectively.

Backlash Generator: Used to compensate for error introduced by hysteresis in mechanical gearboxes. Backlash is used in the Secondary Setpoint summation if the Primary Setpoint value is positive. Use the BKL command (set

backlash compensation) to set the compensation, or, without an argument, to display the current setting for an axis.

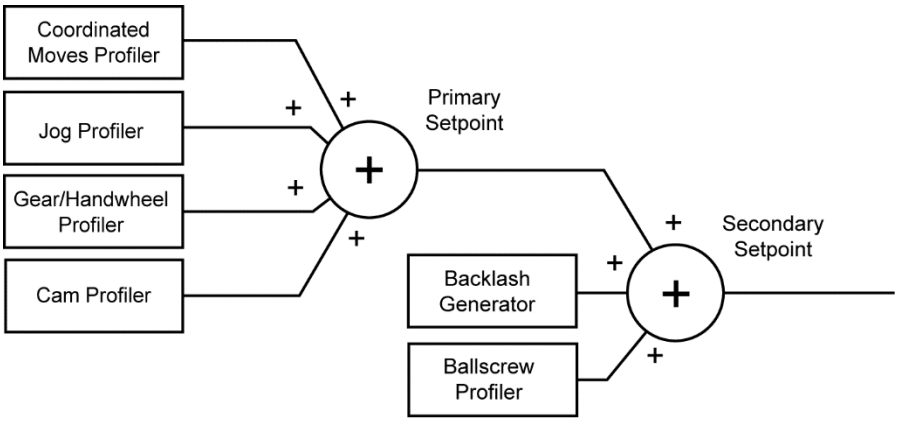
Ballscrew Profiler: Used to compensate for non-linear position error introduced by mechanical ballscrews and gearboxes. Use the `BSC` command (ballscrew compensation) to initialize and control ballscrew compensation for an axis.

The values of the Backlash Generator and Ballscrew Profiler are added to the Primary Setpoint, and this summation is called the Secondary Setpoint (SSP).

SSP = PSP + Backlash + Ballscrew

The information up to and including the SSP is the commanded position. See the figure below.

***Secondary Setpoint Summation***




# *Viewing the Setpoint Calculations*

Servo loop calculations for the actual position of an axis can be observed in PMM. The Servo Loop Status panel shows the motion offsets, primary and secondary setpoints, servo gains and other values, and how they result in the final position output.

In the Explorer, click *Status Panels*, then click *Servo Loop Status*.

Note that PMM's display will be slow due to the communications. The update of the servo loops is the `PERIOD` of the controller (see `PERIOD` in ACR Command Reference or PMM's online help file for further details).

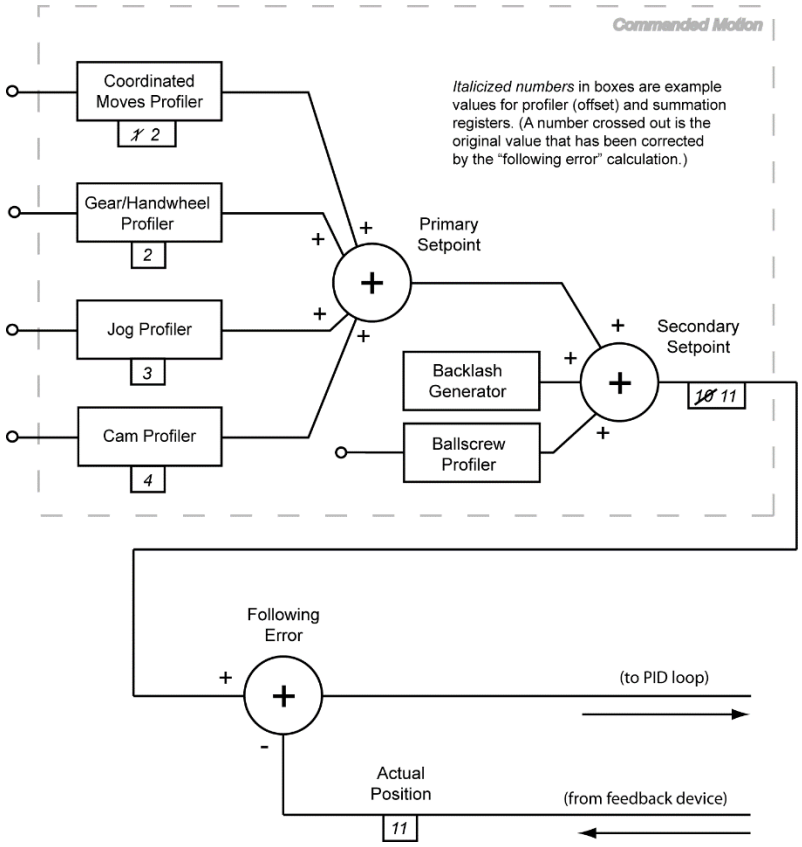
## Following Error

The Secondary Setpoint is compared with the value of the Actual Position received from a feedback device. See the figure below. The difference between the Secondary Setpoint and Actual Position is called the Following Error:

Following Error = Secondary Setpoint - Actual Position

The controller makes adjustments to the motor position through a constant cycle of comparison and correction. Following Error is used by the PID loop (servo control algorithm) to keep the Actual Position equal (or approaching equal) to the Secondary Setpoint.

### *Following Error*




# *Ballscrew Compensation*

Ballscrew compensation is primarily used to compensate for nonlinear position error introduced by mechanical ballscrews and linear encoders. Ballscrew commands are identical to cam commands. Both ballscrews and cams can be active at the same time, each with different settings and offset tables.

The main difference between ballscrew and electronic cam is that the default source for a ballscrew points to the primary setpoint, therefore the `BSC SRC` command is normally not required. The primary setpoint is used so that the ballscrew offset is not fed into the calculation of the ballscrew index, causing an unstable condition.

**NOTE: The primary setpoint is the summation of the coordinated position and the total cam, gear and jog offsets. The secondary setpoint is the summation of the primary setpoint and the total ballscrew and backlash offsets. The secondary setpoint is the one that is actually used by the servo loop.**

## BSC with PPU

When `PPU` is set for an axis you must use a `BSC SCALE` equal to `1/PPU` and enter all values in pulses.

```
PPU X 1000 : REM 1 micron linear encoder, user units 1 mm, BSC SCALE X 0.001.
```

All entries in the long array used to designate a BSC segment MUST be made in encoder pulses.

```
LA0(0)=100 : REM Array entry in encoder pulses, 100 micron.
```

When PPU is specified for the axis that is used as the ballscrew, axis segment lengths must still be entered in encoder pulses.

```
BSC SEG X(0,100000,LA0) : REM Master encoder pulses 100000 microns or 100 mm.
```

## Encoder Accuracy

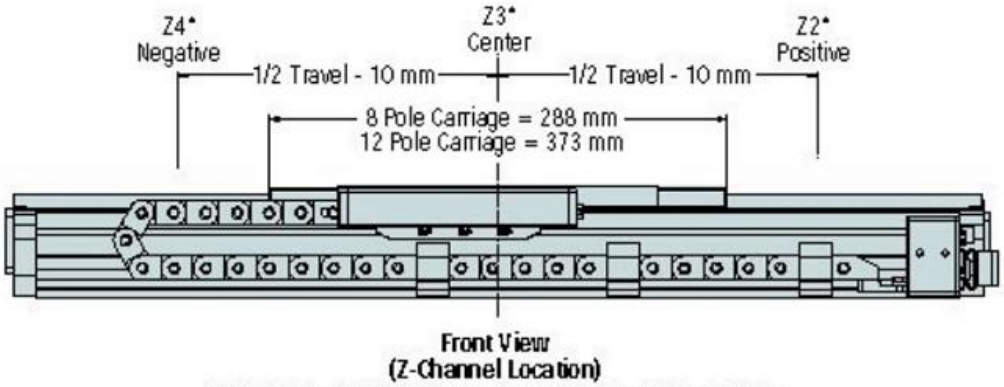
The 406LXR Series makes use of an optical linear encoder for positional feedback. This device consists of a readhead, which is connected to the carriage, and a steel tape scale, which is mounted inside the base of the 406LXR. The linearity of this scale is ±3 microns per meter, however the absolute accuracy can be many times larger. To compensate for this error, an error plot of each 400LXR is done at the factory using a laser interferometer. From this plot a linear slope correction factor is calculated (Figure 2). Then a second error plot is run using the slope correction factor. These tests are conducted with the Point of Measurement (POM) in the center of the carriage 35 mm above the carriage surface.

## Slope Correction

Slope correction is simply removing the linear error of the table. The graphs below show an example of a non-slope corrected error (Figure 2) plot and the same plot with slope correction (Figure 3). As can be seen, the absolute accuracy has been greatly improved. The slope factor is marked on each unit. It is the slope of the line in microns per meter. This factor may be positive or negative, depending on the direction of the error.

If the application requires absolute accuracy, the slope factor must be incorporated into the motion program. This is a matter of either assigning variables for motion positions and using the slope correction in the variable equation or, for ACR series controllers, using the ballscrew compensation feature, which simplifies error correction. Accuracy can be improved even more by using the actual data points and incorporating these into a compensation array used by the BSC command (Figure 4).

**NOTE: The zero position (or starting point) of the error plots is at the extreme NEGATIVE end of travel.**



**Sample Data from LXR Error Report**

| | |
|---|---|
| Axis | 406T12LXRMP |
| Location | |
| Title | |
| Increment Size | 25 |
| Total Travel | 1450 |
| Slope (um/mm) | (0.034) |
| Slope (um/meter) | (34.31) |
| Accuracy (um) | 61.7 |
| Bi-Dir Repeat(um) | 2.2 |
| Corrected Acc.(um) | 16.58 |

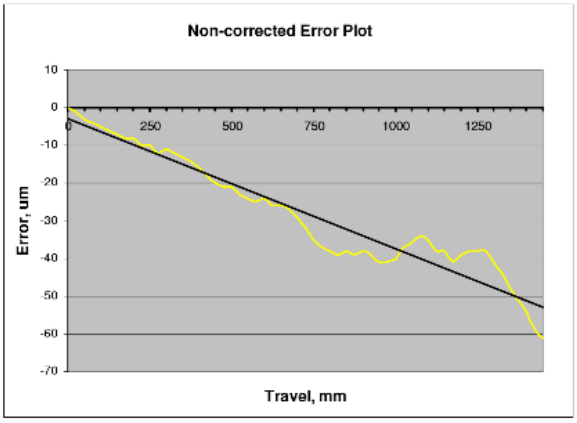



Figure 2
Actual error plot before correction

Slope value =-34.3

### *BSC Using Slope Correction Value*

Slope value 34.3 µm/m. Value at 1450 mm: 34.3 [µm/m] * 1.45 [m] = 49.6 [µm]

```
DIM LA(1)    : REM Dimensions one long array for correction values.
DIM LA0(2)   : REM Dimension array zero with 2 data points.
LA0(0)=0     : REM Set first array value (negative end of travel) to zero.
LA0(1)=50    : REM Set last array value to inverse of slope correction value.
BSC DIM X1   : REM Dimension one segment for correction values.
BSC SEG X (0, 1450000, LA0)    : REM Segment 0 is 1450000 microns(1450 mm).
BSC SCALE X 0.001              : REM Scale = 1/PPU.
BSC ON X                       : REM Activate ballscrew compensation.
```

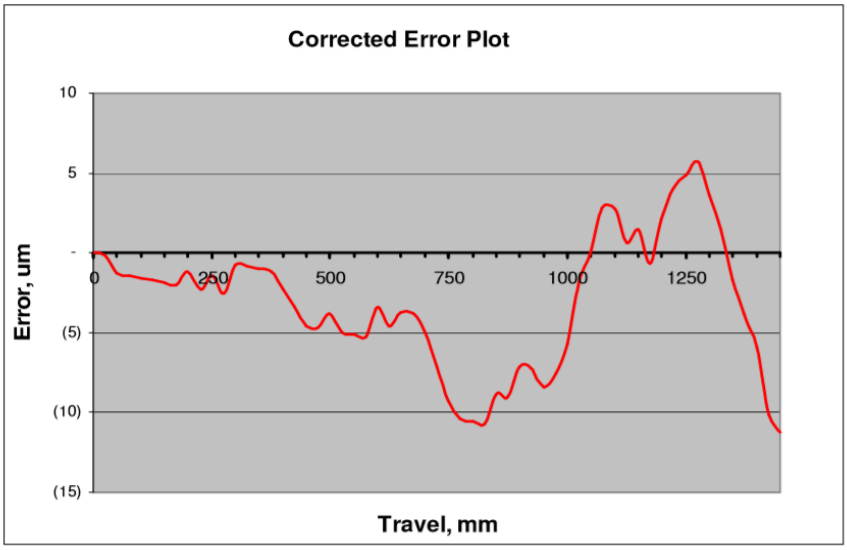



Figure 3
Corrected error plot using slope correction value

### *BSC Using Error Data Points From Laser Report*

```
DIM LA(1)    : REM Dimensions one long array for correction values.
DIM LA0(59)  : REM Dimension array 0 with 59 data points.
LA0(0)=0
LA0(1)=1
LA0(2)=3
LA0(3)=4
```

```
LA0(4)=5
LA0(5)=6
LA0(6)=7
LA0(7)=8
LA0(8)=8
LA0(9)=10
LA0(10)=10
LA0(11)=12
LA0(12)=11
LA0(13)=12
LA0(14)=13
LA0(15)=14
LA0(16)=16
LA0(17)=18
LA0(18)=20
LA0(19)=21
LA0(20)=21
LA0(21)=23
LA0(22)=24
LA0(23)=25
LA0(24)=24
LA0(25)=26
LA0(26)=26
LA0(27)=27
LA0(28)=29
LA0(29)=32
LA0(30)=35
LA0(31)=37
LA0(32)=38
LA0(33)=39
LA0(34)=38
LA0(35)=39
LA0(36)=38
LA0(37)=39
LA0(38)=41
LA0(39)=41
LA0(40)=40
LA0(41)=37
LA0(42)=36
LA0(43)=34
LA0(44)=35
LA0(45)=38
LA0(46)=38
LA0(47)=41
LA0(48)=39
LA0(49)=38
LA0(50)=38
LA0(51)=38
LA0(52)=41
```

```
LA0(53)=44
LA0(54)=48
LA0(55)=51
LA0(56)=54
LA0(57)=59
LA0(58)=61
BSC DIM X1              : REM Dimension one segment for correction values.
BSC SEG X (0, 1450000, LA0)    : REM Segment 0 is 1450000 microns(1450 mm).
BSC SCALE X 0.001              : REM Scale = 1/PPU.
BSC ON X                       : REM Activate ballscrew compensation.
```

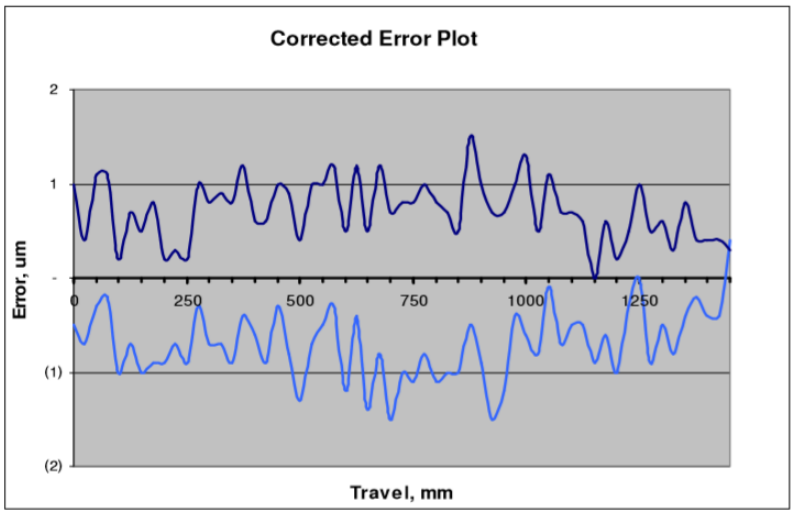



Figure 4

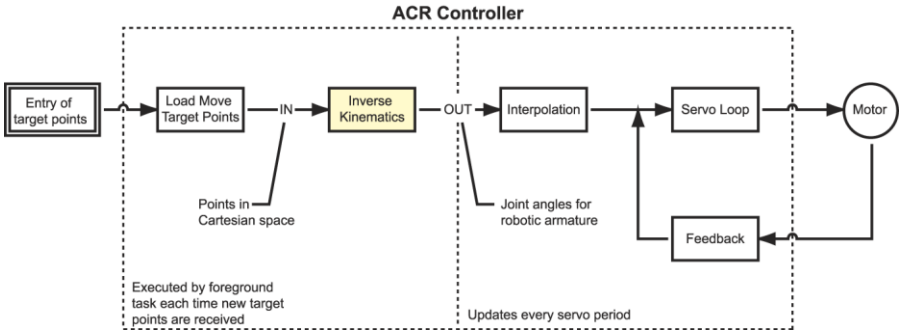
Corrected error plot using laser table compensation points.

# Inverse Kinematics

Kinematics is a branch of mechanics that provides a mathematical means of describing motion. Inverse kinematics looks at a position and works backwards to determine the motions necessary to obtain that position.

Robotic applications frequently use inverse kinematics. Algorithms describe the mechanical system and translate the rotational motion of robotics into Cartesian coordinates. Consequently, an end user provides simple Cartesian coordinates for an application and the inverse kinematics calculates necessary movements to reach that position.

Suppose an application has a cutting tool at the end of a four-axis robotic arm and an HMI. The controller, using algorithms developed by the application builder, transforms the motion target points from Cartesian coordinates to rotational coordinates to position the arm joints and cutting tool. Once transformed, the controller interpolates the target points to generate a motion path. See the illustration below:




# *Programming the Inverse Kinematics*

Each application is different. The algorithm for your application can consist of equations, logical expressions and commands in the AcroBASIC language. You can do the following:

- Store algorithms in any of the programs 0 through 14 (be sure to dimension memory for the program).
- Save the program to Flash memory.
- Use the `PASSWORD` command to protect the program from uploading or listing.
- Include the `INVK` commands in a program, or in the setup before a program.

***Example***

The following program results in a circle instead of a straight line because of the transformation described in program 7 (PROG7).

```
PROG7
PROGRAM
P12361= SIN(P12360)      : REM Describe transformation in PROG7.
P12617= COS(P12360)      : REM Describe transformation in PROG7.
ENDP
PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
```

```
ATTACH SLAVE1 AXIS1 "Y"
PPU X 2000 Y 2000       : REM Scale commands to engineering units
ACC 100 DEC 100 STP 0 VEL 0
INVK PROG7              : REM Tell MASTER0 where the transformations are.
INVK ON                 : REM Turn on the Kinematics.
PROGRAM
_start
X / 0.2                 : REM Incremental move in Cartesian space.
GOTO start
ENDP
RUN PROG0               : REM Run the program.
```

Note the following limitations with the ACR's inverse kinematics feature:

- It only applies to master moves, such as `X4 Y/-8`. Jogging, gearing and camming are unaffected.
- Only the end point of the move is modified—it is not guaranteed that the system will move along a desired path.

The inverse kinematics feature is best suited to testing and prototyping.

# CHAPTER 4
# Writing AcroBASIC Programs

# Writing AcroBasic Programs

AcroBASIC programming is text-based and top down. When writing programs, use subroutines from a main routine. This makes it easier to add, read and test new sections of code rather than having to troubleshoot a large multi-page program. Thus, removing a subroutine is easy by removing the `GOSUB` (or commenting it out by placing a ‘ single apostrophe at the beginning of the line) and re-downloading.

Test your code as the program is developed. Use *Save As* to backup copies of the code before changes are made so you always have a starting point to which you can go back. You are your own revision control.

Below is a sample structure we recommend using—note the Application Examples also use this structure.

Comments are helpful. If you need to revisit your code after some time, they can help describe what sections of code are doing.

The error recovery program would be in another program, such as Program 1. This would spend its time monitoring whether the motion program, Program 0, has stopped running, and handle the errors, reset the drive and restart Program 0.

Parameters and bits are global. This makes it easier to use them across programs. However, be careful if two programs can write and change the value of the same parameter or bit. If one program is using its value and the other program changes the value, this can cause problems. If the first program needs to complete its sequence before the value changes, consider using a local variable as a copy of the parameter or use a user bit to inhibit changing the value until the first program is complete.

Non-motion programs, Program 8 through Program 14, share a 1 ms time slice. An inhibit or dwell in one program will affect all the non-motion programs.

```
PROGRAM
PBOOT                   : REM Assigns program to run automatically.
REM Description of program
GOSUB SETUP             : REM SETUP does this part.

_MAIN
REM This calls 3 subroutines sequentially.
GOSUB SubroutineA       : REM SubroutineA does this.
GOSUB SubroutineB       : REM SubroutineB does this.
GOSUB SubroutineC       : REM SubroutineC does this.
GOTO MAIN               : REM This goes to MAIN looping continuously.
END                     : REM Ends the program.

_SETUP
REM ONE TIME SETUP
<insert AcroBasic code>
RETURN

_SubroutineA
<insert AcroBasic code>
RETURN
```

```
_SubroutineB
<insert AcroBasic code>
RETURN

_SubroutineC
<insert AcroBasic code>
RETURN

ENDP
```

# *Application Examples*

The sample programs in this section provides more in-depth program examples of the following topics:

- Sample Motion Program
- Enable Drives Subroutine
- Absolute Interpolated Motion Subroutine
- Incremental Interpolated Motion Subroutine
- Basic Absolute and Incremental Motion Subroutine
- Absolute Jog Moves Subroutine
- Incremental Jog Moves Subroutine
- Absolute and Incremental Jog Moves Subroutine
- Homing Subroutine
- Advanced Homing
- Homing for XYZ System
- Open Sample
- Teach Array
- Programmable Limit Switch
- EIP Scanner - Wago 750
- Joystick
- Capture Data
- Peer-to-Peer
- ACR7xT Status
- ACR7xT Home to Hard Stop
- Time Subroutine
- Add-On Instructions (AOIs) for IPA
- Xpress HMI with ACR7000
- Xpress HMI with IPA

Note that these samples and others are available for download and use from Parker Community Knowledge Base, also linked from ACR7000 product page.

## Sample Motion Program

Sample two-axis motion program with main program using subroutines for enable, home, interpolated motion and jog moves with full comments. The subroutines are highlighted with headings. The program finishes after the homing subroutine.

Similar samples for one-axis, three-axis and four-axis systems are available on the Knowledge Base.

```
PROGRAM
PBOOT
REM PBOOT assigns the program automatically on powerup or reboot.
REM PBOOT has to be the first command within the program.
REM This is a sample program showing enabling, homing and two types
REM of moves (MOV and JOG).
REM In terminal window, go to Prog0 prompt after downloading and type LRUN
REM to run and view PRINT (?) statements.
REM After running, you can view axis status and program line numbers
REM incrementing under Status Panels > Motion Status Panel.
REM This assumes default assignment that Axis0 is X and Axis1 is Y and
REM are attached to Prog0.
REM The X and Y are axis aliases and only recognized within Prog0, so use the
REM below sample code in Prog0.
REM This sets default values for MOV (default interpolated moves)
ACC 10
DEC 10
STP 10
VEL 1
REM This sets default values for JOG (single axis offset moves)
JOG ACC X 10
JOG DEC X 10
JOG VEL X 1
JOG ACC Y 10
JOG DEC Y 10
JOG VEL Y 2
GOSUB ENABLEDRIVE : REM GO TO SUBROUTINE "ENABLEDRIVE"
REM This will then go to _ENABLEDRIVE and run that subroutine until
REM the return and then come back to this point.
GOSUB HOMING : REM GO TO SUBROUTINE HOMING
REM This will then go to _HOMING and run that subroutine until the return
REM and then come back to this point.
REM Presumes limits/homes assigned per Configuration Wizard and wired.
REM If not used, change to 'GOSUB HOMING to comment this line out and
REM not execute that subroutine.
_MAIN
' This is a label used with GOTO MAIN below to run continuously.
' Comments can be made with ' on its own line. These are not downloaded to
' the controller.
REM Comments can be made with REM on its own line (short for remark).
' Comments can also be made at the end of line with a : REM.
' Comments with the REM are downloaded to the controller and thus retrieved
' on upload.
GOSUB BasicABSMotion    : REM Subroutine for absolute moves.
GOSUB BasicINCMotion    : REM Subroutine for incremental moves.
GOSUB BasicCOMBOMotion
REM Subroutine for combination of absolute and incremental moves.
GOSUB JogABSMotion      : REM Subroutine for absolute jog moves.
```

```
GOSUB JogINCMotion      : REM Subroutine for incremental jog moves.
GOSUB JogCOMBOMotion
'GOTO MAIN              : REM Remove the ' to run this continuously.
END                     : REM Ends the program.
```

### Enable Drives Subroutine

```
_ENABLEDRIVE
DRIVE ON X Y       : REM TURNS ON OUTPUT TO ENABLE DRIVE.
INH 8465(3)        : REM Wait until drive enables or 3 seconds.
IF (BIT 8465) THEN PRINT "Axis0 is enabled"
REM AXIS0 IS ENABLE, PRINT MESSAGE
IF (NOT BIT8465) THEN PRINT "Axis0 is not enabled. Ending program. Check
Motion and Drive Status Panels for errors" : END
REM AXIS0 IS NOT ENABLED, PRINT MESSAGE AND END PROGRAM.
INH 8497(3)        : REM Wait until drive enables or 3 seconds.
IF (BIT 8497) THEN PRINT "Axis1 is enabled"
REM AXIS1 IS ENABLED, PRINT MESSAGE
IF (NOT BIT8497) THEN PRINT "Axis1 is not enabled. Ending program. Check
Motion and Drive Status Panels for errors" : END
REM AXIS1 IS NOT ENABLED, PRINT MESSAGE AND END PROGRAM.
RETURN             : REM RETURN BACK TO GOSUB
```

← Line Wraps

← Line Wraps

### Absolute Interpolated Motion Subroutine

```
' Subroutine of Basic Absolute Moves
_BasicABSMotion
' Interpolated multi-axis moves cause all axes to start and stop at
' the same time.
' The ACC DEC VEL are the trajectory settings:
ACC 10
DEC 10
STP 10
VEL 1
' ABSOLUTE MOVES
' X0, X1, Y0, Y1 etc are MOV (The MOV is implied and not required).
' These are interpolated moves and the first move will complete before the
' next interpolated move is started.
' The program continues execution (commands are not blocking) but will wait
' on next move until current move is done.
X0
X1
Y1
X-1 Y-1
X1 Y2
X0 Y0
INH -516 : REM Inhibit(pause) program until absolute moves are done .
REM The minus in the INH -516 is NOT bit 516, so this is waiting until the
REM InMotion bit turns off.
REM INH is only used with BIT so the BIT is not necessary.
REM Bit 516 is In Motion bit for Master0 - the trajectory calculator
```

```
REM for Prog0.
RETURN
```

### Incremental Interpolated Motion Subroutine

```
' Subroutine of Basic Incremental Moves
_BasicINCMotion
' The / is incremental, from wherever the motor currently is.
X/-8
Y/5
X/2 Y/-3
X/-2 Y/1
INH -516 : REM Inhibit program until incremental moves are done.
REM Bit 516 is In Motion bit for Master0 - the trajectory calculator
REM for Prog0.
RETURN
```

### Basic Absolute and Incremental Motion Subroutine

```
' Subroutine Of Basic both Absolute and Incremental Moves
_BasicCOMBOMotion
X/-4 Y0
X2 Y/-4
X/5 Y/5
X4 Y0
INH -516 : REM inhibit program until combo moves are done
RETURN
```

### Absolute Jog Moves Subroutine

```
' Subroutine of Jog Absolute Moves
' JOG ABS are single axis jog moves.
' Multiple jog moves from multiple axis are independent moves stopping
' at different times.
' They would use their own accel/decel/velocity settings
' (with axis in settings):
JOG VEL X1  : REM Note jog acc / dec / vel are per axis and thus the
              REM axis alias is necessary.
JOG ACC X10
JOG DEC X10
JOG VEL Y2
JOG ACC Y10
JOG DEC Y10
' JOG moves will interrupt the current move so for sequencing an INH
' is needed waiting for the JOG Active bit (bit 792 for axis0) is off.
' The jog offset moves has its own reference, independent of the coordinated
' motion. This allows offset of coordinated motion but can cause confusion.
_JogABSMotion
JOG ABS X-4
INH -792    : REM Inhibit program until jog move is done.
JOG ABS Y5
INH -824
```

```
JOG ABS X-2 Y-2
INH -792
INH -824
JOG ABS Y-1
INH -824
JOG ABS X0 Y0
INH -792
INH -824
RETURN
```

### Incremental Jog Moves Subroutine

```
' Subroutine of Jog Incremental Moves
_JogINCMotion
JOG INC X-8
INH -792    : REM Inhibit program until jog move done.
JOG INC Y-8
INH -824
JOG INC X2 Y2
INH -792
INH -824
RETURN
```

### Absolute and Incremental Jog Moves Subroutine

```
' Subroutine of Both Absolute and Incremental Moves
_JogCOMBOMotion
JOG INC X-4
INH -792    : REM Inhibit program until jog move done.
JOG ABS Y-2
INH -824
JOG INC X5
JOG ABS Y5  : REM This would start an absolute move on Y axis after starting
              REM incremental move on X axis.
INH -792
INH -824
JOG ABS X0 Y0
INH -792
INH -824
RETURN
```

### Homing Subroutine

```
_HOMING
JOG VEL X1        : REM Set axes jog parameters used during homing.
JOG ACC X10
JOG DEC X10
' HLBIT X0
REM X uses PosEOT (input0), NegEOT (input1), Home (input2).
REM The HLBIT LIMIT/HOME assignments are normally set in Configuration
REM Wizard. Uncomment the ' to use.
JOG HOMVF X0.1    : REM Set backup to home velocity.
```

```
JOG HOME X1       : REM Start homing X positive .
REM Infinite WHILE statement while X is trying to HOME.
WHILE ((NOT BIT 16134) AND (NOT BIT 16135))
WEND
REM Prints Information regarding "X" Axis homing.
IF (BIT 16134) THEN PRINT "X HOMING SUCCESSFUL"
IF (BIT 16135) THEN PRINT "X HOMING UNSUCCESSFUL"
JOG VEL Y1 : REM Set axes jog parameters used during homing.
JOG ACC Y10
JOG DEC Y10
' HLBIT Y3
REM Y uses PosEOT (input3), NegEOT (input4), Home (input5)
REM The HLBIT LIMIT/HOME assignments are normally set in Configuration
REM Wizard. Uncomment the ' to use.
JOG HOMVF Y0.1    : REM Set backup to home velocity
JOG HOME Y1       : REM start homing Y positive
REM Infinite WHILE statement while Y is trying to HOME.
WHILE ((NOT BIT 16166) AND (NOT BIT 16167))
WEND
REM Prints Information regarding "Y" Axis homing.
IF (BIT 16166) THEN PRINT "Y HOMING SUCCESSFUL"
IF (BIT 16167) THEN PRINT "Y HOMING UNSUCCESSFUL"
RETURN
ENDP
```

## Advanced Homing

This sample shows how to first home two axes to their respective home sensors, then perform an additional marker search so that they find and settle on their encoders' Z channels. This is a very high-precision homing strategy.

```
PROGRAM
DRIVE ON X Y
INH 8465 (3)
INH 8497 (3)
IF (NOT BIT 8465) THEN PRINT "X DIDN'T ENABLE" : END
IF (NOT BIT 8497) THEN PRINT "Y DIDN'T ENABLE" : END
    GOSUB HOMING : REM GO TO SUBROUTINE HOMING
END
' SUBROUTINE HOMING
_HOMING
JOG VEL X1 Y1           : REM Set axes jog parameters used during homing.
JOG ACC X10 Y10
JOG DEC X10 Y10
HLBIT X0 Y3             : REM If assigned in Config Wizard, remove this line.
' X uses PosEOT (input0), NegEOT (input1), Home (input2).
' Y uses PosEOT (input3), NegEOT (input4), Home (input5).
JOG HOMVF X0.1 Y0.1     : REM Set backup to home velocity.
JOG HOME X-1 Y1         : REM Start homing X negative, Y positive.
' Infinite WHILE statement while both are still trying to HOME.
```

← Line Wraps

```
WHILE (((NOT BIT 16134) AND (NOT BIT 16135)) AND ((NOT BIT 16166) AND (NOT
BIT 16167)))
WEND
' Prints Information regarding Y Axis homing.
IF (BIT 16166) THEN PRINT "Y HOMING SUCCESSFUL"
IF (BIT 16167) THEN PRINT "Y HOMING UNSUCCESSFUL"
' If X Axis homing successful, find  X encoder ref marker.
ACC 10                  : REM Set motion profile for MSEEK incremental move.
DEC 10 VEL 0.5 STP 10
IF (BIT 16134)
    MSEEK X(1,0)        : REM Performs search for index marker in
                          REM 1 incremental unit.
    INH -516            : REM Waits for Master to Not be "IN MOTION"
    IF (BIT 777)        : REM If Capture of Index Marker was complete.
        PRINT "X HOMING SUCCESSFUL" : REM Prints information.
    ENDIF
ENDIF
IF (BIT 16135) THEN PRINT "X HOMING UNSUCCESSFUL"
RETURN
ENDP
```

## Homing for XYZ System

This sample shows a sophisticated homing algorithm for a three-axis system. After homing each axis to a switch, an additional move is performed to settle on the Z channel of each encoder, which is an extremely repeatable way to home. Following that, the actual positions are preloaded. This allows the programmer to set the machine zero to any place desired while respecting that the machine's design may require homing to occur at a specific location, like at a Z pulse on a linear encoder.

The sample also makes use of a practice that should be sparingly used. There are two `WHILE` loops in the code below that contain `IF/THEN` and `GOTO` statements. In many programs, these cause problems because they do not return to the calling code. Here, however, the `GOTO` statements are used to bring the program to an error handler and eventually terminate the program, making this problem irrelevant.

```
PROGRAM
GOSUB EnableDrives
GOSUB HomeAll
REM Insert application code here.
END                 : REM End program.

_EnableDrives
DRIVE ON X Y Z
DWL 0.15    : REM Wait 150ms for servos to enable before commanding moves.
RETURN

_HomeAll
HLIM X3             : REM Enable limits.
HLIM Y3             : REM Enable limits.
HLIM Z3             : REM Enable limits.
JOG ACC X500        : REM Set jog accel for homing.
```

```
JOG DEC X500      : REM Set jog decel for homing.
JOG VEL X100      : REM Set jog velocity for homing.
JOG HOMVF X25     : REM Set jog final velocity for homing.
JOG ACC Y300      : REM Set jog accel for homing.
JOG DEC Y300      : REM Set jog decel for homing.
JOG VEL Y75       : REM Set jog velocity for homing.
JOG HOMVF Y15     : REM Set jog final velocity for homing.
JOG ACC Z100      : REM Set jog accel for homing.
JOG DEC Z100      : REM Set jog decel for homing.
JOG VEL Z25       : REM Set jog velocity for homing.
JOG HOMVF Z5      : REM Set jog final velocity for homing.
REM X Axis settings
SET 16152         : REM Backup to edge is enabled.
CLR 16153         : REM Backup to positive edge.
CLR 16154         : REM Set Final approach direction is positive.
REM Y Axis settings
SET 16184         : REM Backup to edge is enabled.
CLR 16185         : REM Backup to positive edge.
SET 16186         : REM Set Final approach direction is negative.
REM Z Axis settings
SET 16216         : REM Backup to edge is enabled.
SET 16217         : REM Backup to negative edge.
SET 16218         : REM Set Final approach direction is negative.
REM Home Z Axis first.
JOG HOME Z-1
REM Home Successful - BIT16198.  Home Failed - BIT16199.
WHILE (NOT BIT 16198)
    IF (BIT 16199) THEN GOTO HomeFailed
WEND
REM Z is successful, home X and Y.
JOG HOME X1 Y-1
REM X Home Successful   - BIT16134
REM X Home Failed       - BIT16135
REM Y Home Successful   - BIT16166
REM Y Home Failed       - BIT16167
WHILE (NOT BIT 16134 OR NOT BIT16166)
    REM Jump to User error routine if home fails.
    IF (BIT 16135 OR BIT 16167) THEN GOTO HomeFailed
WEND
REM Find the Z markers for each axis encoder for more accurate positioning.
REM MSEEK uses master move profile settings.
ACC 250 VEL 50 DEC 250 STP 250 JRK 1250
REM X axis ballscrew is 10 mm per motor rev, so command a move of 10.5.
MSEEK X(10.5,0)
REM Rising First Marker - Z Mark, ENC0
REM Hardware Capture Parameter - P12292
REM Capture Complete Flag - BIT777
REM Y axis ballscrew is 10 mm per motor rev, so command a move of 10.5.
MSEEK Y(10.5,0)
```

```
REM Rising First Marker - Z Mark, ENC1
REM Hardware Capture Parameter - P12548
REM Capture Complete Flag - BIT809
REM Z axis ballscrew is 5 mm per motor rev, so command a move of 5.5.
MSEEK Z(5.5,0)
REM Rising First Marker - Z Mark, ENC2
REM Hardware Capture Parameter - P12804
REM Capture Complete Flag - BIT841
REM All position counters are now set to 0 by successful MSEEKs.
REM If home sensors/Z marks are not at the desired
REM machine zero location, move to "true zero"
REM or preload current location settings "RES X97 Y57.5 Z4.4".
X-97 Y-57.5 Z -4.4
INH -516    : REM Wait until moves are complete.
RES X Y Z   : REM Reset all counters to 0.
REM Change master move profile back to normal operation settings.
ACC 750 VEL 250 DEC 750 STP 750 JRK 2250
REM Change jog profiles back to normal operation settings.
JOG ACC X1000 Y1000 Z200
JOG DEC X1000 Y1000 Z200
JOG VEL X300 Y300 Z100
RETURN      : REM Go back to main program execution.

_HomeFailed
IF (BIT 16199) THEN PRINT "Z Homing Failed"
IF (BIT 16135) THEN PRINT "X Homing Failed"
IF (BIT 16167) THEN PRINT "Y Homing Failed"
END
ENDP
```

### Open Sample

```
PROGRAM
' THIS PROGRAM IS INTENDED TO BE RUN FROM AN EXTERNAL TERMINAL
' SUCH AS HYPERTERMINAL.  USE PORT 5002 TO CONNECT WINSOCK TO
' ACR7000 ETHERNET.
CLEAR             : REM Clear any variables dimensioned in program.
DIM $V(1,10)      : REM Dimensions one string variable of length 10.
GOSUB OPENPORT    : REM Go to subroutine OPENPORT.
END
' SUBROUTINE OPENPORT
_OPENPORT
' Opens Ethernet Stream3 (PMM uses Stream1).
OPEN "STREAM2:" AS #1
' Continuous loop as long as "X" is not entered.
_LOOP1
' Set String Variable 0 to nothing.
$V0 = ""
PRINT #1, ""
PRINT #1, "What kind of fruit do you want?"
```

```
PRINT #1, "(A)pple, (B)anana, (C)oconut"
PRINT #1, "I would like to have a";
' Infinite WHILE loop if they do not enter anything.
WHILE ($V0 = "")
    $V0 = UCASE$(INKEY$(1))
    REM Stores Keyboard entry into String Variable 0
WEND
IF ($V0 = "A") THEN PRINT #1, "n Apple"
REM If "A" was entered, then print n Apple (reads as "an Apple").
IF ($V0 = "B") THEN PRINT #1, " Banana"
REM If "B" was entered, then print Banana.
IF ($V0 = "C") THEN PRINT #1, " Coconut"
REM If "C" was entered, then print Coconut.
IF ($V0 = "X") THEN GOTO LOOP2
REM If "X" was entered, then go to LOOP2 to terminate program.
IF ($V0 = CHR$(27)) THEN GOTO LOOP2
REM If "ESC key" was entered, then go to LOOP2 to terminate program.
GOTO LOOP1
_LOOP2
PRINT #1, "Program terminated"
CLOSE #1
RETURN
ENDP
```

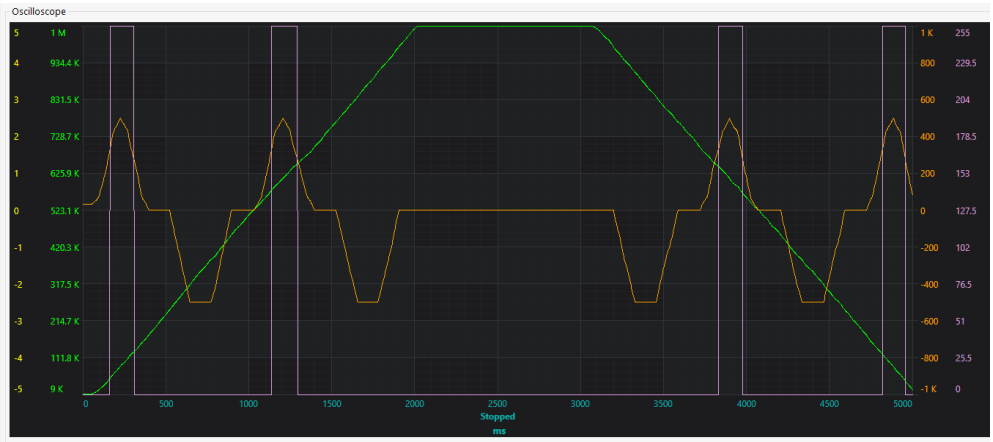
## Teach Array

```
PROGRAM
CLEAR             : REM Clear out any variables dimensioned.
DIM LV(2)         : REM Dimension 2 Long Variables.
DIM DA(1)         : REM Dimension 1 Double Array.
DIM $V(1,10)      : REM Dimension 1 String Variable of length 10.
' Go to subroutine Teach.
GOSUB Teach
END
_Teach
RES X             : REM Reset position to zero.
DRIVE OFF X       : REM Disable drive, teach points by manually moving motor.
' Start of InputPoints Routine
_InputPoints
' Print to the terminal "points to teach" and stores value into String
' Variable 0.
INPUT; "Enter number of points to teach (value must greater than 0) = "; $V0
' Stores the Value of String Variable 0 into Long Variable 1
LV1 = VAL($V0)
PRINT ""
PRINT LV1
' If statement to check if value entered is correct.
IF (LV1<=0)
    PRINT "ENTERED VALUE IS NOT VALID "; $V0
    PRINT "Value must be a number greater than 0"
```

```
    GOTO InputPoints
ENDIF
DIM DA0(LV1)       : REM Dimension array - number of points to teach.
' Use input 24 to tell controller to collect a teach point.
' Use a FOR/TO/STEP/NEXT loop to teach points into an array.
FOR LV0 = 0 TO (LV1-1) STEP 1
    PRINT "TURN MOTOR, THEN HIT INPUT 24 TO TEACH POINT"
    INH 24 : REM Inhibits the program until Input 24 is pressed
    DA0(LV0)=P12290/P12375     : REM Stores Double Array entry with
                                 REM Encoder Positive divide by PPU of Axis0.
    INH -24                    : REM Waits for Input 24 to turn off.
NEXT
PRINT "Teach Completed, Total Points Taught = ";LV1
PRINT "Press Input 24 to enable drives and move to zero/start"
INH 24
INH -24
DRIVE ON X                     : REM Enable Axis0 "X".
DWL 0.5
X0                             : REM Moves to zero position.
INH -516                       : REM Waits for motion to be completed.
PRINT "Input 24 to run taught points"
INH 24
' FOR/TO/STEP/NEXT loop to make absolute moves to position taught.
FOR LV0=0 TO (LV1-1) STEP 1
    X(DA0(LV0))
    INH -516
    PRINT DA0(LV0)             : REM Print the position to the terminal.
NEXT
RETURN
ENDP
```

## Programmable Limit Switch

A programmable limit switch (`PLS` command) turns on an output based on an axis position automatically when enabled. This position is based on an array variable.




Green: Master

Orange: Cam Axis

Pink: PLS Output

```
PROGRAM
```

```
ACC 10 DEC 10 STP 10 VEL 1
DRIVE ON Y Z
DWL 0.1
RES Y Z
GOSUB SetupArrays
CAM DIM Z2        : REM Define 2 cam segments
REM Define cam segment range and source
CAM SEG Z(0,(P12631*1/3),LA0)
CAM SEG Z(1,(P12631*2/3),LA1)
CAM SRC Z1        : REM Define cam source as ENC1
REM Set scale to cam axis PPU for 1/1.  500 cam counts = 500 encoder counts.
CAM SCALE Z(1/8000)
PLS0 SRC P12802
PLS0 DST P4100
PLS0 BASE LA2
PLS0 RATIO 0.01   : REM Array entries per input count
                    REM (500 for cam peak / 5 array for PLS).
PLS0 MASK 256
CAM SRC Z RES
CAM RES Z
CAM ON Z          : REM Start camming.
PLS0 ON
Y/2
DWL1
Y0
CAM OFF Z
PLS0 OFF
END

_SetupArrays
DIM LA(3)         : REM Dimension 2 long arrays.
DIM LA0(9)        : REM LAO has 9 elements.
LA0(00)=0         : REM Start defining LA0 cam table.
LA0(01)=73
LA0(02)=250
LA0(03)=427
LA0(04)=500
LA0(05)=427
LA0(06)=250
LA0(07)=73
LA0(08)=0
DIM LA1(6)
LA1(00)=0
LA1(01)=0
LA1(02)=-500
LA1(03)=-500
LA1(04)=0
LA1(05)=0
DIM LA2(5)
```

```
LA2(0) = 0
LA2(1) = 0
LA2(2) = 0
LA2(3) = 256
LA2(4) = 256
RETURN
ENDP
```

Note that output on position (OOP) is not supported on the IPA, so PLS should be used instead. ACR7000 products have OOP support in addition to PLS. OOP uses a different set of commands.

## EIP Scanner–Wago 750

```
' EtherNetIP Scanner Sample
' Valid for ACR7000 or IPA (not ACR9000).
' For use with Wago 750 series Ethernet/IP bus coupler.
' To check EthernetIP status in terminal emulator, use DIAG ETHIP.
PROGRAM
PBOOT             : REM LRUN to troubleshoot
REM Sample code for IPA or ACR7000 series as scanner to Wago 750 series
REM EtherNet/IP bus coupler.
REM Not compatible with ACR9000 or 3rd party EtherNet/IP devices.
P37392=1          : REM number of I/O nodes
P39424=((192<<24)+(168<<16)+(100<<8)+(28))
REM IP=192.168.100.28 on Node 0 - set this to Wago IP.
P39425=10         : REM Input data interval in ms.
P39426=10         : REM Output data repetition interval.
P39427=0
_START
ETHIP LOOK
PRINT "DISCOVERING"
INH -16674
PRINT "COMPLETE"
IF (bit 16682) THEN PRINT "Failed to find."
IF (P37397<>0)    : REM If any node is not discovered on network...
    DWL 1
    GOTO START
ENDIF
PRINT "Starting network..."
SET 16672         : REM START EIPIO.
WHILE (NOT (BIT16681 OR BIT16682))
WEND
IF (BIT16681)
    REM START SUCCESS
    PRINT "Starting success!"
ENDIF
IF (BIT16682)
    REM START FAILED
    PRINT "Start failed!"
ENDIF
```

```
ENDP
```

## Joystick

This sample uses EtherNet/IP (Wago 750 bus coupler) with two analog inputs to create an analog joystick.

```
#DEFINE XjoyRest  P0
#DEFINE YjoyRest  P1
#DEFINE Deadband  P2
#DEFINE XanalogIn P35332
#DEFINE YanalogIn P35340
#DEFINE Xon       BIT8465
#DEFINE Yon       BIT8497

PROGRAM
PBOOT       : REM LRUN to troubleshoot.
REM Sample code for IPA or ACR7000 series as scanner to Wago 750
REM series Ethernet/IP bus coupler.
REM Not compatible with ACR9000 or other 3rd party Ethernet/IP devices.
P39424=((192<<24)+(168<<16)+(100<<8)+(2))
REM IP=192.168.100.2 Node 0 verify in Status Panels > EtherNet/IP.
REM First 2 analog inputs on EtherNet/IP are X and Y joystick.
DIM LV(1)   : REM Dimension 1 local variable
lv0=0       : REM Reset value
GOSUB START
REM This sets default values for MOV (default interpolated moves)
ACC 10 DEC 10 STP 10 VEL 1
REM This sets default values for JOG (single axis offset moves)
JOG ACC X 10
JOG DEC X 10
JOG VEL X 1
REM In case already running, stop jogging
JOG OFF X Y
GOSUB ENABLEDRIVE : REM GO TO SUBROUTINE "ENABLEDRIVE"
REM This will then go to _ENABLEDRIVE and run that subroutine until
REM the return and then come back to this point.
_MAIN
IF (BIT16683) THEN PRINT "ETHERNET/IP NODE FAILURE" : JOG OFF X : END
IF ((XanalogIn < 0.1) OR (YanalogIn < 0.1)) THEN PRINT "JOYSTICK
DISCONNECTED" : JOG OFF X : END
REM X analog input is into P35332 (See Numeric Status > Ethernet/IP >
REM Node0 ADC > ADC Input Value).
REM Y analog input is into P35340
XjoyRest = 4.6    : REM X joystick resting voltage value.
YjoyRest = 3.823  : REM Y joystick resting voltage value.
Deadband = 0.1    : REM Deadband.
IF (NOT Xon) THEN PRINT "X AXIS NOT ON" : DWL 1 : GOTO MAIN
IF (XanalogIn > (XjoyRest + Deadband)) THEN JOG FWD X : PRINT "JOG X
SPEED",(XanalogIn-XjoyRest)
```

← Line Wraps

← Line Wraps

```
IF (XanalogIn < (XjoyRest - Deadband)) THEN JOG REV X : PRINT "JOG X
SPEED",((XanalogIn-XjoyRest)*-1)
IF ((XanalogIn < (XjoyRest + Deadband)) AND (XanalogIn > (XjoyRest -
Deadband))) THEN JOG OFF X
JOG VEL X (absf(XanalogIn-XjoyRest))
IF (NOT Yon) THEN PRINT "Y AYIS NOT ON" : DWL 1 : GOTO MAIN
IF (YanalogIn > (YjoyRest + Deadband)) THEN JOG FWD Y : PRINT "JOG Y
SPEED",(YanalogIn-YjoyRest)
IF (YanalogIn < (YjoyRest - Deadband)) THEN JOG REV Y : PRINT "JOG Y
SPEED",((YanalogIn-YjoyRest)*-1)
IF ((YanalogIn < (YjoyRest + Deadband)) AND (YanalogIn > (YjoyRest -
Deadband))) THEN JOG OFF Y
JOG VEL Y (ABSF(YanalogIn-YjoyRest))
DWL 0.1      : REM Loop execution very fast. This dwell slows down in case you
               REM do LRUN in the terminal so it does not flood the port.
GOTO MAIN
END          : REM Ends the program
_ENABLEDRIVE
REM ENABLE AXIS0
DRIVE ON X   : REM TURNS ON OUTPUT TO ENABLE DRIVE.
INH 8465(3)  : REM Wait until drive enables or 3 seconds.
IF (BIT 8465) THEN PRINT "Axis0 is enabled."
IF (NOT BIT8465) THEN PRINT "Axis0 is not enabled. Ending program. Check
Motion and Drive Status Panels for errors" : END
REM ENABLE AXIS1
DRIVE ON Y   : REM TURNS ON OUTPUT TO ENABLE DRIVE.
INH 8497(3)  : REM Wait until drive enables or 3 seconds.
IF (BIT 8497) THEN PRINT "Axis1 is enabled."
IF (NOT BIT8497) THEN PRINT "Axis1 is not enabled. Ending program. Check
Motion and Drive Status Panels for errors" : END
RETURN       : REM RETURN BACK TO GOSUB
_START
' EtherNet/IP Scanner sample
' Valid for ACR7000 or IPA (not ACR9000).
' For use with Wago 750 series EtherNet/IP bus coupler.
' To check EtherNet/IP status in terminal emulator, use DIAG ETHIP.
P37392=1     : REM Number of I/O nodes.
P39425=10    : REM Input data interval.
P39426=10    : REM Output data repetition interval.
P39427=0
ETHIP LOOK
PRINT "DISCOVERING..."
INH -16674
PRINT "COMPLETE"
IF (BIT16682) THEN LV0=LV0+1 : PRINT "Failed to find.", LV0
IF (LV0>=3) THEN END
IF (P37397<>0) THEN DWL 1 : GOTO START     : REM If any node is not discovered
                                             REM on the network...
PRINT "Starting network..."
```

← Line Wraps
← Line Wraps
← Line Wraps
← Line Wraps
← Line Wraps
← Line Wraps
← Line Wraps

```
SET 16672 : REM START EIPIO
WHILE (NOT (BIT16681 OR BIT16682))
WEND
IF (BIT16681)
    REM START SUCCESS
    PRINT "Start success!"
ENDIF
IF (BIT16682)
    REM START FAILED
    PRINT "Start failed!"
ENDIF
RETURN
ENDP
```

### Capture Data

```
PROGRAM
REM Program to set up multi-channel high-speed data capture.
REM Initialize local long variables.
DIM LV2
REM Initialize local arrays.
DIM LA(2)          : REM Dimension 2 long integer arrays
DIM LA0(500)       : REM Dimension 500 elements for Long Array0
DIM LA1(500)       : REM Dimension 500 elements for Long Array1
REM General sample settings.
SAMP CLEAR         : REM Clear current sampling settings
P6915 = 10         : REM Sample timer period in ms (0=servo period)
SAMP TRG +792      : REM Start recording on rising edge of axis0 jog
REM Note that motion would be within another program, typically Prog 0.
REM If capturing data for Master0 (Interpolated motion such as X Y), use
REM Master 0 In Motion bit 516.
REM Channel 0 sample settings
SAMP 0 SRC P12290 : REM Set the source to Axis 0 Actual Position.
SAMP 0 BASE LA0   : REM Array for recording data.
REM Channel 1 sample settings
SAMP 1 SRC P6916  : REM Set the source to Global System Clock.
SAMP 1 BASE LA1   : REM Array for recording data.
REM Begin
SET 104           : REM Arm sample trigger.
INH-104           : REM Wait for capture for all arrays to complete.
REM List both arrays of captured data.
REM To see in terminal emulator do LRUN, to exit press ESC key.
FOR LV1 = 0 TO 1 STEP 1
    PRINT "LA";LV1;" ARRAY"
    FOR LV0 = 0 TO 499 STEP 1
        PRINT LA(LV1)(LV0)
    NEXT
    LV0=0
NEXT
ENDP
```

## Peer-to-Peer

```
' EtherNet/IP Peer-to-Peer sample
' Valid for ACR7000 or IPA (not ACR9000).
' To check EtherNet/IP status in terminal, use DIAG ETHIP.
' To check EtherNet/IP status at peer in terminal, use CIP.
PROGRAM
PBOOT
GOSUB ConfigScanner
END
_ConfigScanner
P37392 = 0                                          : REM # I/O nodes.
P37393 = 1                                          : REM # Peer nodes.
P39680 = ((192<<24)+(168<<16)+(100<<8)+(2))         : REM IP for Peer 0 adapter
                                                      REM unit is 192.168.100.2.
P39681 = 10         : REM Input RPI.
P39682 = 10         : REM Output RPI.
P39683 = 0          : REM Participation mode, 0 = mandatory.
P39684 = 33         : REM Connection type.
P39685 = 4          : REM # of groups of data to be exchanged.
P39686 = 100        : REM Max consumed parameters.
P39687 = 100        : REM Max produced parameters.
P39696 = 12288      : REM Peer 0 Group 0 Start Parameter - This is the current
                      REM position of Axis 0 of Peer 0 which are Longs.
P39697 = 4          : REM Peer 0 Group 0 Length – Peer 0 P12288-P12291.
P39698 = 0          : REM Peer 0 Group 0 Direction - 0 is from Peer 0 to
                      REM Scanner.  The Scanner will read these 4 parameters
                      REM into Group 0 Long registers (P39936-P39939).
P39699 = 39400      : REM Peer 0 Group 1 Start Parameter - These are user
parameters which are Floats
P39700 = 2          : REM Peer 0 Group 1 Length – Peer 0 P39400, P39401.
P39701 = 1          : REM Peer 0 Group 1 Direction - 1 is from Scanner to
                      REM Peer0.  Scanner Group1 Floats are P40072 and P40073.
                      REM Writing values at Scanner will then be sent to P39400
                      REM and P39401 on Peer 0.
P39702 = 39300      : REM Peer 0 Group 2 Start Parameter - These are user
                      REM parameters which are Floats.
P39703 = 2          : REM Peer 0 Group 2 Length – Peer 0 P39300, P39301.
P39704 = 0          : REM Peer 0 Group 2 Direction - 0 Peer 0 to Scanner.
REM Writing values at Peer 0 P39300 and P39301 will be sent to Scanner Group
REM 2 Floats P40080 and P40081.
SET 16672           : REM Start the network.
RETURN
ENDP
```

## ACR7xT Status

```
REM Sample program to view axis status. Download to an empty program and then
REM LRUN from terminal emulator to read report. Other programs can be running
REM and will not stop. When downloading do not save to flash or it will stop
REM the other programs.
```

```
PROGRAM
CLEAR
DIM LV7     : REM Dimension/allocation 7 long local variables.
LV1=0
LV3=4       : REM ENTER THE NUMBER OF AXES USED.
?" Dim lists memory allocation for programs, streams, globals and defines.
must be done at sys prompt"
DIM         : REM Must be done at sys prompt.
ATTACH      : REM Lists program master and slave axis attach, alias.
ATTACH AXIS : REM Lists axis type (stepper/DAC), feedback.
FOR LV1 = 0 TO (LV3-1) STEP 1
?""         : REM prints blank line -- used for formatting.
?""
?"AXIS", LV1, " PULSES PER UNIT:  ", P(12375+LV1*256)
REM ACR Extended IO Settings
?"AXIS", LV1, " Enable Drive I/O: ", BIT (8468+LV1*32)
?"AXIS", LV1, " Enable CW/CCW (versus Step/Dir):  ", BIT (8464+LV1*32)
?"AXIS", LV1, " DEO Serves Shutdown Function: ", BIT (8470+LV1*32)
?"AXIS", LV1, " Enable EXC Response:  ", BIT (8469+LV1*32)
?"AXIS", LV1, " Invert Drive Fault Input Level:  ", BIT (8453+LV1*32)
?""
?""
REM Axis Gain Values
? "AXIS", LV1, " PGAIN", P(12304+LV1*256)
? "AXIS", LV1, " IGAIN", P(12305+LV1*256)
? "AXIS", LV1, " ILIMIT", P(12306+LV1*256)
? "AXIS", LV1, " IDELAY", P(12307+LV1*256)
? "AXIS", LV1, " DGAIN", P(12308+LV1*256)
? "AXIS", LV1, " DERIVATIVE WIDTH", P(12309+LV1*256)
? "AXIS", LV1, " FEEDFORWARD VEL", P(12310+LV1*256)
? "AXIS", LV1, " FEEDFORWARD ACC", P(12311+LV1*256)
? "AXIS", LV1, " PLUS TORQUE LIMIT", P(12328+LV1*256)
? "AXIS", LV1, " MINUS TORQUE LIMIT", P(12329+LV1*256)
? "AXIS", LV1, " FBVEL GAIN SETTING", P(12352+LV1*256)
REM Axis Limits
? "AXIS", LV1, " HLDEC:  ", P(12421+LV1*256)
? "AXIS", LV1, " Positive EOT Limit Level Invert:  ", BIT(16144+LV1*32)
? "AXIS", LV1, " Negative EOT Limit Level Invert:  ", BIT(16145+LV1*32)
? "AXIS", LV1, " Home Limit Level Invert:  ", BIT(16146+LV1*32)
? "AXIS", LV1, " Positive EOT Limit Enable:  ", BIT(16148+LV1*32)
? "AXIS", LV1, " Negative EOT Limit Enable:  ", BIT(16149+LV1*32)
?""
?""
REM AXIS0 SLM gives eval overflow error, must query via read-only parameters.
REM NOTE THESE P VALUES ARE MISSING IN HELP FILE
? "AXIS", LV1, " Positive Soft Limit:  ", P(12424+LV1*256)
? "AXIS", LV1, " Negative Soft Limit:  ", P(12425+LV1*256)
? "AXIS", LV1, " Soft Limit Decel:  ", P(12422+LV1*256)
```

```
? "AXIS", LV1, " Positive Soft Limit Enable", BIT(16150+LV1*32)
? "AXIS", LV1, " Negative Soft Limit Enable", BIT(16151+LV1*32)

?""

?""
REM This command enables the servo loop associated with an axis without using
REM the bit flag designated for this purpose.
? "AXIS", LV1, " NOT ENABLED (0 AXIS ENABLED, 1 AXIS DISABLED): ",
BIT(785+32*LV1)
?""
?""
REM STEPPER SETTINGS
? "AXIS", LV1, " User Maximum motor current (Amps):  ", P(7938+LV1*16)
? "AXIS", LV1, " Standby Current Percentage:  ", P(7944+LV1*16)
? "Standby Current Delay (msec.):  ", P(7945+LV1*16)
? "Micro-steps per full step (power of 2) =  ", P(7946+LV1*16)
? "Enable Auto Standby ", BIT(15618+LV1*32)
? "Assert Drive configuration ", BIT(15616+LV1*32)
? ""
? "AXIS", LV1, " STEPPER SETTING2 IN DEC IS: ", P(8066+LV1*16)
? "AXIS", LV1, " STEPPER SETTING3 IN DEC IS: ", P(8067+LV1*16)
? "AXIS", LV1, " STEPPER SETTING4 IN DEC IS: ", P(8068+LV1*16)
? ""
? "LV1= ", LV1
NEXT
ENDP
```

## ACR7xT Home to Hard Stop

```
PROGRAM
REM Stepper motors are open loop. Below code presumes encoder
REM attached for the step motor to detect hard stop.
CLEAR
DIM LV10    : REM DIMENSION 10 LOCAL VARIABLES
DRIVE ON X
JOG ACC X 10
JOG DEC X 10
JOG VEL X 1
GOSUB HOMEHARDSTOP

_MAIN
DWL 2             : REM WAIT 2 SECONDS
JOG ABS X 10      : REM MOVE TO POSITION 10
INH -792          : REM INHIBIT PROGRAM UNTIL MOVE IS DONE
DWL 2
JOG ABS X 0       : REM MOVE TO POSITION 0
INH -792
LV2=LV2+1
PRINT "CYCLES=", LV2
GOTO MAIN
```

```
_HOMEHARDSTOP
JOG REV X
WHILE (BIT 792)
    LV0=P6144
    DWL 0.1
    LV1=P6144
    IF (LV1>=LV0)
        JOG OFF X
    ENDIF
WEND
JOG INC X 1 : REM MOVE 1 REV OFF HARDSTOP
INH -792
JOG RES X   : REM RESET THIS JOG POSITION AS 0
RES X       : REM RESET THE CURRENT POSITION AS 0
PRINT "AT HOME"
RETURN
ENDP
```

## Time Subroutine

This Time subroutine implements a "clock" for showing time since power up or reboot, assuming P6916 is not reset by the user. P6916 resets at $2^{31}$, or 2,147,483,648. P6916 is a free-running clock in milliseconds.

This could be added to a program or, if a program is already running, downloaded into an empty program. To see the values, go into the Terminal Emulator and type `LRUN` at the program prompt after downloading.

```
#DEFINE Time LV0
#DEFINE ms LV1
#DEFINE seconds LV2
#DEFINE ExcSeconds LV3
#DEFINE minutes LV4
#DEFINE ExcMinutes LV5
#DEFINE hours LV6
#DEFINE ExcHours LV7
#DEFINE days LV8
DIM LV10
_CheckTime
Time = P6916 : REM capture current time in ms.

REM Extract the millisecond portion.
ms = Time MOD 1000            : REM Extract any ms less than 1 full second.

REM Extract the second portion.
REM Remove ms from the Time and convert time to seconds.
seconds = (Time - ms)/1000
ExcSeconds = seconds MOD 60   : REM Extract any seconds less than
                                REM a full minute.

REM Extract the minute portion.
REM Remove seconds from the Time and convert time to minutes.
```

```
minutes = (seconds - ExcSeconds) / 60
REM Extract any minutes less than a full hour.
ExcMinutes = minutes MOD 60

REM Extract the hour portion.
REM Remove excess minutes and convert to full hours.
hours = (minutes - ExcMinutes) /60
REM Remove any hours less than a full day.
ExcHours = hours MOD 24

REM only full days are left. Only works up to <25 days.
REM Remove excess hours and convert what is left to days.
days = (hours - ExcHours)/24

PRINT "Approximate Time Running : ";days;" Days ";
PRINT USING "##";ExcHours;" Hours ";
PRINT USING "##";ExcMinutes;" Minutes ";
PRINT USING "##";ExcSeconds;".";ms;" Seconds "
RETURN
```

## Error Recovery (IPA)

This sample on error handling addresses error checking and recovery, which should be programmed into each application. Error handling is then done automatically as the application runs and is helpful in diagnosing problems.

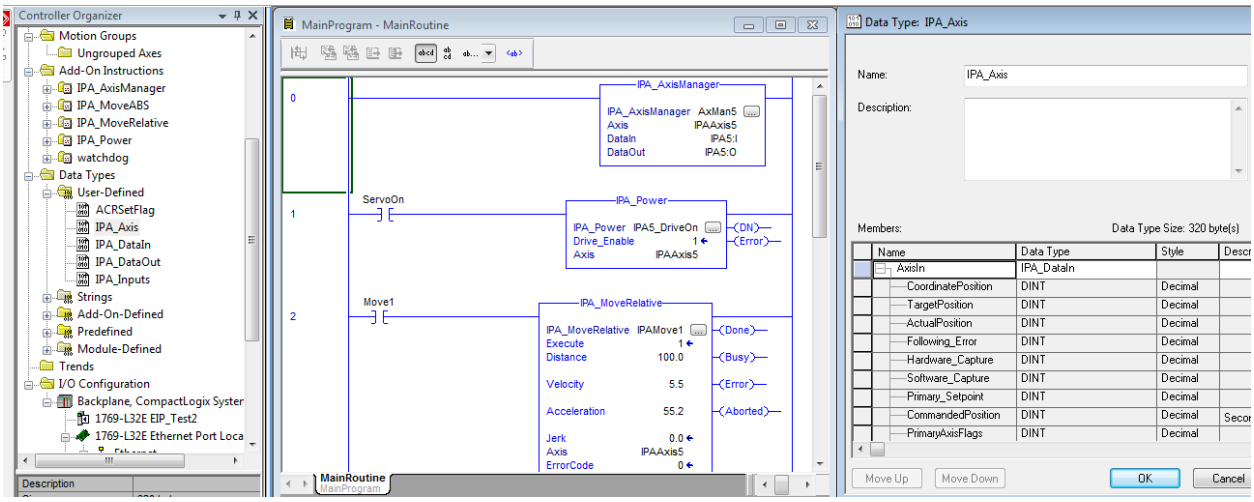
```
PROGRAM
PBOOT
REM Error recovery program.
_LOOP
IF (BIT8467)
    REM Kill all motion was signaled, check causes
    IF (BIT8465)
        REM DRIVE IS STILL ENABLED
        GOSUB CheckLimits
    ELSE IF (BIT10009)
        REM TORQUE ENABLE INPUT WAS OPENED
        GOSUB CheckTorqueEnable
    ELSE IF (BIT9498)
        REM DRIVE FAULTED
        AXIS0 DRIVE RES
        INH -8475 : REM WAIT FOR RESET TO COMPLETE
        IF (BIT9498)
            GOTO FaultLatched
        ENDIF
    ELSE IF (BIT8479)
        REM EXCESS POSTION ERROR WAS TRIPPED
        CLR 8467 CLR 522
    ENDIF
ENDIF

IF (NOT BIT8467)
```

```
    REM Kill has been cleared, restart Program 0.
    RUN PROG0
ENDIF
GOTO LOOP

_CheckTorqueEnable
IF (BIT10011)
    REM Torque enable inputs mismatch.
    REM Requires a HARD power cycle.
    ? "TORQUE ENABLE HEALTH EVENT"
    ? " CYCLE POWER"
    GOTO FaultLatched
ENDIF
WHILE (BIT10010)
    REM WAIT HERE UNTIL THE INPUT IS CLOSED
WEND
CLR 8467
AXIS0 DRIVE ON
RUN PROG0
RETURN

_CheckLimits
IF (BIT16132 OR BIT16133)
    REM HARD LIMIT WAS HIT
    CLR 8467
    CLR 522
ENDIF
IF (BIT16136 OR BIT16137)
    REM SOFT LIMIT WAS HIT
    CLR 8467
    CLR 522
ENDIF
RETURN

_FaultLatched
?"DRIVE FAULT DID NOT CLEAR, CHECK HARDWARE"
?"requires a HARD power cycle"
END
ENDP
```

## Add-On Instructions (AOIs) for IPA

The IPA and ACR7000 are compatible for use with CompactLogix and ControlLogix PLCs and can utilize both Class 1 I/O messaging and Class 3 MSG instructions. Add-On Instructions are available to enhance the development of IPA applications within the RSLogix environment.

This includes an IPA program already written that you can adjust (to set their desired units and select the motor, etc.) and AOIs and UDTs you import into RSLogix. These make it easy to control the IPA with these function blocks directly on the ladder logic:




These AOIs include over 400 predefined boolean tags and 50 parameters that update every EtherNet/IP cycle (adjustable, default 10 ms) for the PLC to read/write to the IPA, including most of the commonly used bits and parameters (axis, master, program, etc.). The AOIs use these tags, but users can also use them.

The AOIs can be downloaded from the IPA product page here.

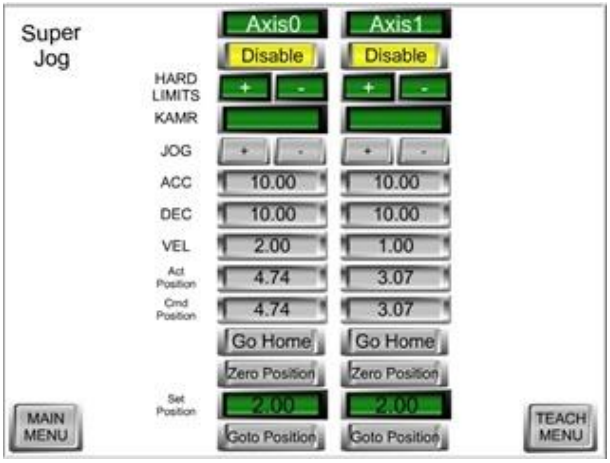
Further details can be found in the EtherNet/IP Programmer's Guide for IPA. Available AOIs include:

- IPA_ServoOn
- IPA_ServoOff
- IPA_Home
- IPA_Move
- IPA_MoveStop
- IPA_MoveVelocity
- IPA_SetPosition
- IPA_SetTorqueLimit
- IPA_Fault Reset

## Xpress HMI with ACR7000

The Xpress HMI is a compatible HMI for the ACR7000 for applications that need an operator touchscreen interface. This sample includes 2-axis jog and teach panels.

Click here for a complete sample on parker.com.

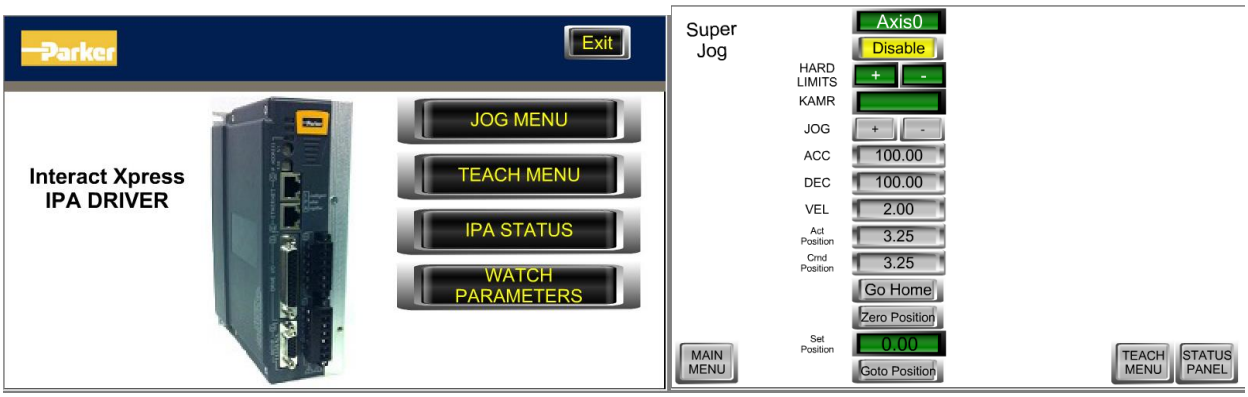



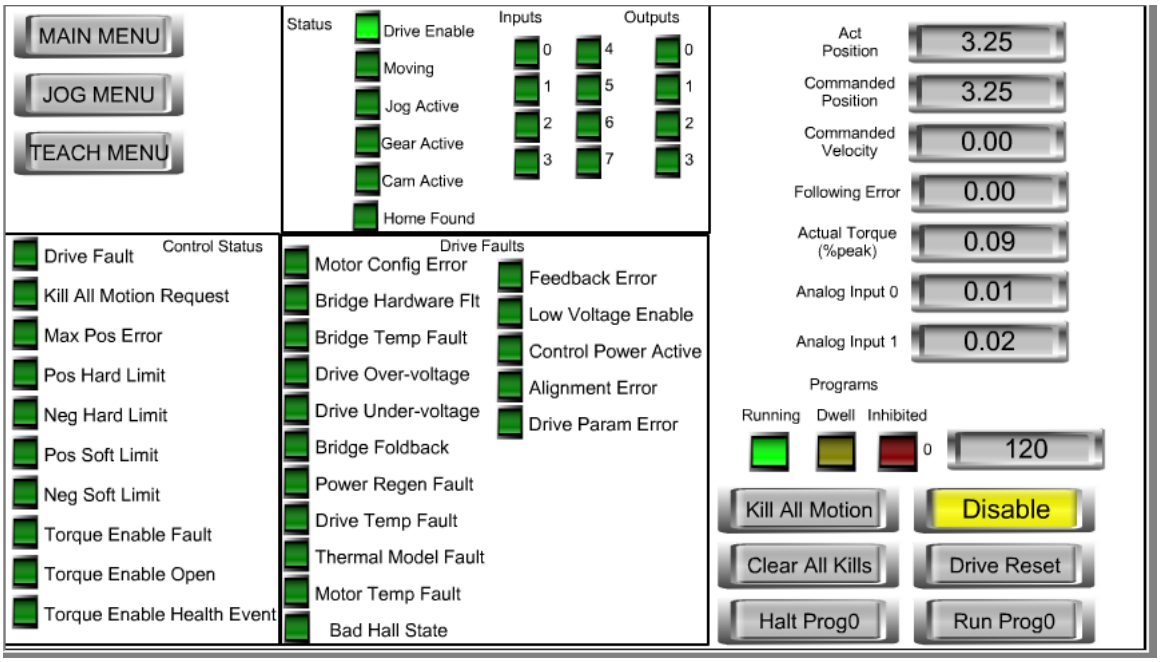


## Xpress HMI with IPA

The Xpress HMI is a compatible HMI for the IPA for applications that need an operator touchscreen interface. This sample includes a jog panel similar to PMM's and a status panel similar to PMM's Common Status Panel.

Click here for a complete sample on parker.com.


Parker
Exit
Interact Xpress
IPA DRIVER
JOG MENU
TEACH MENU
IPA STATUS
WATCH PARAMETERS
Super Jog
Axis0
Disable
HARD LIMITS
KAMR
JOG
ACC
100.00
DEC
100.00
VEL
2.00
Act Position
3.25
Cmd Position
3.25
Go Home
Zero Position
Set Position
0.00
Goto Position
MAIN MENU
TEACH MENU
STATUS PANEL


MAIN MENU
JOG MENU
TEACH MENU
Status
Drive Enable
Moving
Jog Active
Gear Active
Cam Active
Home Found
Inputs
Outputs
Act Position
3.25
Commanded Position
3.25
Commanded Velocity
0.00
Following Error
0.00
Actual Torque (%peak)
0.09
Analog Input 0
0.01
Analog Input 1
0.02
Control Status
Drive Fault
Kill All Motion Request
Max Pos Error
Pos Hard Limit
Neg Hard Limit
Pos Soft Limit
Neg Soft Limit
Torque Enable Fault
Torque Enable Open
Torque Enable Health Event
Drive Faults
Motor Config Error
Bridge Hardware Flt
Bridge Temp Fault
Drive Over-voltage
Drive Under-voltage
Bridge Foldback
Power Regen Fault
Drive Temp Fault
Thermal Model Fault
Motor Temp Fault
Bad Hall State
Feedback Error
Low Voltage Enable
Control Power Active
Alignment Error
Drive Param Error
Programs
Running
Dwell
Inhibited
120
Kill All Motion
Disable
Clear All Kills
Drive Reset
Halt Prog0
Run Prog0

# Testing Programs

PMM's Panels and Oscilloscopes give visual indications of the controller status. The Motion Status Panel lists the program line numbers as well as axis positions, program status and common errors. This same information is in the Bit Status Panel and Numeric Status Panel but placed in a single panel. You can have your own list of bits and parameters with Watch Lists, a new feature of PMM.

If a program stopped running, the line number will be the last line executed when the program halted. This can help determine why the program stopped, if there is a syntax error or if motion is commanded and the there is a KAMR (Kill All Motion Request). If the line number is 0, the program never ran.

If a program is starting but you are not sure about the program flow, you can insert `PRINT` statements in the program. As an example see EIP scanner - Wago 750. Print statements can be your own programmer notes (strings are in quotes), parameter values such as motor positions or global parameters that may be updated from external HMI/PLC/PC connections.

## *Program Not Running?*

There are several common reasons a program might not run:

**Syntax error in the code.** This is not as likely unless the syntax issue is in a part of code that does not always run, but instead runs based on some early logic and causes the program to sometimes take another branch.

Check the Motion Status Panel for a line number. This will be set to the line last executed when the program halted. In the Terminal Emulator, go to program prompt and click *List With Line Numbers* to see the program with line numbers in controller's memory (these are automatically assigned on download).

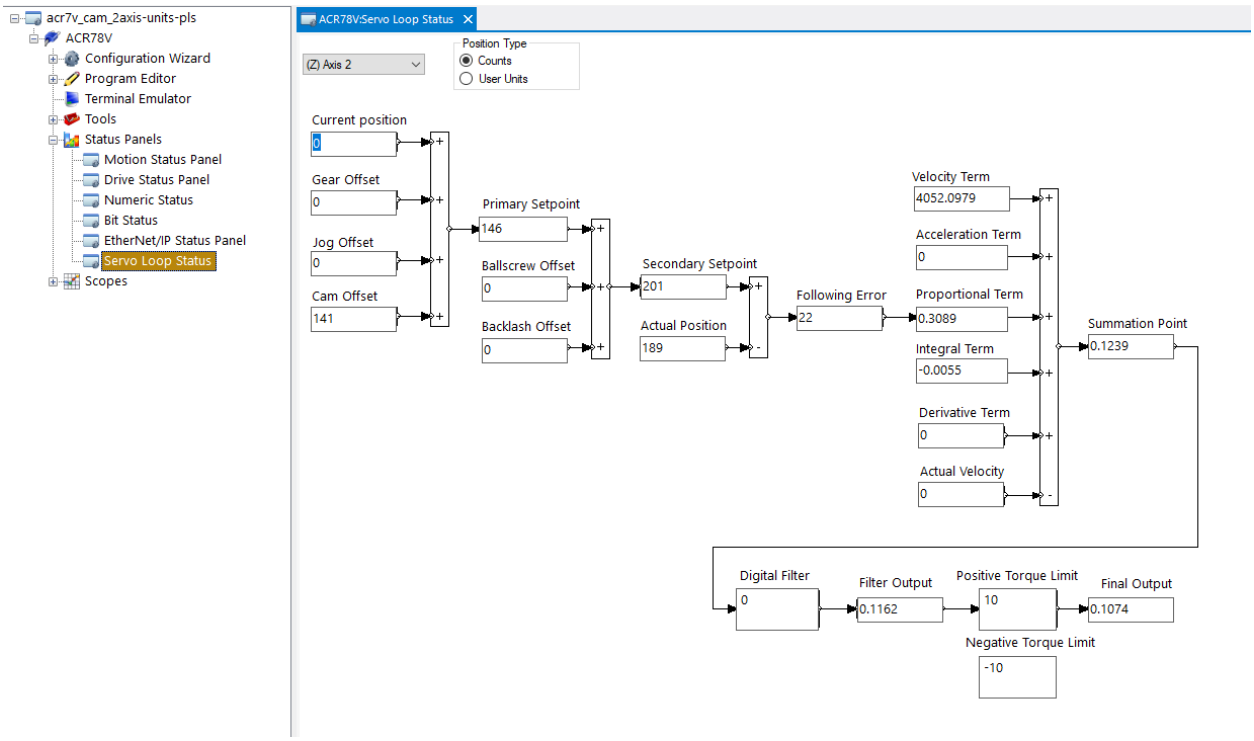
**HALT command.** A legitimate command to halt a program via the `HALT` command or something setting the program's control flag directly (BIT1033 Program Halt Request for Prog 0). Is there an HMI or other external connection used in the system normally during operation such as a PLC or PC?

**Axis not ready.** Trying to command a move before the axis is able to move, either because the KAMR flags have not been cleared yet or drive is not fully enabled yet, etc.

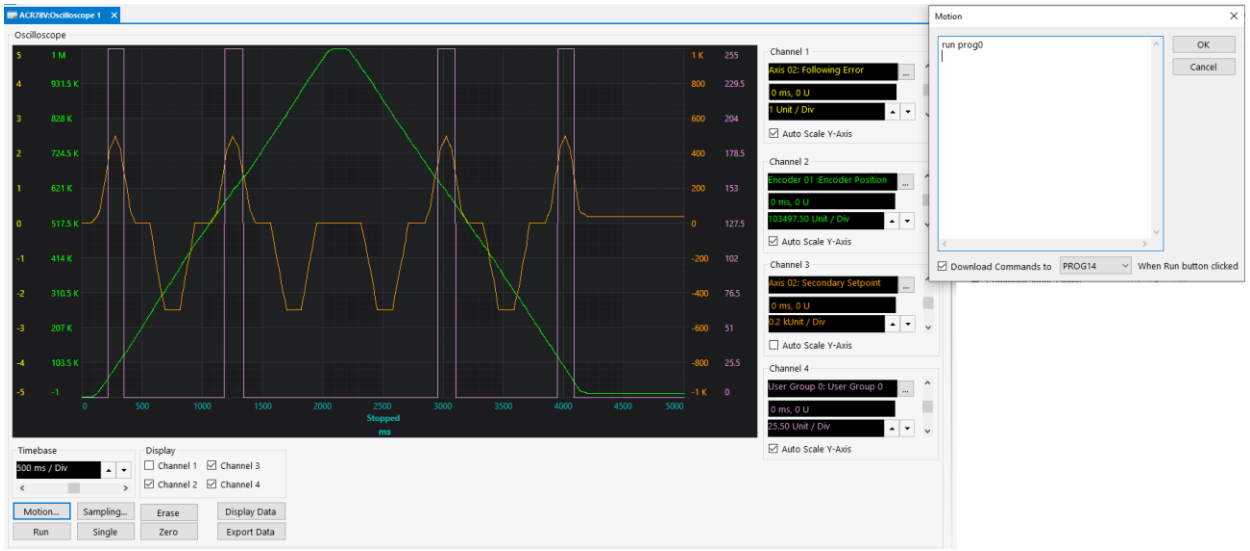
**Out of memory.** This would not generally occur on first power up, more if you were using `GOSUB` commands and the subroutines did not always have a `RETURN` from them. In this case, the pointers created by the `GOSUB` branch would not always be cleared by doing a `RETURN` from the subroutine and these pointers would build up in memory over time and eventually exceed the memory allocation for that particular program space. The result is a program crash.

## *Axis Motion Status?*

Each axis can have multiple types of motion as we have seen from the Setpoint Summation explanation. Visually, we can view axis motion status in *Status Panels* → *Servo Loop Status* and select the axis and desired units (counts or user units):




# *Graphing with Oscilloscopes*




PMM includes a 4-channel oscilloscope and you can use it to graph any parameter and any flag, including user parameters and user flags.

## Sampling

The channels can be sampled on a regular basis from the PC or at higher speed with onboard sampling. This will start with the selected bit trigger, typically the Master In Motion bit or, for a single axis, its Jog Active bit, though

any bit can be used in either rising edge (bit turning on starts the sampling) or falling edge (bit turns off starts the sampling) mode.

Note that Program 14 has a large memory allocation set as default to use local arrays to capture the selected channels. After sampling is complete, if onboard was used, the data will then be uploaded to the PC and graphed. *Run* will run the code specified by the *Motion* button continuously and graph it. *Single* will run the code within *Motion* one time and graph.

The channels can be toggled on/off with the checkboxes. Users can display the numeric data with Display Data and Export Data that they can then import into Excel or other programs for further analysis.

To scope a program, change the *Motion* code to "`RUN PROG0`" but leave *Download Commands* set to PROG14 (as shown in picture above); that sets where the data collection is done and we do not want it to conflict with the arrays used in our user PROG0.

# *Adding Lines of Code to Programs*

You can add lines of code to a program that is already downloaded to the controller. This can be useful when testing or debugging an application when you do not want to make a permanent change to the program stored in PMM. This would be done in PMM's Terminal Emulator while online with the controller.

Each code statement you want to add must include a line number. Otherwise, the controller could not understand where to place each code statement. To determine the correct line numbers, go into the Terminal Emulator in Parker Motion Manager and click *List Line Number*.

This turns on line numbering with the Force Line Numbers with List bit (bit 5651) and then sends a `LIST` command. The `LIST` command displays the current program.

Having determined the correct line number placement for the code statements, enter the line number, a space and the command. For example:

```
15 VEL 10
```

The new program lines are stored in the program space.

**NOTE: Code changes made with this procedure are not reflected in the program stored in PMM. To ensure your changes are permanent, enter them in the PMM Program Editor and download it to the controller with the *Save to Flash* option checked.**

# *Trace a Program*

PMM's Motion Status Panel lists the program line numbers while the program is running. The program can run faster than this panel updates unless a dwell or inhibit is encountered. To see the stream of line numbers as they are executed go into the Terminal Emulator and type `TRON` and then `LRUN` at the correct program prompt (e.g. `P00>`). This will then echo the line numbers of the program in sequence. This can be helpful to determine if

certain lines are executed or not in case of conditionals. To exit listen mode, press *Escape*. To turn trace mode off, use TROFF.

```
SYS>PROG0
P00>NEW
P00>10 DIM LV1
P00>20 LV0 = 0
P00>30 PRINT CHR$(65+LV0);
P00>40 LV0 = LV0+1
P00>50 IF (LV0 < 3) THEN GOTO 30
P00>60 PRINT
P00>TRON
P00>LRUN
<10><20><30>A<40><50><30>B<40><50><30>C<40><50><60>
P00>TROFF
P00>LRUN
ABC
P00>_
```

# CHAPTER 5
# Binary Host Interface

# Binary Host Interface

You can enhance communications with the ACR series controller through the binary host interface.



**NOTE: For Windows applications, it usually makes much more sense to use the ComACRServer6 API, a COM server designed to automation communications to ACR controllers. ComACRServer6 is compatible with VB.NET, C#, VBA and many other Windows programming tools (any that can use COM).**

## *Binary Data Transfer*

The binary data transfers in this chapter consist of a control character (Header ID) followed by a stream of data encoded according to the current state of the `MODE` command. Note that regardless of the mode, the Header ID is never converted during binary data transfer.

**NOTE: Much of the data in this section pertains specifically to serial/parallel communications on older ACR products. The protocol has not been changed to preserve backwards compatibility with older applications. Information relating to serial/parallel data transfer is not relevant to ACR7000 and IPA controllers, which only have Ethernet.**

During binary transfers to the controller, the delay between bytes must be no more than the communications timeout setting for the given channel. If the timeout activates, the transfer is thrown out and the channel goes back to waiting for a normal character or a binary header ID. The default communication timeout is 50 milliseconds.

The following is a list of valid data conversion modes. The default mode for the FIFO channel is zero and the default for the COM1 and COM2 channels is one. Note that high bit stripping cannot be done without also activating the control character-prefixing mode.

| Mode | Description |
|---|---|
| MODE 0 | No Conversion (recommended for Ethernet) |
| MODE 1 | Control Character Prefixing |
| MODE 2 | No Conversion |
| MODE 3 | Control Character Prefixing and High Bit Stripping |

## Control Character Prefixing

Control character prefixing follows Kermit communications protocol conventions. The escape code for control character prefixing is the '#' character. The control character-prefixing mode prevents valid data within a binary packet from being confused with the serial XON / XOFF flow control codes.

## Transmitting

If the character to be sent is either a 0x7F or a character in the range of 0x00 to 0x1F, the character is 'XORed' with 0x40 and proceeded with a '#' character. Otherwise, the byte is sent normally.

For example, if the character to be sent is 0x01, the character is transmitted as a "#A" string ( 0x01 XOR 0x40 = 0x41 = 'A' ). The special case where the character to be sent is the '#' character is handled with the two character "##" string.

## Receiving

When receiving control prefix encoded data, a '#' character is thrown away and causes the next character to be read from the data stream. If the character is in the range of 0x3F to 0x5F, the character is 'XORed' with 0x40 to decode the true value. Otherwise, the character is used exactly as read from the stream.

## High Bit Stripping

High bit stripping follows Kermit communications protocol conventions for 7-bit data paths. The escape code for high bit stripping is the '&' character and must be used in conjunction with the control character prefixing described above.

High bit stripping is for cases in which a 7-bit data path must be used for binary data transfer. This mode introduces a large overhead in the transfer of binary data since over half of the bytes are expanded to two-byte sequences and several are expanded to three bytes. If possible, an 8-bit data path should be used for binary data transfer.

### Transmitting

If the character to be sent is greater than 0x7F, the character is 'ANDed' with 0x7F and proceeded with the '&' character. Note that the AND may result in a control code which must then be handled by control character prefixing. The original character may also need to be sent with control character prefixing.

For example, if the character to be sent is 0xC2, the character is transmitted as a "&B" string ( 0xC2 AND 0x7F = 0x42 = 'B' ). As another example, if character to be sent is 0x83, the character is transmitted as the three character "&#C" string ( 0x83 AND 0x7F = 0x03 (control character) ). The special case where the character to be sent is the '&' character is handled with the two character "#&" string.

### Receiving

When receiving high bit encoded data, '#' characters are handled as normal control character prefix sequences. If the received character is neither a '#' nor a '&' character, the character is used exactly as read from the stream.

If the received character is the '&' character, it is thrown away and causes the next character to be read from the data stream. This new character may be a '#' character, which will initiate control prefix decoding sequence. The result is a value in the range of 0x00 to 0x7F, which is then 'ORed' with 0x80 to reestablish the high bit in the data.

## *Binary Data Packets*

Packets allow binary access to system parameters at any time. This method must be used if commands are sitting in the input queue since `PRINT` statements would also be buffered. The packet is the quickest way to access information such as current position and following error for display in an application program.

### Packet Request

Packets are requested by sending a four-byte binary request record. The following is a list of the bytes contained in this record:

| Data Field | Description |
|---|---|
| Byte 0 | Header ID ( 0x00 ) |
| Byte 1 | Group Code |
| Byte 2 | Group Index |
| Byte 3 | Isolation Mask |

### Group Code and Index

The group code and group index work as a pair to select the data coming back in a data packet. The group code selects a general data grouping and the group index selects a set of eight fields within that group. The isolation mask then selects which of these eight fields is to compose the final data packet.

### Isolation Mask

The isolation mask acts as a filter to select only the specific data required (for example, actual position for Axis 2, Axis 3 and Axis 5). If a bit is set in this mask, the corresponding data field is allowed to return in the data packet. In order to return all eight fields, the isolation mask must be 0xFF. Mask bit 0 is used to isolate the first field in a group and bit 7 is used to isolate the last field.

## Parameter Access

The following is a list of groups and what the isolation mask will isolate:

| Group | Description | Isolation Usage |
|---|---|---|
| 0x10 | Flag Parameters | Eight consecutive parameters |
| 0x18 | Encoder Parameters | ENC0-ENC15 |
| 0x19 | DAC parameters | DAC0-DAC7 |
| 0x1A | PLC parameters | PLC0–PLC7 |
| 0x1B | Miscellaneous | Eight consecutive parameters |
| 0x1C | Program Parameters | PROG0 - PROG15 |
| 0x20 | Master Parameters | MASTER0 - MASTER7 |
| 0x28 | Master Parameters | MASTER8 - MASTER15 |
| 0x30 | Axis Parameters | AXIS0 - AXIS7 |
| 0x38 | Axis Parameters | AXIS8 - AXIS15 |
| 0x40 | CMT Parameters | CMT0 - CMT7 |
| 0x50 | Logging Parameters | Eight consecutive parameters |
| 0x60 | Encoder Parameters | ENC16 - ENC23 |

## Packet Header

After a packet request is received, the ACR responds by sending back a four-byte packet header. This header is a direct echo of the request record. The echoing allows host software to do asynchronous sampling. A request can be sent by one part of the program and packet retrieval can be done by a centralized receiver. This routine would recognize the 0x00 in the header as an incoming packet and act accordingly.

In a synchronous retrieval mode, it is possible for extra data to be in front of an incoming packet header. This would occur if there were any ASCII data pending at the time of the request, such as during a `LIST`. In order to retrieve a packet correctly, the host software must be able to process this data while waiting for the packet header to arrive. This should not be a problem, however, if all system echoing is turned off and no ASCII data retrieval is being done.

## Packet Data

After the packet header is received, the data arrives as a set of four-byte fields. The bits in the isolation mask determine the number of fields and what they apply to. If the mask is 0xFF, a total of eight fields (32 bytes) would follow. The first field to be returned corresponds to the bit position of the lowest bit in the mask that is set.

Long integers (LONG) are returned as a four-byte field. Floating point numbers (FP32) are returned in 32-bit IEEE floating-point format. Both types of field are returned with the low order byte first.

### Usage Example

This example requests actual positions from axes 2, 3 and 5:

| Fields: | Header | Axis 2 | Axis 3 | Axis 5 |
|---|---|---|---|---|
| Output: | 00 30 02 2C | | | |
| Input: | 00 30 02 2C | 20 21 22 23 | 30 31 32 33 | 50 51 52 53 |

Actual Positions:

- AXIS2: 0x23222120
- AXIS3: 0x33323130
- AXIS5: 0x53525150

# *Binary Parameter Access*

Binary parameter access provides a method of reading from and writing to single system parameters on the controller. Unlike binary data packets, binary parameter access uses the index of the parameter directly from Appendix A. There are no groups or masks.

A parameter access header consists of a Header ID (0x00) followed by a Packet ID code and a two-byte parameter index. The Packet ID codes for the different types of packets are shown below. The following pages define each of the packets in detail.

***Packet ID Codes***

| Code | Packet Type | Description |
|---|---|---|
| 0x88 | Binary Get Long | Receive long integer from controller |
| 0x89 | Binary Set Long | Send long integer to controller |
| 0x8A | Binary Get IEEE | Receive IEEE value from controller |
| 0x8B | Binary Set IEEE | Send IEEE value to controller |

### Usage Example

This example requests current position from axis 0 parameter P12288:

| Fields: | Header | Parameter Value |
|---|---|---|
| Output: | 00 88 00 30 | |
| Input: | 00 88 00 30 | 10 11 12 00 |

Current Position Parameter Value:

- AXIS0: 0x00121110

## Binary Get Long

This packet gets a single parameter from the controller. The parameter index is a two-byte value sent low-order byte first. The parameter value in the receive packet is a four-byte long integer received low-order byte first.

*Transmit Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x88) |
| Byte 2-3 | WORD | Parameter Index |

*Receive Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x88) |
| Byte 2-3 | WORD | Parameter Index |
| Byte 4-7 | LONG | Parameter Value |

## Binary Set Long

This packet sets a single parameter on the controller. The parameter index is a two-byte value sent low-order byte first. The parameter value is a four-byte long integer and is sent low order byte first.

*Transmit Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x89) |
| Byte 2-3 | WORD | Parameter Index |
| Byte 4-7 | LONG | Parameter Value |

*Receive Packet*

None.

## Binary Get IEEE

This packet gets a single parameter from the controller. The parameter index is a two-byte value sent low-order byte first. The parameter value in the receive packet is a four-byte image of an IEEE floating point number received low-order byte first.

*Transmit Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x8A) |
| Byte 2-3 | WORD | Parameter Index |

*Receive Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x8A) |
| Byte 2-3 | WORD | Parameter Index |
| Byte 4-7 | IEEE32 | Parameter Value |

### Binary Set IEEE

This packet sets a single parameter on the controller. The parameter index is a two-byte value sent low-order byte first. The parameter value is a four-byte image of an IEEE floating point number and is sent low-order byte first.

*Transmit Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x8B) |
| Byte 2-3 | WORD | Parameter Index |
| Byte 4-7 | IEEE32 | Parameter Value |

*Receive Packet*

None.

# *Binary Peek Command*

A binary peek command consists of a four-byte header followed by an address and the data to be fetched from that address. The header contains a data conversion code that controls pointer incrementing and the FP32 → IEEE floating point conversion. The conversion only applies to older ACR controllers. The ACR7000 and IPA use ARM Cortex processors, which support 32-bit IEEE754 floating point values natively.

**NOTE: Refer to Binary Global Parameter Access Note at end of Binary Host Interface section for details.**

The command returns the header and peek address followed by the requested data.

### *Transmit Packet*

| Data Field | Description |
|---|---|
| Byte 0 | Header ID (0x00) |
| Byte 1 | Packet ID (0x90) |
| Byte 2 | Conversion Code |
| Byte 3 | Peek Word Count |
| Long 0 | Peek Address |

### *Receive Packet*

| Data Field | Description |
|---|---|
| Byte 0 | Header ID (0x00) |
| Byte 1 | Packet ID (0x90) |
| Byte 2 | Conversion Code |
| Byte 3 | Peek Word Count |
| Long 0 | Peek Address |
| Long 1 | Peek Data 0 |
| Long 2 | Peek Data 1 |
| : | |
| Long N | Peek Data (Count - 1) |

### *Conversion Codes*

| Code | Source | Destination |
|---|---|---|
| 0x00 | LONG | LONG |
| 0x01 | FP64 | IEEE32 |
| 0x02 | FP32 | IEEE32 |

### Usage Example

This example peeks at three words, starting at peek address 0x404500.

**NOTE: Addresses shown are for example only. Addresses will vary from controller to controller depending on system memory allocation.**

| Fields: | Header | Address | Data0 | Data1 | Data2 |
|---|---|---|---|---|---|
| Output: | 00 90 00 03 | 00 50 40 00 | | | |
| Input: | 00 90 00 03 | 00 50 40 00 | 10 11 12 13 | 20 21 22 23 | 30 31 32 33 |

Requested data at address:

- 0x405000: 0x13121110
- 0x405001: 0x23222120
- 0x405002: 0x33323130

# *Binary Poke Command*

A binary poke command consists of a four-byte header followed by an address and the data to be stored at that address. There is no information returned from this command. The header contains a data conversion code that controls pointer incrementing and the IEEE → FP32 floating point conversion. The conversion only applies to older ACR controllers. The ACR7000 and IPA use ARM Cortex processors, which support 32-bit IEEE754 floating point values natively.

**NOTE: Refer to Binary Global Parameter Access Note at end of Binary Host Interface section for details.**

#### *Transmit Packet*

| Data Field | Description |
|---|---|
| Byte 0 | Header ID (0x00) |
| Byte 1 | Packet ID (0x91) |
| Byte 2 | Conversion Code |
| Byte 3 | Poke Word Count |
| Long 0 | Poke Address |
| Long 1 | Poke Data 0 |
| Long 2 | Poke Data 1 |
| : | |

| Data Field | Description |
|---|---|
| Long N | Poke Data (Count - 1) |

### *Receive Packet*

None.

### *Conversion Codes*

| Code | Source | Destination |
|---|---|---|
| 0x00 | LONG | LONG |
| 0x01 | IEEE32 | FP64 |
| 0x02 | IEEE32 | FP32 |

## Usage Example

This example pokes data into three words, starting at poke address 0x405000.

**NOTE: Addresses shown are for example only. Addresses will vary from controller to controller, depending on system memory allocation.**

| Fields: | Header | Address | Data0 | Data1 | Data2 |
|---|---|---|---|---|---|
| Output: | 00 91 00 03 | 00 50 40 00 | 10 11 12 13 | 20 21 22 23 | 30 31 32 33 |

Data poked into addresses:

- 0x405000: 0x13121110
- 0x405001: 0x23222120
- 0x405002: 0x33323130

# *Binary Address Command*

A binary address command consists of a four-byte header containing a program number and a parameter code. The command returns the header followed by the base address of the parameter type in question. If the returned address is zero, no parameters of that type have been allocated in the given program.

Peeking at the returned address will return the number of variables dimensioned for the requested type. In the case of numeric variables (DV, SV, LV), the count will be followed by the actual numeric data. For arrays (DA, SA, LA), the count will be followed by the addresses of the individual arrays. These addresses point to storage areas as if they were normal numeric variables of the same type (count followed by data).

### *Transmit Packet*

| Data Field | Description |
|---|---|
| Byte 0 | Header ID (0x00) |
| Byte 1 | Packet ID (0x92) |
| Byte 2 | Program Number |
| Byte 3 | Parameter Code |

### *Receive Packet*

| Data Field | Description |
|---|---|
| Byte 0 | Header ID (0x00) |
| Byte 1 | Packet ID (0x92) |
| Byte 2 | Program Number |
| Byte 3 | Parameter Code |
| Long 0 | Parameter Address |

### *Parameter Codes*

| Code | Mnemonic | Description |
|---|---|---|
| 0x00 | DV | Double Variables |
| 0x01 | DA | Double Arrays |
| 0x02 | SV | Single Variables |
| 0x03 | SA | Single Arrays |
| 0x04 | LV | Long Variables |
| 0x05 | LA | Long Arrays |
| 0x06 | $V | String Variables |
| 0x07 | $A | String Arrays |

## Usage Example

This example requests the starting address of the Single Variable information for Program 5.

**NOTE: Addresses shown are for example only. Addresses will vary from controller to controller, depending on system memory allocation.**

| Fields: | Header | Parameter Address |
|---|---|---|
| Output: | 00 92 05 02 | |
| Input: | 00 92 05 02 | 00 80 40 00 |

Starting address of the Single Variable information for Program 5:

Address: 0x408000

# *Binary Parameter Address Command*

A binary parameter address command consists of a four-byte header containing a parameter index. The command returns the header followed by the address of the parameter. If the returned address is zero, the parameter index was invalid.

***Transmit Packet***

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x93) |
| Byte 2-3 | WORD | Parameter Index |

***Receive Packet***

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x93) |
| Byte 2-3 | WORD | Parameter Index |
| Long 0 | LONG | Parameter Address |

## Usage Example

This example requests the address of the axis 0 current position parameter.

**NOTE: Addresses shown are for example only. Addresses will vary from controller to controller, depending on system memory allocation.**

| Fields: | Header | Parameter Address |
|---|---|---|
| Output: | 00 93 00 30 | |
| Input: | 00 93 00 30 | 31 50 40 00 |

Current Position Parameter Address:

AXIS0: 0x405031

# Binary Mask Command

A binary mask command consists of a four-byte header followed by an address and two bit masks to be combined with the data at that address. There is no information returned from this command. The address must point to a long integer storage area. The NAND mask is used to clear bits and the OR mask is used to set bits (OR mask is dominant). The data is modified as follows:

data = ( data AND NOT nandmask ) OR ormask

### *Transmit Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID (0x00) |
| Byte 1 | BYTE | Packet ID (0x94) |
| Byte 2 | BYTE | Reserved (0x00) |
| Byte 3 | BYTE | Reserved (0x00) |
| Long 0 | BYTE | Data Address |
| Long 1 | BYTE | NAND Mask |
| Long 2 | BYTE | OR Mask |

### *Receive Packet*

None.

## Usage Example

This example uses the Binary Mask Command to clear all of the Opto-isolated Outputs and then set Output 32. The data address for Opto-isolated Outputs Parameter P4097 is assumed to have been previously returned using the Binary Parameter Address Command on the previous page.

**NOTE: Addresses shown are for example only. Addresses will vary from controller to controller, depending on system memory allocation.**

| Fields: | Header | Parameter Address | NAND Mask | OR Mask |
|---|---|---|---|---|
| Output: | 00 94 00 00 | 43 60 40 00 | FF FF FF FF | 01 00 00 00 |

Opto-isolated Output Parameter P4097 Modified Data at address:

0x406043: 0x00000001

# Binary Parameter Mask Command

A binary parameter mask command consists of a four-byte header followed by two bit masks to be combined with a system parameter. There is no information returned from this command. The parameter index in the header must be a long integer. The NAND mask is used to clear bits and the OR mask is used to set bits (OR mask is dominant). The data is modified as follows:

data = ( data AND NOT nandmask ) OR ormask

### *Transmit Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Byte 0 | BYTE | Header ID ( 0x00 ) |
| Byte 1 | BYTE | Packet ID ( 0x95 ) |
| Byte 2-3 | WORD | Parameter Index |
| Long 0 | LONG | NAND Mask |
| Long 1 | LONG | OR Mask |

### *Receive Packet*

None.

## Usage Example

This example uses the Binary Parameter Mask Command to clear all of the Opto-isolated Outputs and then set Output 32.

| Fields: | Header | NAND Mask | OR Mask |
|---|---|---|---|
| Output: | 00 95 01 10 | FF FF FF FF | 01 00 00 00 |

Opto-isolated Output Parameter P4097 Modified Data:

P4097: 0x00000001

# Binary Move Command

A binary move consists of a variable length header followed by a number of four-byte data fields. The bit-mapped information in the header determines the number of data fields and their content. All data fields are sent low order byte first.

### *Binary Move Packet*

| Data Field | Data Type | Description |
|---|---|---|
| Head 00 | BYTE | Header ID (0x04) |
| Head 01 | BYTE | Header Code 0 |

| Data Field | Data Type | Description |
|---|---|---|
| Head 02 | BYTE | Header Code 1 |
| Head 03 | BYTE | Header Code 2 |
| Head 04 | BYTE | Header Code 3 |
| Head 05 | BYTE | Header Code 4 |
| Head 06 | BYTE | Header Code 5 |
| Head 07 | BYTE | Header Code 6 |
| Head 08 | BYTE | Header Code 7 |
| Data 00 | IEEE32 | Master VEL |
| Data 01 | IEEE32 | Master FVEL |
| Data 02 | IEEE32 | Master ACC/DEC |
| Data 03 | LONG* | Slave 0 Target or NURB/Spline control point |
| Data 04 | LONG* | Slave 1 Target or NURB/Spline control point |
| Data 05 | LONG* | Slave 2 Target or NURB/Spline control point |
| Data 06 | LONG* | Slave 3 Target or NURB/Spline control point |
| Data 07 | LONG* | Slave 4 Target or NURB/Spline control point |
| Data 08 | LONG* | Slave 5 Target or NURB/Spline control point |
| Data 09 | LONG* | Slave 6 Target or NURB/Spline control point |
| Data 10 | LONG* | Slave 7 Target or NURB/Spline control point |
| Data 11 | LONG* | Slave 8 Target or NURB/Spline control point |
| Data 12 | LONG* | Slave 9 Target or NURB/Spline control point |
| Data 13 | LONG* | Slave 10 Target or NURB/Spline control point |
| Data 14 | LONG* | Slave 11 Target or NURB/Spline control point |
| Data 15 | LONG* | Slave 12 Target or NURB/Spline control point |
| Data 16 | LONG* | Slave 13 Target or NURB/Spline control point |
| Data 17 | LONG* | Slave 14 Target or NURB/Spline control point |
| Data 18 | LONG* | Slave 15 Target or NURB/Spline control point |

| Data Field | Data Type | Description |
|---|---|---|
| Data 19 | LONG* | Primary Center |
| Data 20 | LONG* | Secondary Center |
| Data 21 | IEEE32 | Primary Scaling or NURB/Spline Knot |
| Data 22 | IEEE32 | Secondary Scaling or NURB Weight |

* These fields are in IEEE32 format if bit 2 of header code 3 is set.

## Header Code 0

There are two versions defined for Header Code 0 based on Secondary Master Flag Bit Index 5, Enable Rapid Move Modes.

The default-disabled mode for this flag (Secondary Master Flag Bit Index 5 cleared) uses the following Header Code 0 definition.

***Enable Rapid Move Modes Flag Disabled—Default Cleared Value:***

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | FVEL Lockout | Forces FVEL to zero for this move |
| Bit 1 | FOV Lockout | Forces FOV to 1.0 for this move |
| Bit 2 | STP Ramp Activate | Sets STP equal to DEC, else STP 0 |
| Bit 3 | Code 3 Present | Header contains "Header Code 3" |
| Bit 4 | Velocity Data Present | Packet contains master VEL |
| Bit 5 | Acceleration Data Present | Packet contains master ACC/DEC |
| Bit 6 | Counter Dir | Count down if set, else up |
| Bit 7 | Counter Mode | Master move counter enable |

The enabled mode for this flag (Secondary Master Flag Bit Index 5 Set) uses the following Header Code 0 definition. The Move Modes for this header code are defined following the header code definitions.

***Enable Rapid Move Modes Flag Enabled—Set Value:***

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Move Mode Bit 1 | Selects the move mode for this move along with Header Code 0 Bit 2. |

| Data Field | Data Type | Description |
|---|---|---|
| Bit 1 | FOV/ROV Lockout | Forces FOV or ROV to 1.0 for this move |
| Bit 2 | Move Mode Bit 0 | Selects the move mode for this move along with Header Code Bit 0. |
| Bit 3 | Code 3 Present | Header contains "Header Code 3" |
| Bit 4 | Velocity Data Present | Packet contains master VEL |
| Bit 5 | Acceleration Data Present | Packet contains master ACC/DEC |
| Bit 6 | Counter Dir | Count down if set, else up |
| Bit 7 | Counter Mode | Master move counter enable |

### *Header Code 1*

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Master Bit 0 | Master for this move packet |
| Bit 1 | Master Bit 1 | |
| Bit 2 | Master Bit 2 | |
| Bit 3 | Interrupt Select | Interrupt host when move starts |
| Bit 4 | Arc Direction | CCW if set, else CW |
| Bit 5 | Arc Mode | Packet contains center points or Spline Knot present |
| Bit 6 | Arc Plane Bit 0 | Primary and secondary axis or NURB Mode |
| Bit 7 | Arc Plane Bit 1 | For binary arc move commands or SPLINE Mode |

### *Header Code 2*

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Slave 0 Present | Slave target positions to be contained in this move packet |
| Bit 1 | Slave 1 Present | |

| Data Field | Data Type | Description |
|---|---|---|
| Bit 2 | Slave 2 Present | |
| Bit 3 | Slave 3 Present | |
| Bit 4 | Slave 4 Present | |
| Bit 5 | Slave 5 Present | |
| Bit 6 | Slave 6 Present | |
| Bit 7 | Slave 7 Present | |

### *Header Code 3*

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Incremental Target | Target positions are incremental |
| Bit 1 | Incremental Center | Center points are incremental |
| Bit 2 | Floating Point Data | Targets and centers are IEEE32 |
| Bit 3 | Arc Radius Scaling | Packet contains radius scaling / NURB/Spline |
| Bit 4 | FVEL Data Present | Packet contains master FVEL |
| Bit 5 | Block Skip Check | Sets the master Block Skip Check |
| Bit 6 | NURB or Spline | Move data packet for NURB or Spline Interpolation |
| Bit 7 | Extended Codes | Extended codes 4,5,6 and 7 are present. This bit should be set if DBCB is used |

### *Header Code 4*

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Reserved | Reserved |
| Bit 1 | | |
| Bit 2 | | |
| Bit 3 | Master Bit 3 | Master for this move packet |

| | | |
|---|---|---|
| Bit 4 | Reserved | Reserved |
| Bit 5 | | |
| Bit 6 | | |
| Bit 7 | | |

### *Header Code 5*

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Reserved | Reserved |
| Bit 1 | | |
| Bit 2 | | |
| Bit 3 | | |
| Bit 4 | | |
| Bit 5 | | |
| Bit 6 | | |
| Bit 7 | | |

### *Header Code 6*

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Slave 8 Present | Slave target positions to be contained in this move packet |
| Bit 1 | Slave 9 Present | |
| Bit 2 | Slave 10 Present | |
| Bit 3 | Slave 11 Present | |
| Bit 4 | Slave 12 Present | |
| Bit 5 | Slave 13 Present | |
| Bit 6 | Slave 14 Present | |
| Bit 7 | Slave 15 Present | |

*Header Code 7*

| Data Field | Data Type | Description |
|---|---|---|
| Bit 0 | Reserved | Reserved |
| Bit 1 | | |
| Bit 2 | | |
| Bit 3 | | |
| Bit 4 | | |
| Bit 5 | | |
| Bit 6 | | |
| Bit 7 | | |

The following Move Modes definition applies to Header Code 0 used with the Master Enable Rapid Move Modes flag set.

## Move Modes

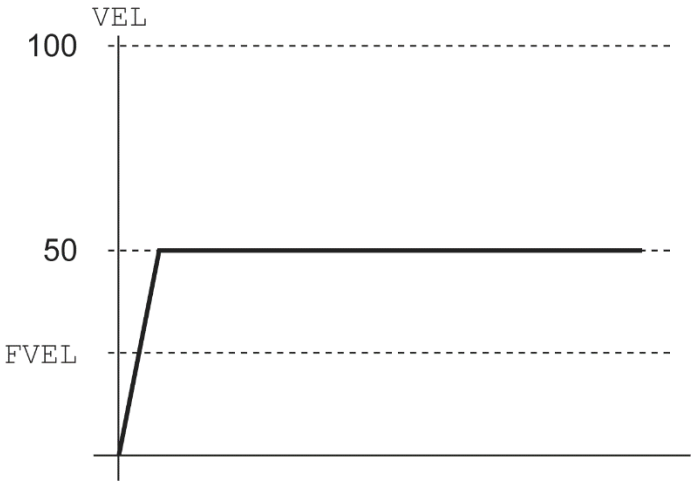
Bits 0 and 2 in Header Code 0 indicate which type of move mode is contained in the binary move packet as follows:

| Bit 1 (Header Code 0 Bit 0) | Bit 0 (Header Code 0 Bit 2) | Move Mode |
|---|---|---|
| 0 | 0 | Move Mode 0 - Feed Continuous |
| 0 | 1 | Move Mode 1 - Feed Cornering |
| 1 | 0 | Move Mode 2 - Feed Stopping |
| 1 | 1 | Move Mode 3 – Rapid |
| Where: 0 = Bit Cleared; 1 = Bit Set | | |

*Example 1*

The following illustrates Move Mode 0—Feed Continuous:

**Move Mode 0—Feed Continuous**

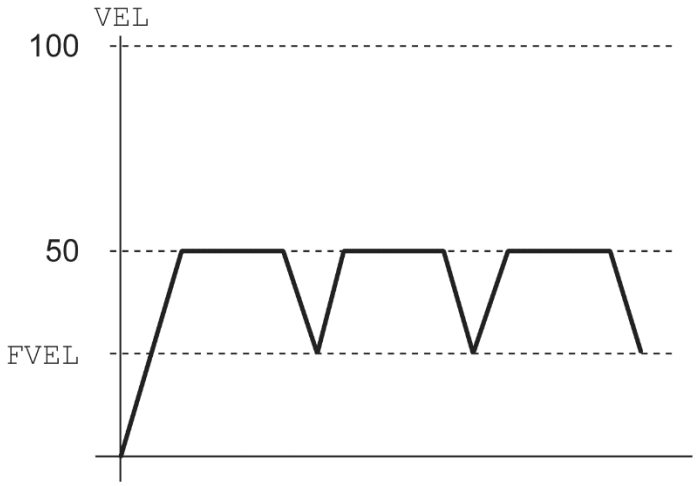



```
VEL = 100.0
FVEL = 20.0
FOV = 0.5
ROV = 0.25
```

***Example 2***

The following illustrates Move Mode 1—Feed Cornering:

**Move Mode 1—Feed Cornering**

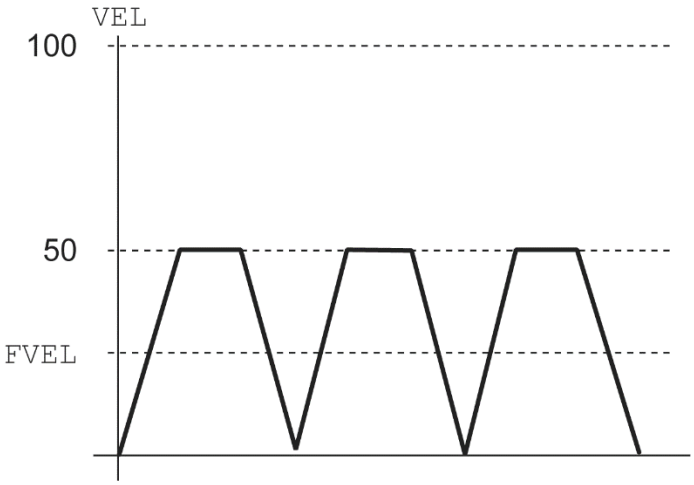



```
VEL = 100.0
FVEL = 20.0
FOV = 0.5
ROV = 0.25
```

***Example 3***

The following illustrates Move Mode 2—Feed Stopping:

**Move Mode 2—Feed Stopping**

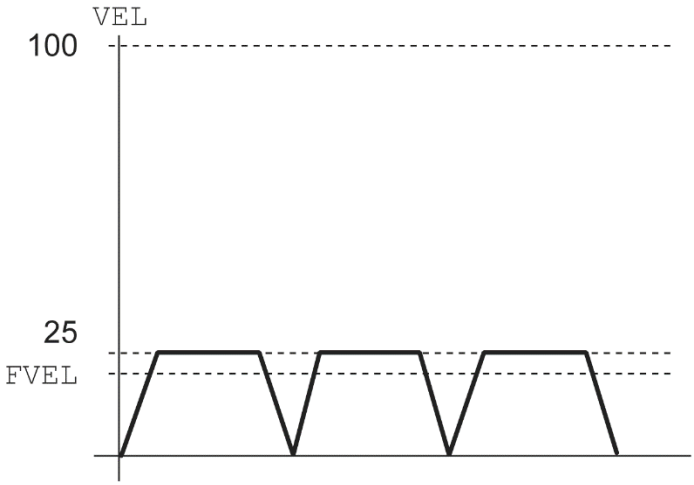



```
VEL = 100.0
FVEL = 20.0
FOV = 0.5
ROV = 0.25
```

***Example 4***

The following illustrates Move Mode 3—Rapid:

**Move Mode 3—Rapid**




```
VEL = 100.0
FVEL = 20.0
FOV = 0.5
ROV = 0.25
```

## Linear Moves

The bits in header code 2 indicate which target positions are contained in the binary move packet. If the "incremental target" bit in header code 3 is set, the targets are relative to the current target positions of the slaves; otherwise, the targets are absolute. The "floating point data" bit in header code 3 indicates that the target data is in IEEE floating point format, otherwise they are long integers.

## Arc Moves

When the "arc mode" bit in header code 1 is set, a circular arc is generated using two of the first three slaves attached to a master. Any slaves that are given a target position, but are not part of the circular interpolation, are executed as normal linear moves. This allows for helical interpolation.

The "arc plane" bits in header code 1 are combined to generate a number from 0 to 3 that defines the primary and secondary axes for the arc as follows:

| Arc Plane | Primary Axis | Secondary Axis |
|---|---|---|
| 0 | Slave 0 | Slave 1 |
| 1 | Slave 1 | Slave 2 |
| 2 | Slave 2 | Slave 0 |
| 3 | Reserved | Reserved |

The "arc direction" bit in header code 1 indicates the direction of the arc relative to the primary and secondary axes. A counter-clockwise arc is defined as an arc from the positive primary axis toward the positive secondary axis.

The radius of the arc will be equal to the distance between the arc target position and the given center point. If the arc target position is equal to the target position of the previous move, a 360-degree path will be generated. The target position of the previous move must lie on the defined arc or the axes will jump to that location before the arc begins.

If the "incremental center" bit in header code 3 is set, the center points are relative to the current target positions of the slaves, otherwise the center points are absolute. The "floating point data" bit in header code 3 indicates that the given center points are in IEEE floating point format, otherwise they are long integers.

### NURB or SPLINE Moves

When the "NURB or Spline" bit in header code 3 (Bit 6) is set, the move data packet includes NURB or Spline curve data. In addition, bit 5 and 6 in header code 1 will differentiate if the data is NURB or Spline. Bit 5 of header code 1 is set when Spline data includes Knots.

The control points for NURB and Spline are sent as DATA3 thru DATA10, similar to the way the normal slave targets are sent. Load the Knot in DATA13 and Weight in DATA14 and set bit 3 in header code 3.

# *Binary SET and CLR*

The immediate setting and clearing of bits can be accomplished with a 3-byte binary command sequence. This sequence is a 1-byte command header followed by a two-byte index value. The index value is sent low order byte first. The command is not queued and the set or clear occurs when the command is first seen by the board.

***Binary SET***

| Data Type | Description |
|---|---|
| Byte 0 | Header ID (0x1C) |
| Byte 1 | Index Byte 0 |
| Byte 2 | Index Byte 1 |
| Byte 3 | 0x00, present for backwards compatibility. |

### *Binary CLR*

| Data Type | Description |
|---|---|
| Byte 0 | Header ID (0x1D) |
| Byte 1 | Index Byte 0 |
| Byte 2 | Index Byte 1 |
| Byte 3 | 0x00, present for backwards compatibility. |

### *Usage Example*

| Binary Output | Description |
|---|---|
| 1C 08 02 | Set bit 520 ( 0x0208 ) |
| 1D 20 00 | Clear bit 32 ( 0x0010 ) |

# *Binary FOV Command*

The immediate setting of feedrate override for any or all axes can be accomplished with an eight-byte binary command sequence. This sequence is a four-byte command header followed by a four-byte FOV value. The command is not queued and the FOV occurs when the command is first seen by the board.

The second byte in the header is a bit mask that determines which masters are affected by the FOV value that follows. The FOV value is an image of an IEEE 32-bit floating-point value, sent low order byte first.

For more than eight masters (not possible on the ACR7000 or IPA) the header bit mask byte 1 should be set to zero and the two optional 16 master header bit mask byte 2 and byte 3 should be filled accordingly.

### *Binary Format*

| Data Type | Description |
|---|---|
| Byte 0 | Header ID (0x07) |
| Byte 1 | Header Bit Mask |
| Byte 2 | 16 Master Header Bit Mask, Part 1 |
| Byte 3 | 16 Master Header Bit Mask, Part 2 |
| Byte 4 | FOV Byte 0 |
| Byte 5 | FOV Byte 1 |
| Byte 6 | FOV Byte 2 |

| Data Type | Description |
|---|---|
| Byte 7 | FOV Byte 3 |

### *Header Bit Mask*

| Data Type | Description |
|---|---|
| Bit 0 | Master 0 Affected |
| Bit 1 | Master 1 Affected |
| Bit 2 | Master 2 Affected |
| Bit 3 | Master 3 Affected |
| Bit 4 | Master 4 Affected |
| Bit 5 | Master 5 Affected |
| Bit 6 | Master 6 Affected |
| Bit 7 | Master 7 Affected |

NOTE: Masters affected by the FOV contained in this command.

### *16 Master Header Bit Mask, Part 1*

| Data Type | Description |
|---|---|
| Bit 0 | Master 0 Affected |
| Bit 1 | Master 1 Affected |
| Bit 2 | Master 2 Affected |
| Bit 3 | Master 3 Affected |
| Bit 4 | Master 4 Affected |
| Bit 5 | Master 5 Affected |
| Bit 6 | Master 6 Affected |
| Bit 7 | Master 7 Affected |

NOTE: Masters affected by the FOV contained in this command.

*16 Master Header Bit Mask, Part 2*

| Data Type | Description |
|---|---|
| Bit 8 | Master 8 Affected |
| Bit 9 | Master 9 Affected |
| Bit 10 | Master 10 Affected |
| Bit 11 | Master 11 Affected |
| Bit 12 | Master 12 Affected |
| Bit 13 | Master 13 Affected |
| Bit 14 | Master 14 Affected |
| Bit 15 | Master 15 Affected |

NOTE: Masters affected by the FOV contained in this command.

*Usage Example*

This example uses the following IEEE conversions:

0.500 = 3F000000

0.123 = 3DFBE76D

| Binary Output | Description |
|---|---|
| 07 08 00 00 00 00 00 3F | Set Master 3 FOV to 0.5 |
| 07 05 00 00 6D E7 FB 3D | Set Master 0 and Master 2 FOV to 0.123 |

# *Binary ROV Command*

The immediate setting of rapid feedrate override for any or all axes can be accomplished with an eight-byte binary command sequence. This sequence is a four-byte command header followed by a four-byte ROV value. The command is not queued and the ROV occurs when the command is first seen by the board.

The second byte in the header is a bit mask that determines which masters are affected by the ROV value that follows. The ROV value is an image of an IEEE 32-bit floating-point value, sent low order byte first.

For more than eight masters (not possible on the ACR7000 or IPA) the header bit mask byte 1 should be set to zero and the two optional 16 master header bit mask byte 2 and byte 3 should be filled accordingly.

### *Binary Format*

| Data Type | Description |
|---|---|
| Byte 0 | Header ID (0x1F) |
| Byte 1 | Header Bit Mask |
| Byte 2 | 16 Master Header Bit Mask, Part 1 |
| Byte 3 | 16 Master Header Bit Mask, Part 2 |
| Byte 4 | ROV Byte 0 |
| Byte 5 | ROV Byte 1 |
| Byte 6 | ROV Byte 2 |
| Byte 7 | ROV Byte 3 |

### *Header Bit Mask*

| Data Type | Description |
|---|---|
| Bit 0 | Master 0 Affected |
| Bit 1 | Master 1 Affected |
| Bit 2 | Master 2 Affected |
| Bit 3 | Master 3 Affected |
| Bit 4 | Master 4 Affected |
| Bit 5 | Master 5 Affected |
| Bit 6 | Master 6 Affected |
| Bit 7 | Master 7 Affected |

NOTE: Masters affected by the ROV contained in this command.

### *16 Master Header Bit Mask, Part 1*

| Data Type | Description |
|---|---|
| Bit 0 | Master 0 Affected |
| Bit 1 | Master 1 Affected |
| Bit 2 | Master 2 Affected |
| Bit 3 | Master 3 Affected |

| Data Type | Description |
|---|---|
| Bit 4 | Master 4 Affected |
| Bit 5 | Master 5 Affected |
| Bit 6 | Master 6 Affected |
| Bit 7 | Master 7 Affected |
| NOTE: Masters affected by the ROV contained in this command. | |

*16 Master Header Bit Mask, Part 2*

| Data Type | Description |
|---|---|
| Bit 8 | Master 8 Affected |
| Bit 9 | Master 9 Affected |
| Bit 10 | Master 10 Affected |
| Bit 11 | Master 11 Affected |
| Bit 12 | Master 12 Affected |
| Bit 13 | Master 13 Affected |
| Bit 14 | Master 14 Affected |
| Bit 15 | Master 15 Affected |
| NOTE: Masters affected by the ROV contained in this command. | |

*Usage Example*

This example uses the following IEEE conversions:

0.500 = 3F000000

0.123 = 3DFBE76D

| Binary Output | Description |
|---|---|
| 07 08 00 00 00 00 00 3F | Set Master 3 ROV to 0.5 |
| 07 05 00 00 6D E7 FB 3D | Set Master 0 and Master 2 ROV to 0.123 |

# *Application: Binary Global Parameter Access*

Also see Binary Peek and Binary Poke commands.

## Description

Global user variables (see Variable Memory Allocation) can be read and set using the Binary Peek and Poke Command interface.

**NOTE: A maximum word count of 255 can be used when using the Binary Peek and Poke Command interface.**

*Hardware Dependent System Pointer Address*

| Controller | System Pointer Address |
|---|---|
| ACR1200 | 0x400008 |
| ACR1500 | 0xC08008 |
| ACR2000 | 0x400008 |
| ACR8000 | 0x403E08 |
| ACR8010 | 0x403E08 |
| ACR8020 | 0x400009 |

## Reading Global Variables

Peek at the System Pointer Address (see above information) to receive the Global_Variable_Address. If the returned address is zero, there are no dimensioned global variables (see the `DIM` command). If the returned address is other than zero, peek at this address to receive the number of dimensioned global variables.

Read a global variable P(index) using the following addressing scheme for Peek:

Peek_Address = Global_Variable_Address + 1 + ( index * 2 )

Where index = 0 to (number of dimensioned global variables – 1).

Even though global variables are stored on-board as floating point 64 (FP64) numbers, they are returned as IEEE32 numbers (Conversion Code 0x01).

## Setting Global Variables

Peek at the System Pointer Address (see System Pointer Address on previous page) to receive the Global_Variable_Address. If the returned address is zero, there are no dimensioned global variables (see the `DIM` command). If the returned address is other than zero, peek at this address to receive the number of dimensioned global variables.

To prevent corruption of user memory, always verify P(index) is within the dimensioned global variable range before performing a POKE command. Set a global variable P(index) using the following addressing scheme for Poke:

Poke_Address = Global_Variable_Address + 1 + ( index * 2 )

Where index = 0 to (number of dimensioned global variables – 1).

Even though global variables are sent as IEEE32 numbers, they are stored on-board as floating point 64 (FP64) numbers (Conversion Code 0x01).

# CHAPTER 6
# Troubleshooting

# Troubleshooting

When a system does not function as expected, the first thing to do is identify and isolate the problem. When this is accomplished, steps may be taken toward resolution.

## *Problem Isolation*

The first step is to isolate each system component and ensure that each component functions properly when it is run independently. This may require dismantling the system and putting it back together piece by piece to detect the problem. If additional units are available, it may be helpful to exchange them with the system's existing components to help identify the source of the problem.

Determine if the problem is mechanical, electrical or software related and note whether it can be recreated or is repeatable.

Random events may appear to be related, but they are not necessarily contributing factors to the problem.

There may be more than one problem. Isolate and solve one problem at a time.

## *Information Collection*

Document all testing and problem isolation procedures. If the problem is particularly difficult to isolate, be sure to note all occurrences of the problem along with as much specific information as possible. These notes may come in handy later and will also help prevent duplication of testing efforts.

Once the problem is isolated, refer to the table below, Common Problems and Their Solutions. If instructed to contact Parker Technical Assistance, please refer to Technical Assistance for contact information.

## *Troubleshooting Table*

This section includes a table of common problems and their solutions. For locations of the ACR7000 or IPA status LEDs, see their Hardware Installation Guides. This table only lists problem LED indications.

***Common Problems and Their Solutions***

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| Power Status LED | | |
| Power status LED is not on | There is no power to the controller. | Check for disconnected power cable.<br><br>Check for blown fuse.<br><br>Verify the power source meets requirements outlined in the Hardware Installation Guide. |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| Power status LED is steady red | There is inadequate power to the controller. | Verify the power source meets requirements outlined in the Hardware Installation Guide.<br><br>Remove all cables except power.<br><br>If the LED turns green after removing the cables, re-attach the cables one at a time to determine which cable or device is causing the problem.<br><br>If the LED does not turn green, contact Parker Technical Assistance. The unit is likely damaged. |
| Power status LED is alternating red/green | Controller encountered error during boot process. | Contact Parker Technical Assistance. The unit will likely need to be sent back to the factory for repair. |
| Axis Status LED | | |
| Axis status LED is not on | Axis is disabled with no fault (normal state for steppers or servo motors). | Enable drive. |
| Axis status LED is red | Axis fault. Motion on this axis is disabled during a fault state.<br><br>NOTE: The LED illuminates red whenever the drive fault input is activated (drive faulted, no axis cable connected, etc.). | Check for faulted drive. Enable drive. Refer to Operation section of this table.<br><br>Check for axis cable disconnected. |
| Ethernet Status LED | | |
| Ethernet link/activity: yellow LED is off | No Ethernet link is detected. | Check for the correct type of cable and try different Ethernet cable. |
| Ethernet speed: green LED is flashing | Ethernet port is getting intermittent 10 Mbps and 100 Mbps connection. | Verify the Ethernet port on the PC is functioning correctly.<br><br>Verify the ACR controller Ethernet port is functioning correctly with different PC. |
| Ethernet Communication | | |
| Communication Error: 11003 | Wrong computer IP address and/or subnet mask. | Change IP address of computer in Network Settings. |
| Communication Error: 10061 | Same IP address as ACR. | Change IP address of computer in Network Settings. |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| Communication Error: 11010 | Wrong IP address configured in PMM communication window. | Enter in correct ACR IP address in controller's connection window. Check IPA's dial switches for IP address.<br><br>In PMM, click *Ping*. If ping fails, see Connecting to the Controller. |
| Operation | | |
| Drive will not enable | Motion Enable Input is open.<br><br>Verify by checking *Status Panels* → *Bit Status* → *Miscellaneous Control Flags*. | Check if 24 VDC is applied to the Motion Enable Input. |
| | | Bit 5646 indicates status of 24 VDC Motion Enable Input.<br><br>Bit 5645 indicates if Motion Enable Input has been latched.<br><br>If both 5645 and 5646 are set, reapply 24 VDC to Motion Enable Input.<br><br>If only 5645, then SET 5647 to clear 5645 latch. |
| | Encoder signal fault and/or encoder signal is lost.<br><br>Verify by checking *Status Panels* → *Bit Status* → *Encoder Flags*.<br><br>NOTE: Each encoder input has specific flag sets. | For Encoder Signal Fault: Check for incorrect termination. Noise in the system can cause missed and/or false encoder feedback values. |
| | | For Encoder Signal Lost: Check feedback cables. |
| | Amplifier/drive is not powered on. | Check if power is applied to the amplifier/drive.<br><br>Many drives have separate control and motor power. Check fuses/breakers in cabinet and estop/safety circuit. |
| | Excess position error (EXC). Motor has exceeded maximum position error.<br><br>Verify by checking *Status Panels* → *Bit Status* → *Axis Flags* → *Primary Axis Flags*.<br><br>Each axis is indicated by Bit "Not Excess Error." | Increase the EXC setting. |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| | Axis Enable output or Axis Fault input are wrong. | Use Configuration Wizard to select Parker drive used, or select *Other* and set NO or NC for Enable and Fault. |
| | Compax3 with ACR7xC: Drive X4/3 STO Enable input open. Check machine safety circuit is not open. | C3 X4/3 requires 24 VDC to enable. Apply and then enable from ACR7xC controller with DRIVE ON command. |
| | Compax3 with ACR7xC: Drive configuration incorrect for Axis type (External Stepper or External Servo). | Check Compax3 Drive configuration for Axis type: I10T10 Position Control Step/Dir 5V for Stepper, Torque Control Standard Mode for Servo. |
| | PD-xxP with ACR7xC: Drive configuration incorrect for Axis type (External Stepper or External Servo). | Check P series drive configuration Control Mode (Pulse Input for Stepper, Analog Torque for Servo). |
| | PD-xxP with ACR7xC: Inputs need to be pulled high to 24 VDC. | For 71-032478 cable, connect White with Blue Stripe to DC reference, Blue wire to +24 VDC. |
| Drive will not enable (cont.) | Drive faulted<br><br>*PMM → Status Panels → Motion Status Panel.* | For ACR7xC controller, check drive for faults/errors/alarms.<br><br>For ACR7xV or IPA, see *Bit Status → Servo Drive Flags*.<br><br>For ACR7xT, see *Bit Status → Stepper Drive Flags*. |
| Drive will enable, but will not hold torque | Incorrect configuration for motor attached. | Correct the configuration for servo or stepper through the Configuration Wizard.<br><br>For ACR7xC, check drive configuration or stepper drive current settings. |
| | Servo motor running open loop.<br><br>Verify that the drives are running open loop: *Status Panels → Bit Status → Axis Flags → Primary Axis Flags*.<br><br>Each axis is indicated by Bit "Open Servo Loop." | Disable drive and clear the appropriate bit. |
| | Tuning gains are not set correctly. | Use PMM's Servo Tuner (*Tools → ServoTuner → Select Axis*). |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| | Check if the tuning gains are set too low: *Status Panels → Numeric Status → Axis Parameters → Servo Parameters.* | |
| | Torque limit is not set correctly.<br><br>Verify torque limit setting: *Status Panels → Numeric Status → Axis Parameters → Limit Parameters → Plus/Minus Torque Limit.* | Example: TLM X1 indicates torque is limited to 10% of drive motor capacity for axis X. |
| Drive will enable, but loads drops | Brake is released before motor is enabled. | Delay motion after enabling by brake's release time.<br><br>ACR7xV: Set Brake Relay Delay on Enable (P28686 for Axis 0).<br><br>ACR7xC: Brake would be released by drive, check drive settings.<br><br>IPA: Set C41 OUTBD (Output Brake Delay) in milliseconds. |
| Load drops when drive disabled | Motor is disabled before brake engages. | Delay disabling motor by brake's set time.<br><br>ACR7xV: Brake Delay after Disable (P28687 for Axis 0).<br><br>ACR7xC: Brake would be set by drive, check drive settings.<br><br>IPA: Set C103 INUFD (User Fault Delay) in milliseconds. |
| Drive will enable, but motor will not move | Stepper output motion does not occur. ACR controller not configured for stepper output in Configuration Wizard. | Correct configuration for stepper through Configuration Wizard. Tuning gains must remain at default values: PGAIN 0.002441406; IGAIN, ILIMIT, IDELAY, DGAIN, DWIDTH, FFVEL, FFACC, and TLM=0. |
| | Axis encountered limits.<br><br>Verify: *Status Panels → Bit Status → Axis Flags → Quinary Axis Flags.*<br><br>Each axis is indicated by Bit "Positive/Negative End-of-Travel Limit Encountered." | Clear the appropriate Positive/Negative End-of-Travel Limit Encountered Bit.<br><br>Clear any Master Kill All Motion Request Bits and Axis Kill All Motion Request Bits. *PMM → Status Panels → Motion Status → Clear All Kills.* |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| | | Check PMM Configuration Wizard's EOT polarity is correct. |
| | Master Kill All Moves request is active.<br><br>Verify: *Status Panels → Bit Status → Master Flags*.<br><br>Each master is indicated by Bit "Kill All Moves Request." | Clear the appropriate Master Kill All Moves Request Bit.<br><br>Also clear all associated Slave Kill All Motion Request Bits. |
| | Axis Kill All Motion Request is active.<br><br>Verify: *Status Panels → Bit Status → Axis Flags → Quaternary Axis Flags*.<br><br>Each axis is indicated by Bit "ACR9000 Kill All Motion Request." | Clear the appropriate Axis ACR7000 Kill All Motion Request Bit.<br><br>*PMM → Status Panels → Motion Status Panel → Clear All Kills*. |
| | Master in feedhold or feedholding state.<br><br>Verify: *Status Panels → Bit Status → Master Flags*.<br><br>Each master is indicated by Bit "In Feedhold or Feedholding." | Set the appropriate Cycle Start Request Bit. |
| | Slave axis not attached to master.<br><br>Check the configuration by going into the correct PROG level. Type ATTACH. | Correct the configuration through the Configuration Wizard and download the setup code. |
| | Jog or Master Velocity set to zero (no Master Profile).<br><br>Check these parameters by going into the correct PROG level. Type VEL or JOG VEL. | Assign Velocity or Jog Velocity values. Example: VEL 1 or JOG VEL X 1. |
| | Commanded feedrate override set to zero.<br><br>Check the feedrate override by going into the correct PROG level. Type FOV. | Assign the appropriate feedrate override value.<br><br>Example: FOV 1 indicates a master feedrate of 1. |
| | Brake is not released. | Check wiring for brake and brake voltage requirements. Confirm brake voltage is not backwards. |
| | Torque Limit is set to zero (servo). | Assign the appropriate Torque Limit value. |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| | Verify Torque Limit setting by *Status Panels → Numeric Status → Axis Parameters → Limit Parameters → Plus/Minus Torque Limit.* | Example: TLM X1 indicates torque is limited to 10% of drive motor capacity for axis X. |
| Improper operation | Feedback device counts are missing. | Check the feedback cable and connections.<br><br>Check the amplifier to send back correct signals. |
| Servo motors make audible noise | Incorrect tuning gain settings. | Check tuning gain settings. |
| | Incorrect motor commutation. | Verify drive settings, motor connections. |
| | For ACR7xV multi-axis servo, motor power and feedback cables switched. | Confirm motor and feedback cables for each axis. |
| | Brake not released. | If motor has brake or load has brake, check its releasing when drive enables. Disconnect load, power brake directly, motor shaft should be free to turn. |
| | Contamination/dry mechanics. | Remove motor to confirm it is the mechanics. Check gearhead for contamination. Check leadscrew and square rail bearings for lubricant. See manufacturer's recommendations. |
| | Mechanical misalignment. | Check load, bearings, ballscrew for misalignment. Disconnect load from motor to confirm if it is the mechanics. |
| Stepper motors hot | Stepper motors are running at full current when in motion and will be hotter compared to servos. | Step motors can be hot to touch and safety (burn) hazard, shroud/guard motor from human contact. |
| | Wrong motor current set. | Check step motor current settings.<br><br>*PMM → Config Wizard → Drive/Motor.*<br><br>Confirm max case temperature does not exceed motor rating. |
| | Motor set for max current at 100%. | Enable Standby Current Reduction. This will reduce holding torque when not commanding motion. |
| | Motor current too high for ambient temperature. | Lower motor current for steady state case temperature. |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| | Motor current too high for no heat-sink. | Lower motor current for steady state case temperature. |
| Stepper motors make audible noise | Contamination/dry mechanics. | Remove motor to confirm it is the mechanics. Check gearhead for contamination. Check leadscrew and square rail bearings for lubricant. See manufacturer's recommendations. |
| | Mechanical misalignment. | Check load, bearings, ballscrew for misalignment. Disconnect load from motor to confirm if it is the mechanics. |
| | One of the two step motor phases are not connected.<br><br>Check motor resistance when disconnected from drive. A phase and B phase should have similar resistance. | Check motor connectors. Re-terminate motor connection.<br><br>For eight-lead step motors, check that the center taps are connected.<br><br>Replace step motor if resistance check indicates motor short. |
| Motor not moving at correct velocity | Velocity commanded exceeds Master velocity limit VEL LIMIT.<br><br>Axis max velocity is limiting MAXVEL.<br><br>Check the feedrate override by going into the correct PROG level. Type FOV. | Confirm in terminal emulator at motion program. Example: VEL LIMIT.<br><br>Confirm in terminal emulator at motion program. Example: MAXVEL X or AXIS0 MAXVEL.<br><br>Assign the appropriate feedrate override value. Example: FOV 1 indicates a master feedrate of 1. |
| Incorrect torque limit (servo only) | Verify Torque Limit setting by *Status Panels → Numeric Status → Axis Parameters → Limit Parameters → Plus/Minus Torque Limit*. | Assign the appropriate Torque Limit value. Example: TLM X1 indicates torque is limited to 10% of drive motor capacity. |
| "Not valid while in motion" message received | Tried to enable/disable axis while motion is commanded. | Check if axis is making coordinated motion: *Status Panels → Bit Status → Master Flags*. Each master is indicated by Bit "In Motion."<br><br>Check if the axis is making jog motion: *Status Panels → Bit Status → Axis Flags → Primary Axis Flags*. Each axis is indicated by Bit "Jog Active." |
| Motion stops unexpectedly | Axis has encountered soft limits. | Jog off the limit. Clear the appropriate Positive/Negative Soft Limit Encountered Bit. Clear the associated Master Kill All Moves |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| | Verify: *Status Panels* → *Bit Status* → *Axis Flags* → *Quinary Axis Flags*. Each axis is indicated by Bit "Positive/Negative Soft Limit Encountered." | Request Bits. *PMM* → *Status Panels* → *Motion Status* → *Clear All Kills*. |
| | Axis has encountered Positive/Negative End-of-Travel (EOT) Limits.<br><br>Check if EOT limits have been encountered: *Status Panels* → *Bit Status* → *Axis Flags* → *Quinary Axis Flags*. Each axis is indicated by Bit "Positive/Negative EOT Limit Encountered." | Clear the appropriate Positive/Negative End-of-Travel Limit Encountered Bit. Clear any Master Kill All Motion Request Bit and any Axis Kill All Motion Request Bits. |
| I/Os not working | Positive/Negative End-of-Travel (EOT) Limits not working.<br><br>Check the wiring of the limits, referring to their respective hardware installation guides.<br><br>Check if the Positive/Negative EOT Limits are enabled: *Status Panels* → *Bit Status* → *Axis Flags* → *Quinary Axis Flags*. Each axis is indicated by Bit "Positive/Negative EOT Limit Enable." | Check that the associated inputs toggled:<br><br>If using onboard I/O: *Status Panels* → *Bit Status* → *Onboard I/O* → *Onboard I/O* → *Inputs*.<br><br>If using expansion I/O: *Status Panels* → *Bit Status* → *EtherNet/IP Scanner Flags* → *Node xx Digital Inputs*.<br><br>Check Configuration Wizard for inputs assigned as EOTs.<br><br>NOTE: A triggered output will create a contact closure, not a voltage source. |
| I/O not working properly | Incorrect I/O wiring. | Check wiring and external circuitry. Refer to the Hardware Installation Guide. |
| Stepper motor losing position | In vertical applications or applications where a spring/applied force is resisting motion, when the step motor stops, motor current is reduced to standby current, reducing holding torque. | In *PMM* → *Configuration Wizard* → *Motor*, turn off Standby Current Reduction. |
| | Step motor stalling. Motor does not have torque for the load/acceleration/friction or speed is beyond motor's performance. | Reduce load, reduce acceleration (change motion profile to trapezoidal), reduce velocity, use larger motor, review motor sizing. |
| | Mechanical friction may be preventing accurate positioning, though motion would be repeatable. | If you have encoder on motor or load, turn on Position Maintenance mode to enable end of move correction. |

| PROBLEM | CAUSE / VERIFICATION | SOLUTION |
|---|---|---|
| Stepper motor moving at standstill | Confirm other programs are not running or other motion is not being commanded. | See *Status Panels → Motion Status Panel*. Also see *Status Panels → Servo Loop Status*. |
| | If step motor has encoder and excess position error is set in Fault menu in Configuration Wizard, Position Maintenance is turned on. | Within program, at that section, consider turning PM off or adjusting Position Maintenance settings. |
| Servo motor runs away | Analog output / encoder multiplier mismatch.<br><br>Verify analog output by *Status Panels → Numeric Status → Object Parameters → DAC Parameters*.<br><br>Verify encoder input by *Status Panels → Numeric Status → Encoder Parameters → Encoder Parameters → Encoder Position*. | If encoder feedback is correct for appropriate direction, change "DAC GAIN" to the opposite value.<br><br>If encoder feedback is not correct for appropriate direction, change "ENC MULT" to the opposite value. |
| | Unstable servo loop if drive is in velocity mode. | Check the servo amplifier is in analog torque mode. |
| | Unstable servo loop gains. | Servo gains too high. Can be issue if load has decreased significantly. Retune with *PMM → Tools → Servo Tuner*. |
| | Motor miswiring to drive. | Confirm motor and drive wiring and motor configuration. Use servo drive alignment procedure. |
| | Encoder disconnected from motor. Disable drive but have control power (for feedback power), move motor or load and encoder counts should be changing incrementally as it is moved. | Replace motor with spare. Contact local sales channel to set up a repair for the motor or linear motor stage. |
| | Amplifier has an analog input offset. | Correct the analog offset in the amplifier. |
| | Electrical noise. | Reduce electrical noise or move the product away from the noise source. |
| | Improper shielding. | Use shielded, twisted pair wiring for encoder inputs, DAC/stepper outputs, and ADC inputs. |
| | Improper wiring. | Check wiring for shorts, opens, and mis-wired connections. |

# APPENDIX A
# Connecting to the Controller

# Connecting to the Controller

Connect one end of an Ethernet cable to the PC. Connect the other end to one of the controller's two RJ-45 socket connectors. The two RJ-45 sockets can be used interchangeably and have the same IP address.

**Turn on Control Power to the ACR/IPA.**

The ACR7000 has a programmed IP address, whereas the IPA's is set via rotary switches. The default address is shown below. This address can be changed after initial communication is established. The PC will need to be assigned a compatible IP address to communicate with the controller. These steps are detailed below.

### *Default IP Address*

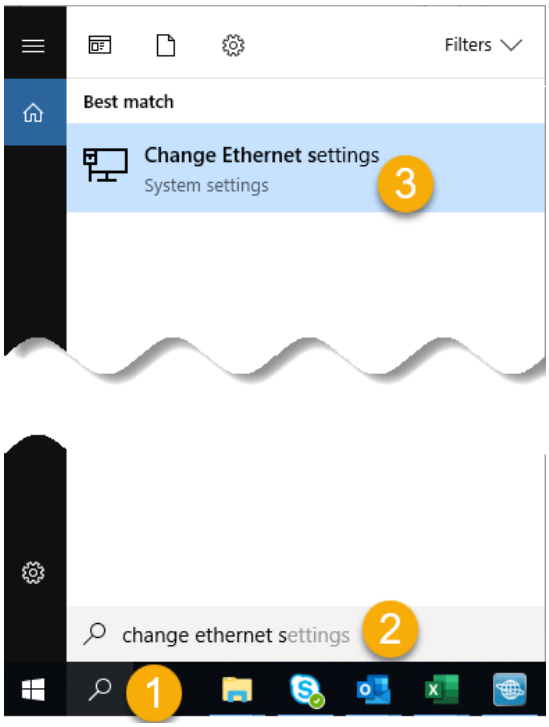
The factory assigns the following to each ACR7000 and IPA.

| IP Address | Subnet Mask (fixed) |
|---|---|
| 192.168.100.01 | 255.255.255.0 |

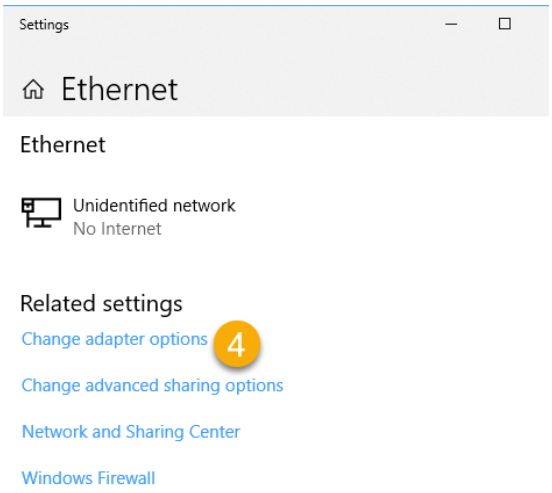
# *Setting the IP Address and Subnet Mask—PC*

Set the IP address and Subnet mask for your PC. These instructions are for Windows 10 users. If you have another Windows version, the steps may vary. Please consult your Network Administrator.

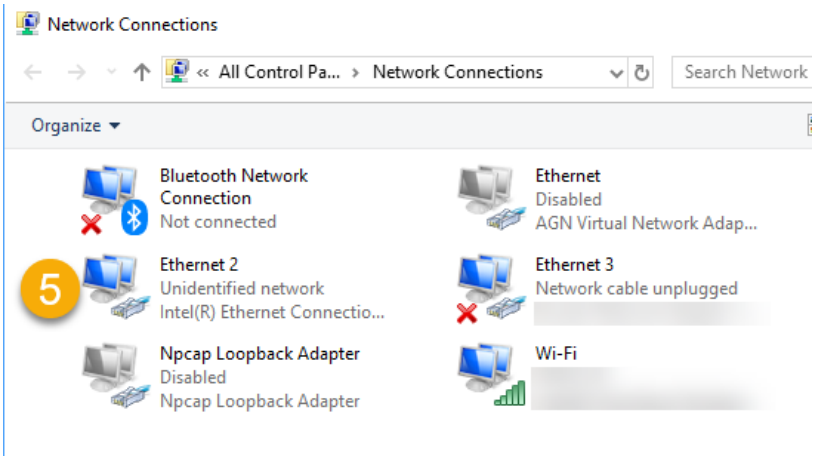
1. Open the Windows Search tool (tap the Windows key).
2. Type *Change Ethernet Settings*.
3. Click *Change Ethernet Settings*.



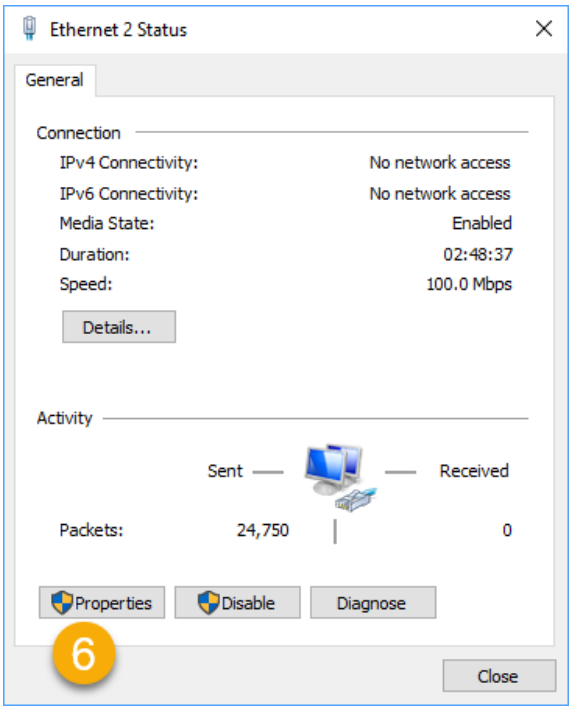

4. Select *Change adapter options*.




5. Select *Ethernet*. More than one Ethernet connection may be displayed. When a cable is plugged into the controller and PC and the controller is powered on the Ethernet connection will show as "Unidentified network".

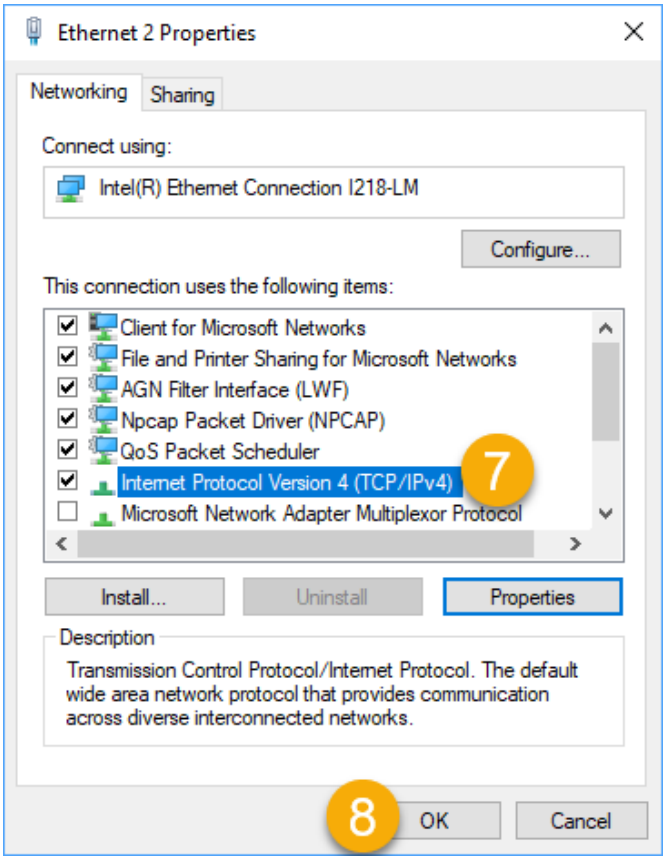



6. Click *Properties*. Administrator rights may be required.



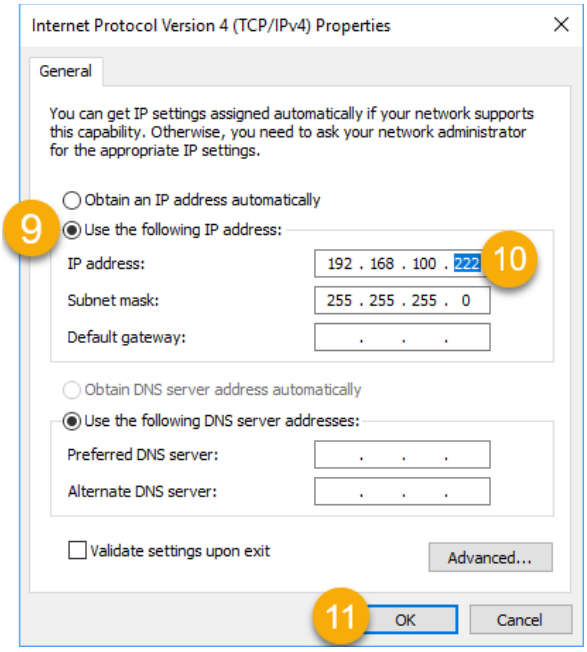

7. Select *Internet Protocol Version 4 (TCP/IPv4)*.

8. Click *Properties*.



9. Click the radio button next to *Use the following IP address*.

10. Enter an IP address with the same first three octets as the default ACR7000 IP address (192.168.100). The last octet of the ACR7000 is "1" by default. Select a different number for the PC—the valid range is 1 to 254. Using 0 or 255 is not valid. In the example the IP address is set to 192.168.100.222. Set the Subnet mask value to 255.255.255.0. Your window should look like the following:




11. Click *OK*. It is now safe to close these windows.

**NOTE: It is good practice to isolate the ACR/IPA and related devices on their own subnet so that their performance is not affected by high volume network traffic.**

# *Verifying the IP Address*

The following verifies that Ethernet is set up correctly.

12. In Parker Motion Manager, the *IP Address* field is the value for the controller.
13. On the Connect screen, click *Connect*.

In the Terminal Emulator, type `VER`. If Ethernet is set up correctly, the Terminal Emulator will report the controller's firmware version information.

# *Troubleshooting*

Having problems connecting? This section covers some easy-to-execute troubleshooting options to fix common connection problems. This is not a procedure, but rather just of list of things to check.

1. Make sure the unit is powered on with Control Power.

2. Make sure the Ethernet cable is connected from the controller to the PC. The LEDs will indicate the hardware connection:


3. If there are no lights, unplug the Ethernet cable and plug it in again. Check the lights again. If they do not turn on, try another Ethernet cable.
4. Check that another device on the network does not have the same IP address.
5. Disconnect the Ethernet cable. Click *Ping* in PMM. Ping should fail. If it does not, connect the PC directly to the controller without going through a switch. Check that the PC's IP address is different than the controller's address. Try clicking *Ping* again and if successful, click *Connect*.
6. Confirm the PC does not have VPN running—close it if it is.
7. Cycle power on the controller and restart PMM. Note that Ethernet communications are available about 20 seconds after cycling power on the controller.
8. The IPA has rotary dial switches to set the IP address. If set to 99, then the IP address is set based on the `IP` command. If unable to connect, change the dial switches to 1 and 0, changing the IP address back to the default of 192.168.100.1. Update PC's IP address per above and try connecting again. After connection, you can check the IPA's intended IP address using the `IP` command.

**NOTE: ACR7000 series products do not have dial switches. If you change the IP address from the default, we recommend labeling the controller with the new IP address on the front or side of the unit.**

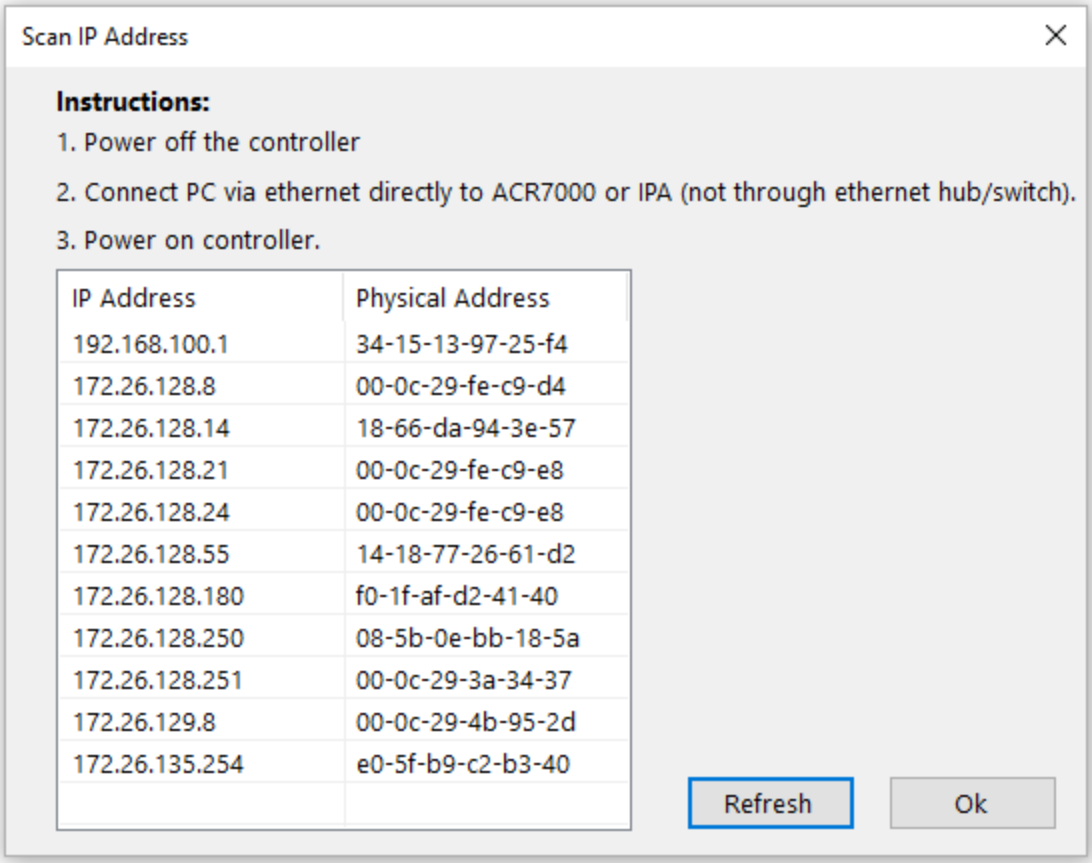
# *Lost the ACR's IP Address?*

There are multiple ways to find the IP address of an ACR whose address has been lost. It is also possible to reset the controller's memory without software, useful if the address cannot be found.

## Finding an ACR with the Scan Tool

This procedure is the simplest and requires no additional software beyond PMM.

1. Open PMM.
2. Make sure the PC is plugged into the ACR/IPA via Ethernet. The connection should be direct—there should be no switch between the ACR/IPA and the PC.
3. Make sure the controller's Ethernet lights are on.
4. In PMM, click *Tools* → *Scan IP Address*.
5. The Scan IP Address dialog will appear. It will list IP addresses available for connection from this PC.




6. The IP address of the ACR controller is usually at or near the top. IP addresses for IPAs are highlighted in orange, but this is not done for ACR7000 series controllers. Click *Refresh* to scan again or click *Ok* to select that IP address for use in the project.
7. Not sure which IP address is the right one? Try turning off Wi-Fi.

## Finding an ACR Using WireShark

This procedure uses WireShark, a popular third-party tool for recording network traffic that is free and open-source. An understanding of WireShark is needed to use this procedure, so it is not recommended for novice users.

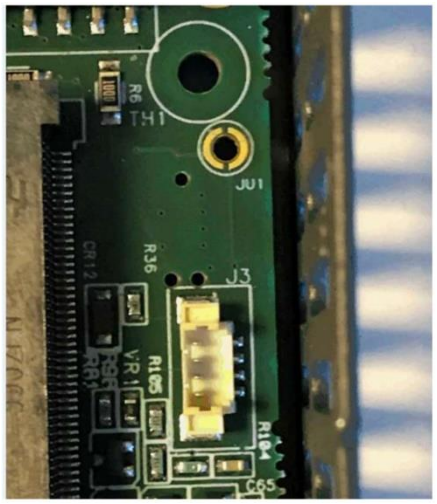
**NOTE: WireShark is a third-party tool and is not maintained or endorsed by Parker. These instructions are provided for the convenience of users who are familiar with and prefer to use WireShark.**

1. Make sure the PC is plugged into the ACR/IPA via Ethernet. The connection should be direct—there should be no switch between the ACR/IPA and the PC.
2. Open WireShark (may need to run as Administrator) and monitor the PC's Ethernet port. Filter for ARP commands.
3. Cycle power on the controller. After it powers on, it will broadcast a gratuitous ARP, which will indicate its IP address.
4. Label the controller.

## Resetting the ACR74T via Hardware

This procedure is a way to recover the ACR74T in the event the IP address cannot be found. The procedure wipes memory on the controller and is equivalent to a `FLASH RES`.

1. Remove power and disconnect all cables.
2. Remove cover taking electronic static discharge precautions (ESD wrist strap, foot strap and ideally a jacket).
3. Locate JU1 jumper on the control board (board with the Ethernet port) and short the two sides together while turning on control power. The pads can be shorted using copper tap, ground strap or a small piece of wire (ideally stranded).




4. Connect with terminal software (such as PuTTY or Hyperterminal) to 192.168.100.1 port 5002. Type `clear -user`, press *Enter* and cycle power. This erases the controller's programs and non-volatile settings back to factory default. The IP address will be reset to default (192.168.100.1). Re-connect with PMM and re-download the project.
5. Use the `IP` command and `ESAVE` to change the IP address. Cycle power for the new address to take effect. Change the PC's IP address if the first three octets have changed.
6. Label the controller.

# APPENDIX B
# Ethernet Basics

# Ethernet Basics

The appendix contains supplemental materials not directly related to any specific ACR series controller discussion.

## *IP Addresses, Subnets and Subnet Masks*

The factory assigns an IP address of 192.168.100.1 and a subnet mask of 255.255.255.0 to each controller. Before adding the controller to your network, assign it an IP address and subnet mask appropriate for your network.

Caution—Talk with your Network Administrators before assigning an IP address or subnet mask to a controller. They can provide you with an available IP address, as well as which subnet mask is appropriate for your particular network configuration.

Isolate the controller and related devices on their own subnet. The high-volume traffic on networks could affect the ACR controller's performance. A closed network restricts the flow of traffic to only the controller and related devices.

The IP address and subnet mask you assign each controller determines the subnet to which the controller belongs. To manage the flow of data across a network, it can be divided into subnets, smaller networks within a network, to provide more efficient delivery of data.

## *IP Addresses*

An IP address is an identifier for a device on a TCP/IP network. Every device connected to the Internet must use a unique IP Address.

The IP address is comprised of a 32-bit binary address that is subdivided into four 8-bit segments known as octets. Because people do not generally think in binary, the address is expressed in dotted decimal format. Each binary octet is converted to a decimal number ranging from 0 to 255, with each octet separated by a decimal point. For example, an IP address in dotted decimal format looks like the following:

192.168.100.120

The address consists of a network ID and a host ID. The network ID acts as a general address, like a zip code. The host ID is the address for a specific device within the network, like a home address. Most IP addresses fall into one of the following address classes:

- Class A range. The first 8 bits are for the network ID; The remaining 24 bits are for the host ID.
- Class B range. The first 16 bits are for the network ID; The remaining 16 bits are for the host ID.
- Class C range. The first 24 bits are for the network ID; The remaining 8 bits are for the host ID.

The number of bits used for the network ID determine how many hosts a given address can support. Class A networks provide a small number of network IDs but a very large number of host IDs and class C networks provide a huge number of network IDs but a small number of host IDs.

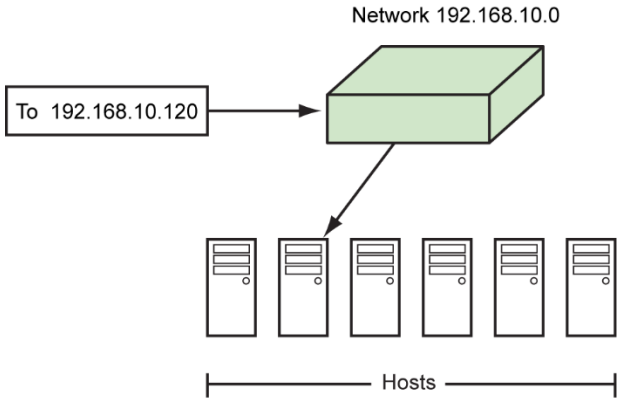
Before a computer or router can send data, it has to identify the network ID through the address class. Each class is assigned a range of numbers.

| Address Class | First octet in dotted decimal format begins with | Excluded from Internet, allowed for Intranet |
|---|---|---|
| A | 0 to 127 | 10.0.0.0 to 10.255.255.255<br>127.0.0.0 to 127.255.255.255 |
| B | 128 to 191 | 172.16.0.0 to 172.31.255.255 |
| C | 192 to 223 | 192.168.0.0 to 192.168.255.255 |

Certain IP addresses have particular meanings and are not assigned to host devices:

- Using zeroes as a host ID signifies the entire network. For example, the IP address of 192.168.0.0 indicates network 192.168 where specific hosts can be found.
- Using 255 in an octet indicates a broadcast, where data is sent to all host devices on a network. For example, the IP Address 192.168.255.255 will broadcast data to all host devices in that network.

Suppose you have 6 computers in a class C network. All share the same network address 192.168.10 in the first three octets. The final octet for each computer is different, and represents the host ID.




Some addresses are reserved for private networks or intranets, where networks are masked or protected from the Internet:

- 10.0.0.0 to 10.255.255.255
- 172.16.0.0 to 172.31.255.255
- 192.168.0.0 to 192.168.255.255

For additional information on private IP addresses, refer to IEEE specification RFC 1918 Address Allocation for Private Internets. You can view it at http://www.faqs.org/rfcs/rfc1918.html.

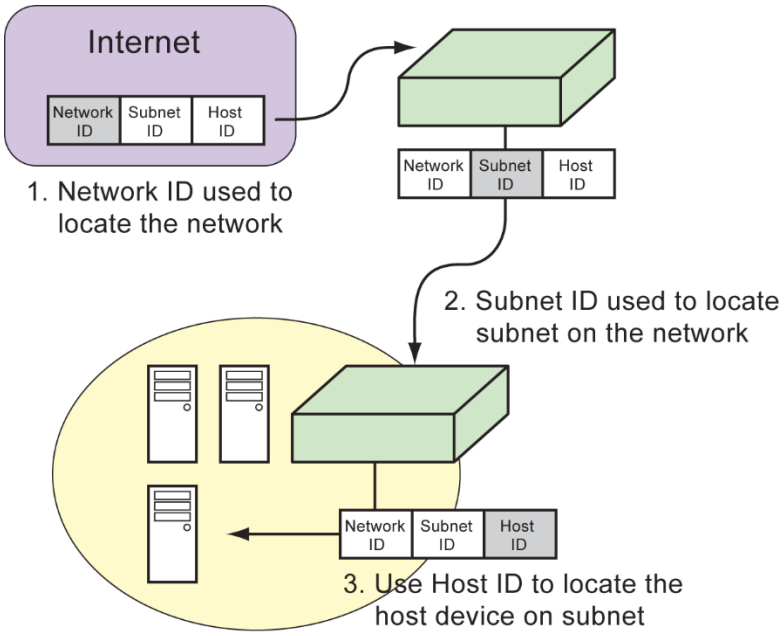
# Subnets

As networks increase in size, it becomes more complex to deliver information. Subnets provide a logical way to break apart network addresses into smaller, more manageable groups. There are additional benefits including more efficient communications between devices and increases to the overall network capacity.

# Subnet IDs

When sending data from one host to another, routers use the network ID (see above) in the IP address to locate the network. On finding the network, the network is searched for the specific host. With a great deal of network traffic this proves cumbersome. Under these circumstances, an IP address does not provide enough information for routers and host devices to efficiently locate a host device.

To provide another level of addressing, some of the host ID is borrowed to create a subnet ID. The subnet ID allows you to logically group devices together (often related to a specific network segment). Once data arrives at the network, the subnet ID allows routers or host devices to locate the appropriate network segment and then the host.

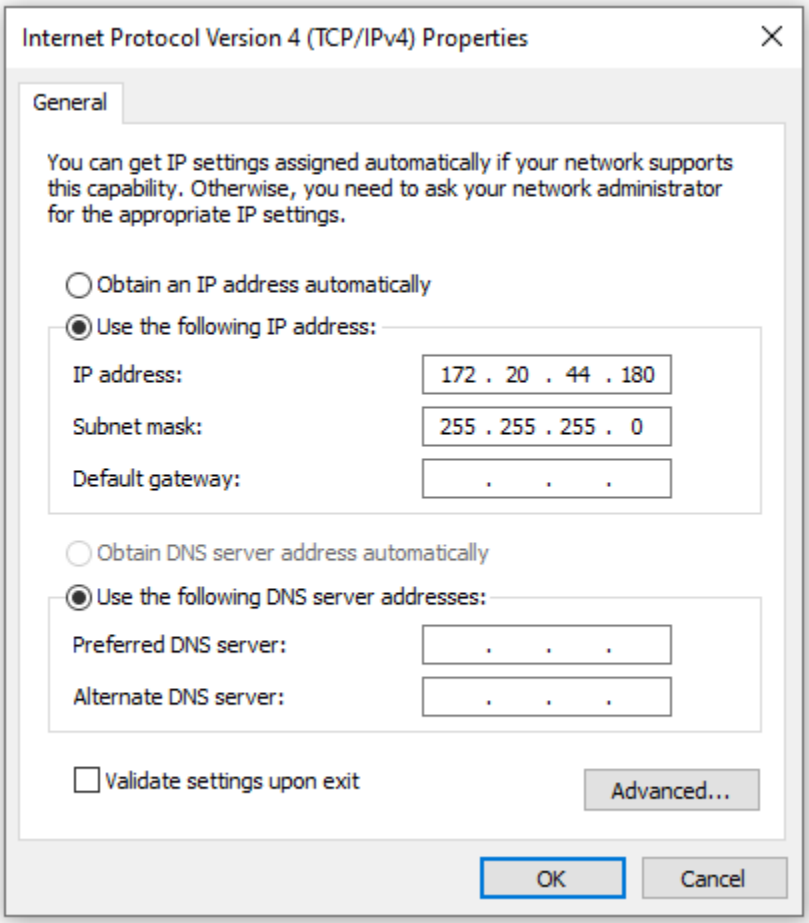



Suppose you have a class C network, comprised of 6 computers. All share the same network ID 192.168 but are divided into two subnets. Three computers use 192.168.10, where 10 is the subnet ID; the remaining three use 192.168.5, where 5 is the subnet ID.

# Subnet Masks

A subnet mask determines how many bits after the network ID are used for the subnet ID. As the subnet ID increases, the number of host IDs available for that network decrease. Similarly, a smaller subnet ID allows you to increase the number of hosts on the network. For simplicity, this discussion only looks at complete octets in dotted decimal format and does not explore converting partial masks from binary to decimal.

What subnet mask to use depends on your network configuration and address class. Where the host ID appears in the IP address, use a zero in the subnet mask. And where the network ID and subnet ID appear, use 255 in the subnet mask. Suppose on network 172.20.0.0 (class B) you have to set up a new computer. You assign it 172.20.44.180 as the IP address. As a class B network, the first two octets are reserved for the network ID. The third octet is reserved for the subnet ID and the last octet is for the host ID. So, using the subnet mask 255.255.255.0 identifies the final octet as the host ID.



# APPENDIX C
# Servo PID Tuning

# Servo PID Tuning

PMM's Servo Tuner helps you tune each analog torque servo drive to determine the gains for your application. This can be done after the Configuration Wizard has been completed and downloaded.

For an introduction to PMM's Servo Tuner, click here.

## *Purpose of Tuning*

The tuning process determines the PID gains (see explanations below) that provide optimum servo performance for your electromechanical system (servo motor with attached load). The gains can be adjusted in the Servo Tuner under Tools in the Explorer. The Servo Tuner graphs motor performance and the gains can then be saved to memory and optionally the project. Different machines may have different gains based on friction, component rigidity/compliance and part-to-part variations.

The tuning gains should be adjusted with a move that is significant enough to excite the mechanical load. A slow, low-speed move cannot excite the mechanical resonances of the load and tuning to such can lead to an unstable system during higher-speed moves.

After tuning, test a move typical for the machine requirements to confirm system performance and further adjust gains if necessary.

## *Test Simple Motion First*

This first procedure is important to verify that the system is functional and stable.

1. In PMM's Servo Tuner, select the axis to tune.
2. Click *Enable Drive*. The motor should energize and be stable at standstill.
3. Click *Move Settings* and change if necessary. Start with a 1 revolution move in 1 second to confirm that the motor starts and stops okay. Distances will be in user units (mm, inches, revs, etc.), so translate 1 motor revolution to those units. Leave *S-Curve* at 25% and *Profile Definition* as *Trapezoidal*.
4. For linear systems, users can check the *Return Move* box. Set *Dwell Before Return* to 1 second.
5. Click *Single Run* to do the move once.
6. Click *Sampling* and change the selection to *Onboard Sampling* to allow *Repeat Run* to work. The *Repeat Run* button runs the move repetitively, uploading and graphing motor performance after each move.

## *Basic Tuning Process*

This procedure is for developing a set of gains that will allow the machine to run at optimum performance.

1. Increase the Excess Position Error setting to 4 revolutions. This can be done in the Configuration Wizard and then downloaded to controller but will be in user units. This can be changed back after tuning.
2. Change the move's *Profile Definition* to *Trapezoidal* with a distance of 1 revolution in 0.10 seconds with *S-Curve* set to *None*.
3. Change *Channel 3* from *Actual Velocity* to *Actual Position* for the specific axis being tuned.

4. Adjust the Proportional and Derivative gains per the diagram below, monitoring the Following Error.

**NOTE: Be ready to disable or power down the system in event of uncontrolled motion.**

*Torque Limit* can be used to limit the DAC output to the servo. The default is 10, which is the full ±10 VDC output range for the servo output to the drive. The drive's torque scaling per volt determines the current command and torque output to which this translates.

In most cases, just the Proportional Gain and Derivative Gain can be iteratively adjusted until overshoot and oscillatory responses at the end of the move are small and following error is minimal (first-order response). While doing this, keep looking at the Following Error after the Secondary Setpoint has stopped moving. Following error during motion is normal but most servo systems only need to settle into the final position quickly.

*Save to Controller*—After axis has been tuned, click Save to Controller. This stores axis tuning gains to non-volatile memory.

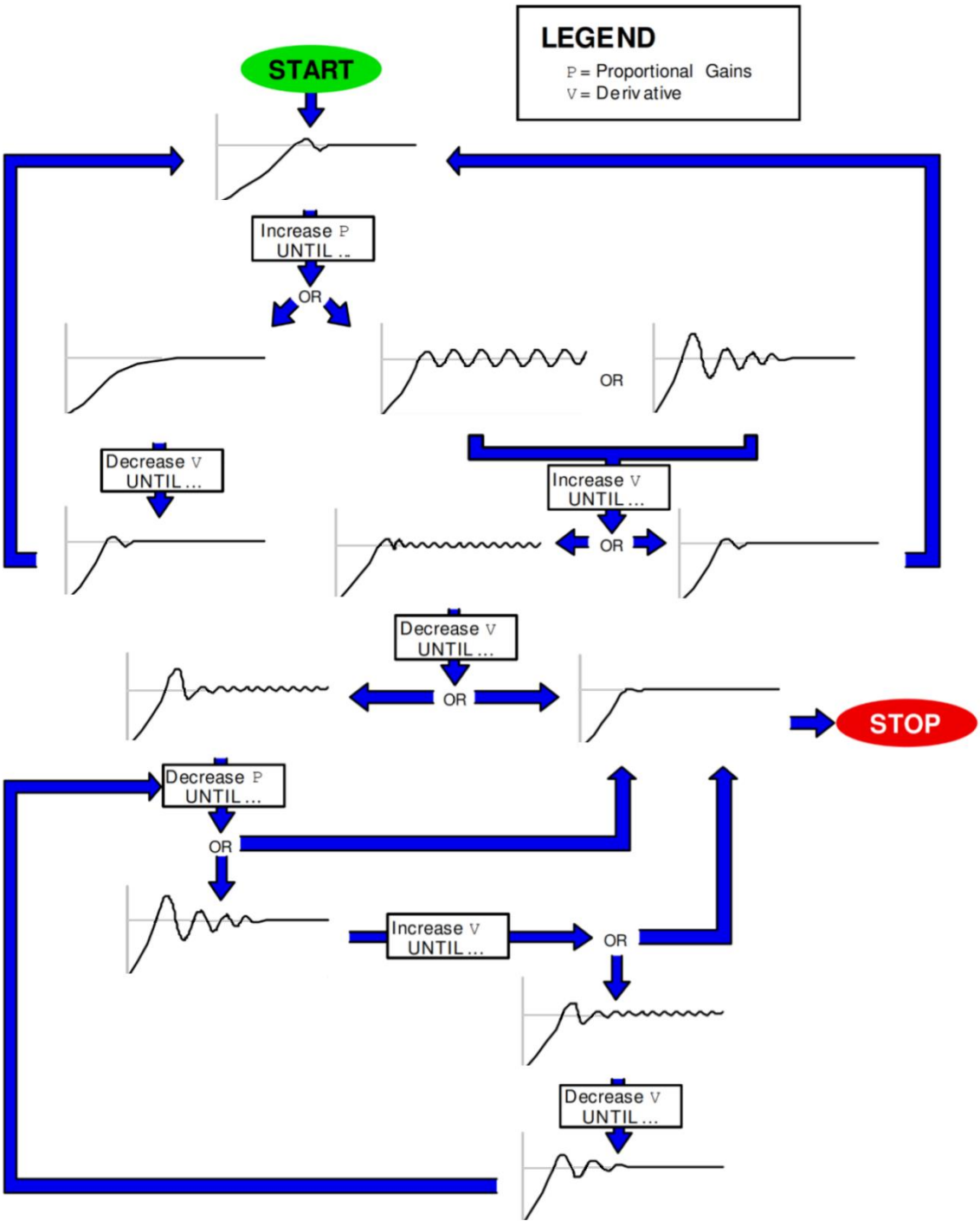
*Save to Project*—This allows you to save the tuning gains as part of the project. This can be handy for creating a backup copy of the project for the machine.

However, if users have multiple machines, different machines' mechanics will vary; even if the machine and parts are the same, alignments will be slightly different, seal/wiper friction will be slightly different, the load may be slightly different, etc. Depending upon the application needs, you may want to re-tune each system starting from the default values instead of a previous machine's tuning gains. In this case you would not save the gains to the project. This situation is uncommon.

You can use *File* → *Save As* to create a backup for this specific machine with the final tuning gains, leaving the initial project at the default tuning gains.

For multi-axis systems, you can select a different axis to tune using the Axis pull-down and repeating the procedure.

**Tuning Process Flowchart**



# *Explanation of Tuning Gains*

### Proportional Gain (PGAIN)

This command modifies the value used in the PID algorithm to control proportional gain. The default gain is 0.0024414 (10 volts at 4096 pulses) for all axes. Units are volt/pulse.

### Derivative Gain (DGAIN)

This command modifies the value used in the PID algorithm to control derivative gain. The default gain is 0.0001 for all axes.

### Integral Gain (IGAIN)

This command modifies the value used in the PID algorithm to control integral gain. Increase the integral gain to counter any steady-state error after the commanded move is complete. The default gain is 0.0 for all axes. Units are volts/second/pulse.

**NOTE: If ILIMIT is zero the integral will remain off, even if the IGAIN value is set to something other than zero.**

### Integral Limit (ILIMIT)

This command modifies the value used by the PID filter to limit the amount of integral term allowed to build up in the loop. The default gain is 0.0 for all axes. Units are volts/second/pulse.

### Integral Delay (IDELAY)

This command modifies the value used in the PID algorithm to control integral delay. The integral delay determines the amount of time, after a move ends, before integration begins. If the value is set to zero, integration is active all the time, even during moves. The default gain is 0.0 for all axes. Units are milliseconds.

### Torque Limit (TLM)

This is the controller's analog max output in volts (max is 10 volts = 100%). Users can use this to limit motor torque—1 volt would be a 10% limit. This would be a limit to the servo amplifier in torque mode. The amplifier setting sets the motor current scaling for 10 volt input to the drive and can be used to correlate the motor torque to DAC output voltage.

The IPA uses the standard ACR tuning gains (PGAIN, DGAIN, etc.), however with the IPA the feedback resolution is normalized to 8000 counts per revolution. With other the ACR controllers the gains are feedback resolution-dependent.

Thus, the IPAs gains will be similar for standard and high-resolution servos. An ACR's gains will be linearly lower for higher resolution servos.

# *Tips and Tricks*

Auto Scale is on by default for vertical scaling in PMM. Note the following error values as you are tuning; the graph may appear larger but that may just be the Auto Scale adjusting based on the data spread.

The position values and following error are in encoder counts.

### Can't reach speed?

It could be a lack of torque from the motor. Try slowing down acceleration ramp.

Your power supply could be pulling down if it does not have the capacitance or is undersized for the current needed. Monitor the Bus Voltage parameter for the ACR7xV or the voltage of the servo drive for ACR7xC.

Check that you are not approaching the max speed of the motor. Check the motor speed-torque rating for the voltage at which it is powered.

Move Settings will automatically calculate based on the distance. If the distance is too short with low acceleration/deceleration, the maximum velocity may not be reached. Increase the distance or the accel/decel.

### Can't accelerate?

Check the DAC output. If this is 10 volts, check that the amplifier's voltage scaling is correct. If it is, you may need to:

- Get a larger motor.
- Decrease the system load.
- Decrease friction.
- Lower the acceleration/deceleration for the moves.

### Derivative Smoothing

Take away humming noise from the servo motor due to `DGAIN`. The smoothing parameter is P12402 for Axis 0.

Axis parameter "DGAIN Smooth" is used to subdue the humming noise in the torque loop due to `DGAIN`. The default value is 0, which means no smoothing is applied. The user may typically change this value from 0 to 5. The `DGAIN` command must be used after changing this parameter to make this change effective.

***Example***

```
REM The DGAIN term will be averaged over 4 samples.
P12402 = 4 : REM Turn on smoothing.
DGAIN X0.0001
```

# *Advanced Tuning Gains*

See PMM online help or Command Reference for further details.

### FF Velocity (FFVEL)

This sets the velocity feedforward for an axis. Used to correct for velocity error while moving.

### FF Acceleration (FFACC)

This sets the acceleration feedforward gain for an axis. Used to correct for acceleration error while accelerating or decelerating.

### Derivative Width (DWIDTH)

Sets the control derivative sampling rate. Default width is 0. Determines how often following error is sampled when calculating derivative term. At 0, sampling occurs at servo interrupt rate (`PERIOD`). For legacy systems only. For modern servo motors, this should be left at 0.0.

### Feedback Velocity (FBVEL)

Sets the velocity feedback gain to amplify the rate of change of feedback. Only for analog feedback systems or dual-feedback loop systems (motor feedback for velocity and load-mounted encoder for position). For standard servo drives, this should be left at 0.0.

### Lowpass Filter (LOPASS)

This command initializes the output filter as a lowpass filter, reducing high-frequency noise that may occur in a system. Setting the cutoff frequency to zero turns off the lowpass filter.

### Notch Filter (NOTCH)

This command sets up the first half of the output filter to act as a notch filter, reducing mechanical resonance that may occur in a system. Setting the center frequency to zero turns off the notch filter.

# APPENDIX D
# PMM Improvements Over ACR-View

# PMM Improvements Over ACR-View

The ACR7000 family is supported by a new software tool, Parker Motion Manager (PMM). Parker Motion Manager combines the best of ACR-View and years of user feedback in a modern, user-friendly and scalable software development environment. PMM is newly built and independent from ACR-View, allowing both software tools to be installed on the same PC if needed.

New features include:

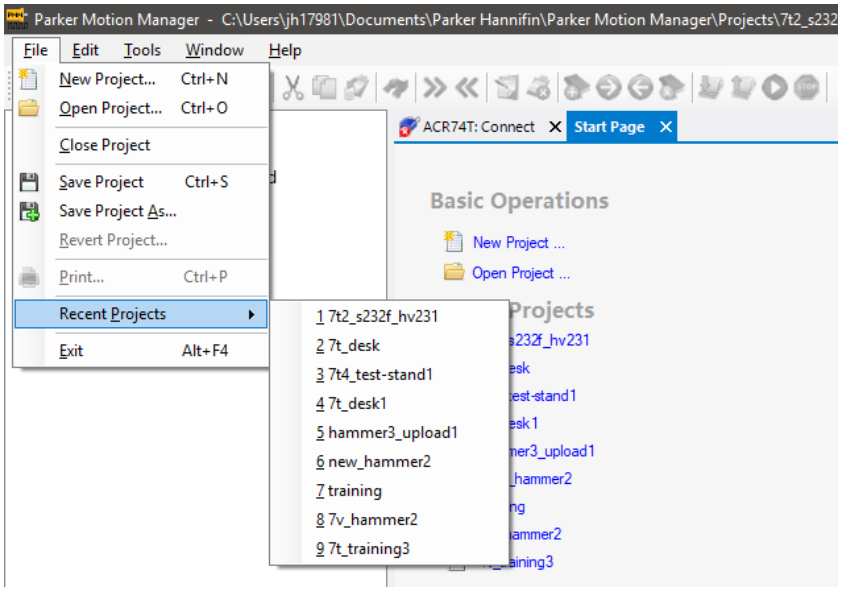
- Streamlined Configuration Wizard
- Product-specific status panels
- Improved Terminal Emulator
- Reimagined scope tools
- Projects now stored as single files for easy sharing and archiving

For ACR programmers new to Parker Motion Manager (PMM), many improvements have been implemented. The layout is similar in terms of having a connection panel, Configuration Wizard, program editor and status panels. This summary is a brief synopsis of the major differences between the two development packages.

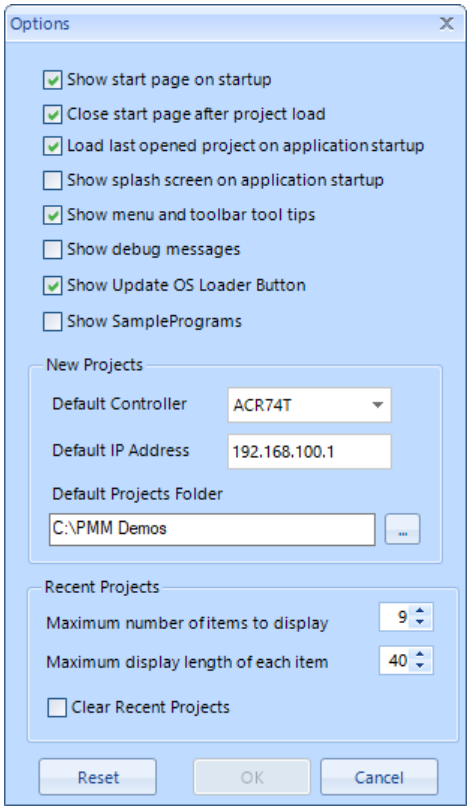
Parker Motion Manager supports all ACR7000 controllers plus the IPA single-axis drive/controller. It does not support the ACR9000, ACR9600 or any prior generation ACR.

**Improvement 1**—Both use ComACRServer6, which is installed with PMM or ACR-View 6. If users have developed PC interfaces to the ACR9000 or IPA, these will work the same with the ACR7000.

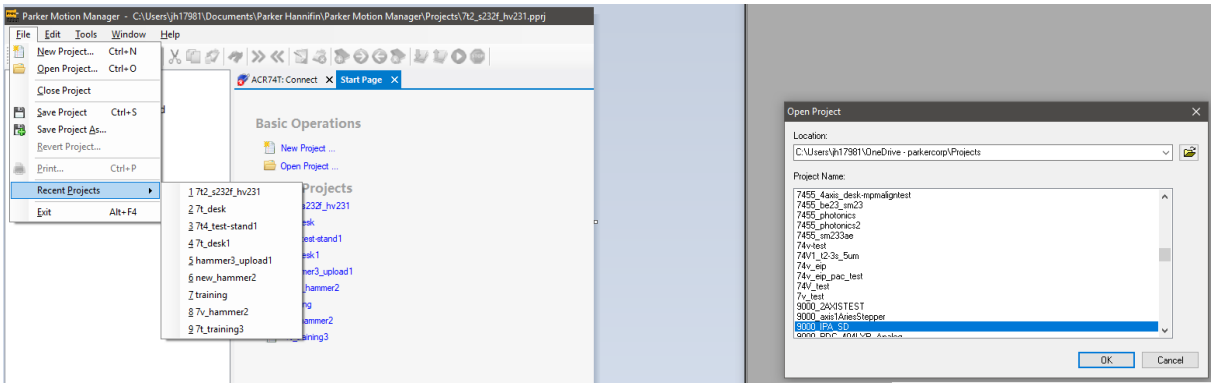
**Improvement 2**—When starting PMM, users will immediately note the *File → Recent Projects* menu option allowing users to quickly jump in and get working. These are both on the optional Start Page and under the *File* menu. The default number of recent projects is nine files, but this can be changed under *Tools → Options*.



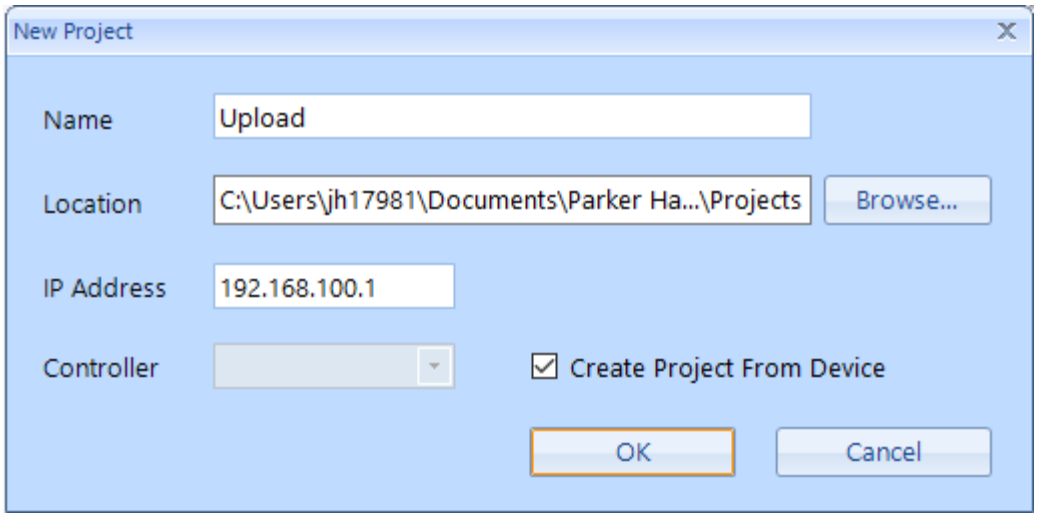

By default, PMM will reopen the last project worked on. This can be changed in the Options menu:




As comparison, ACR-View had a scroll list of projects but they were not sorted by last used. The image below shows PMM on the left and ACR-View on the right.

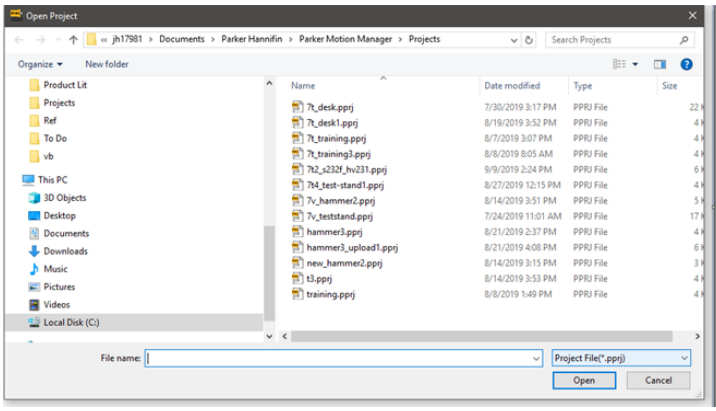


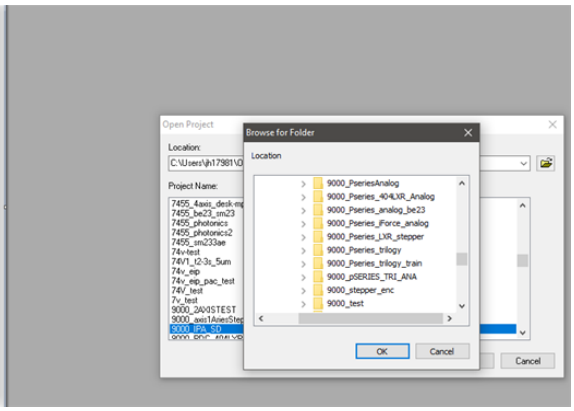

**Improvement 3**—Uploading from an existing controller is now easier from the New Project window (*File → New Project*).




With ACR-View, users would have to create a new project, select the controller and then upload.

**Improvement 4**—With Parker Motion Manager, projects are now stored as a single file (.pprj), making it easier to transfer and share (below, left). In ACR-View, projects were a folder with separate files for the .8k programs, configuration, etc. (below right) that had to be zipped to be shared.



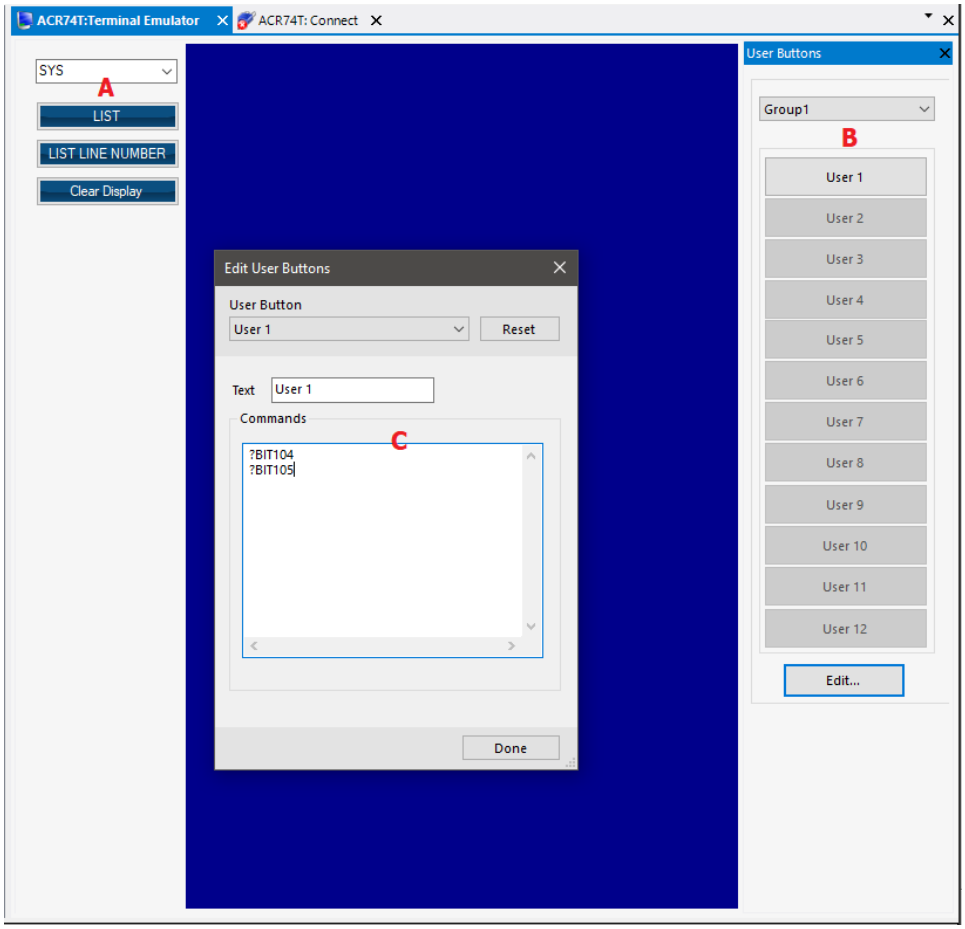




**Improvement 5**—In Parker Motion manager, System Code is generated every time on Finish or before downloading. System Code is not stored within the project. This prevents changes in the Configuration Wizard from failing to update System Code.

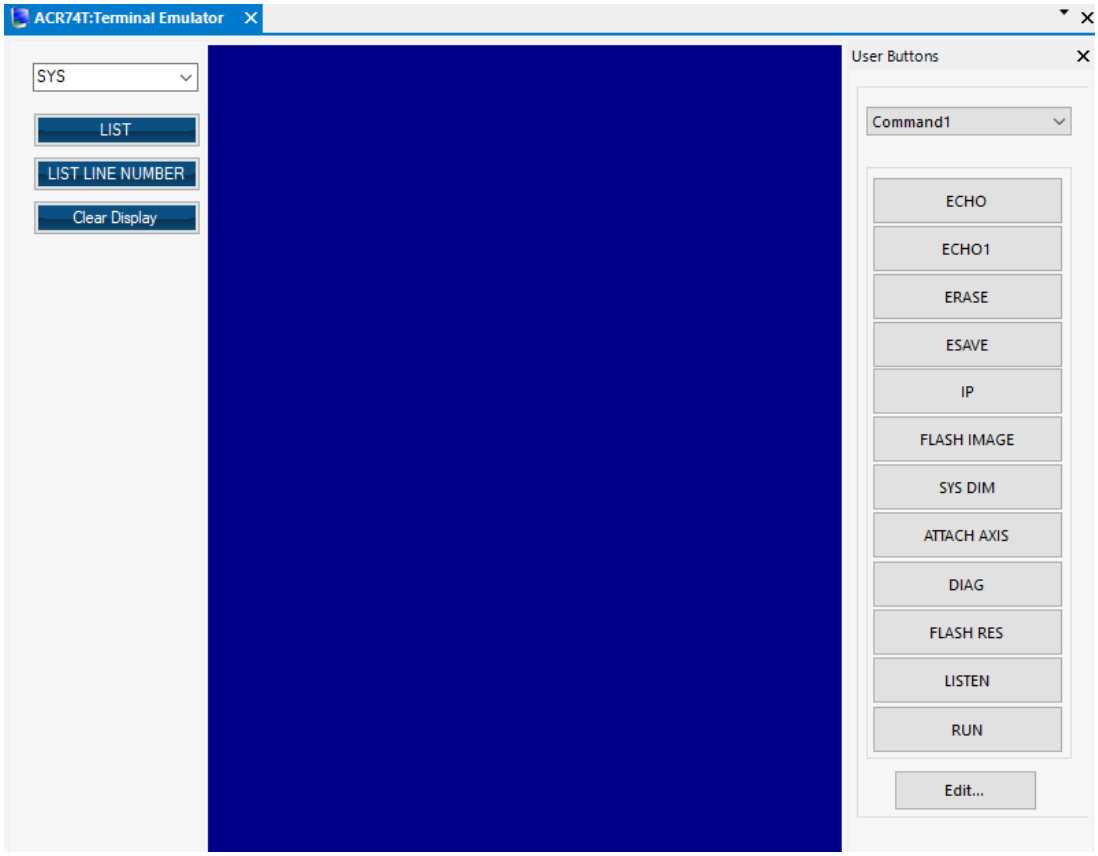
In ACR-View, it was generated as users moved through the Configuration Wizard, which caused issues if users reopened the project and jumped around in the different sections to edit/tune/adjust settings.

**Improvement 6**—Terminal Emulator changes:

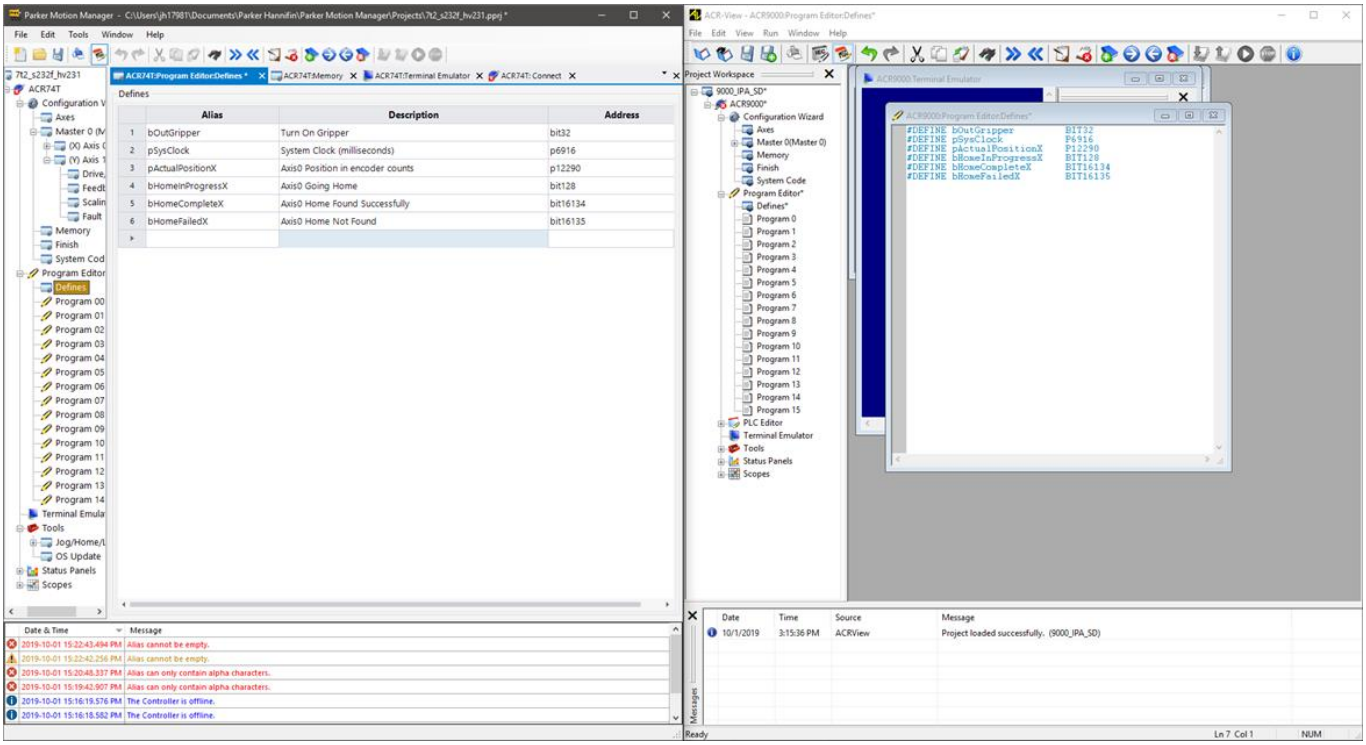
A. Users can now arrow up and arrow down rather than retype commands. This much-loved feature of power coders (from DOS PCs of yester-year) is now available!
B. Dedicated buttons include *LIST* (list program), *LIST LINE NUMBERS* and *Clear Display*. There is also a pull-down to select which program prompt you want to receive commands.
C. Expanded buttons on the right for common commands (5 groups of 12 user buttons).
D. Expanded User Buttons with multiple lines.



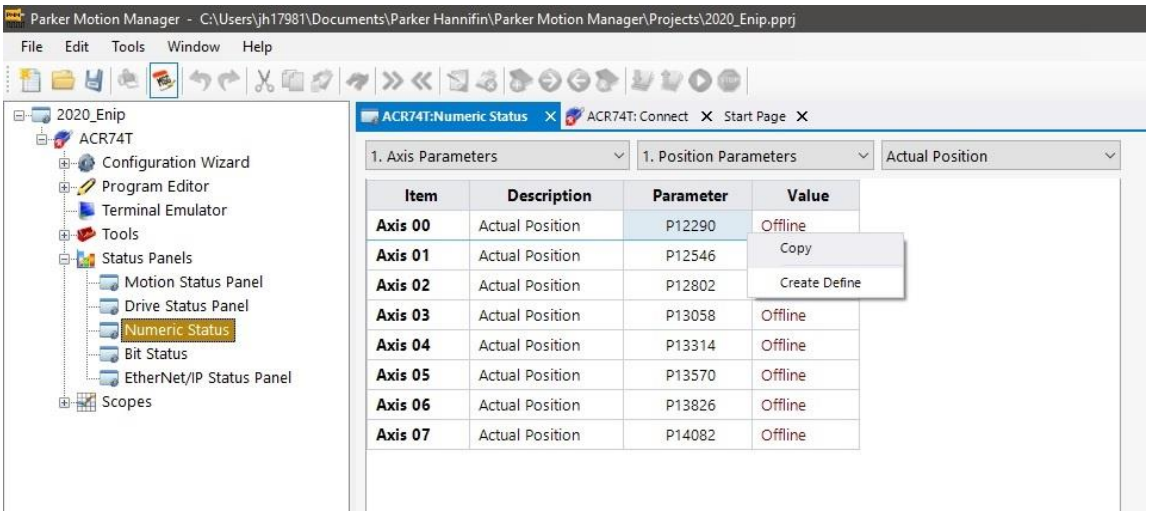

E. The fourth and fifth groups of buttons have been pre-populated with common commands. Power users have full edit access and can re-use these groups, but these predefined buttons help new users.

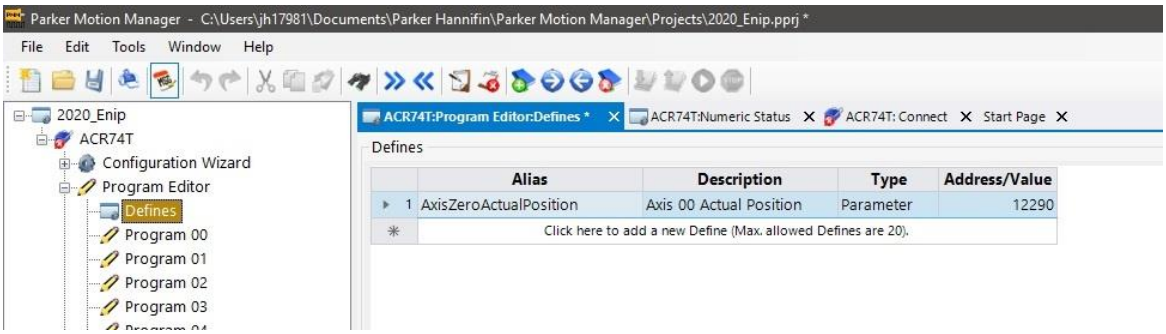



**Improvement 7**—The Defines editor is now in table format, making it easier for programmers to put in a description and name for bits, parameters and constants.

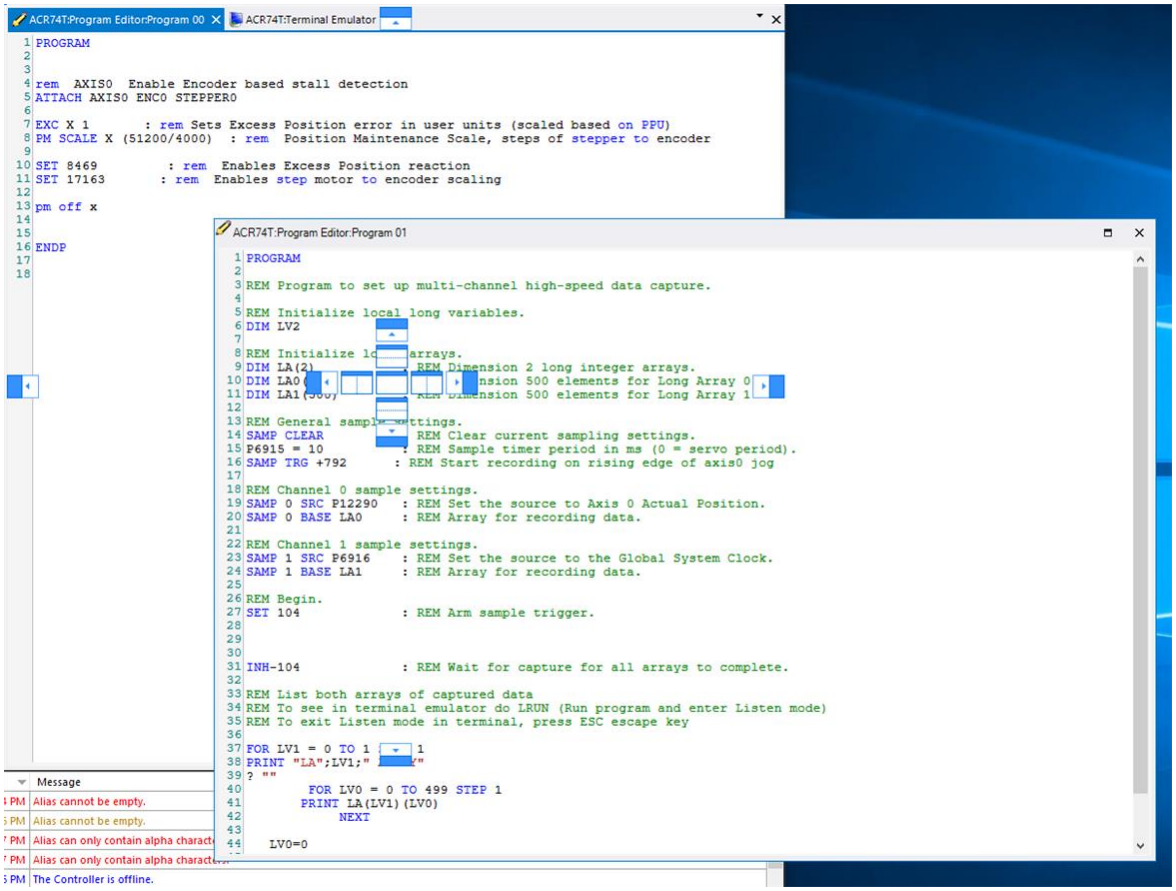


**Improvement 8**—Defines can be created directly from the Numeric Status panels by right-clicking.

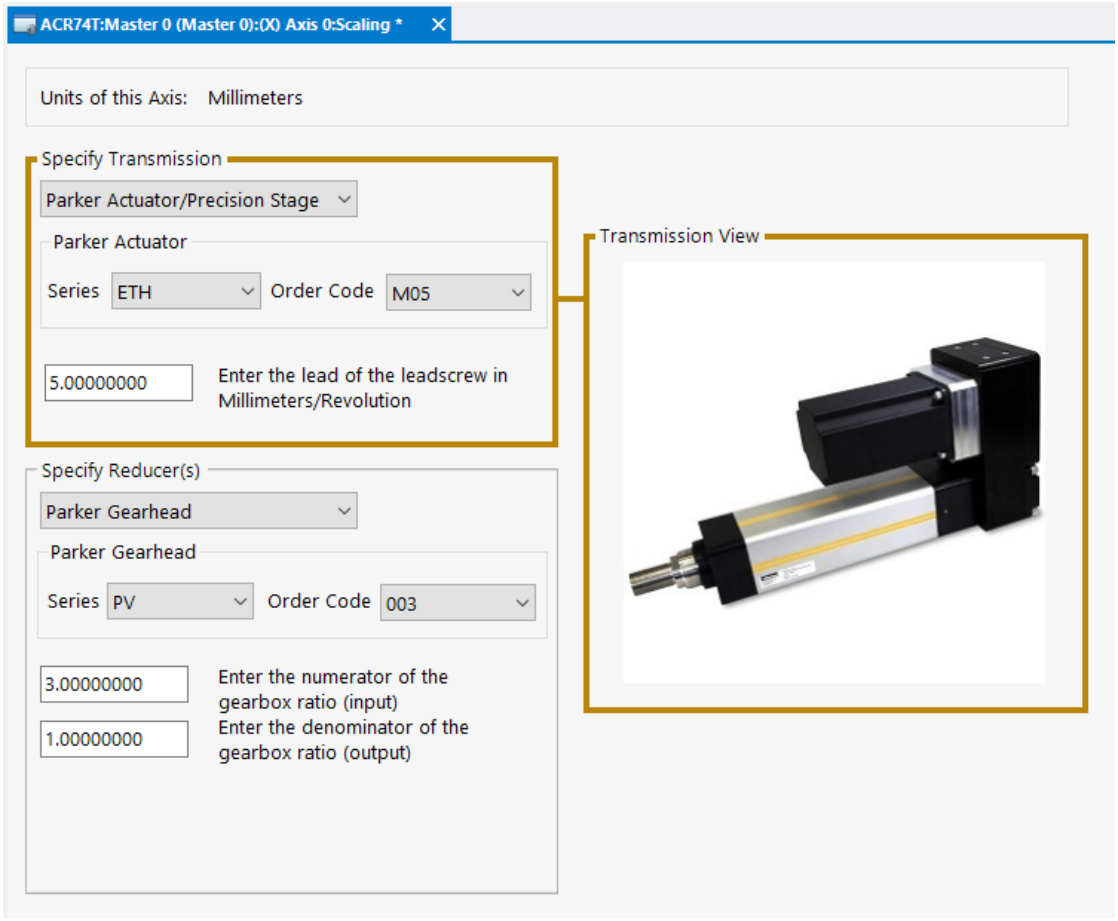


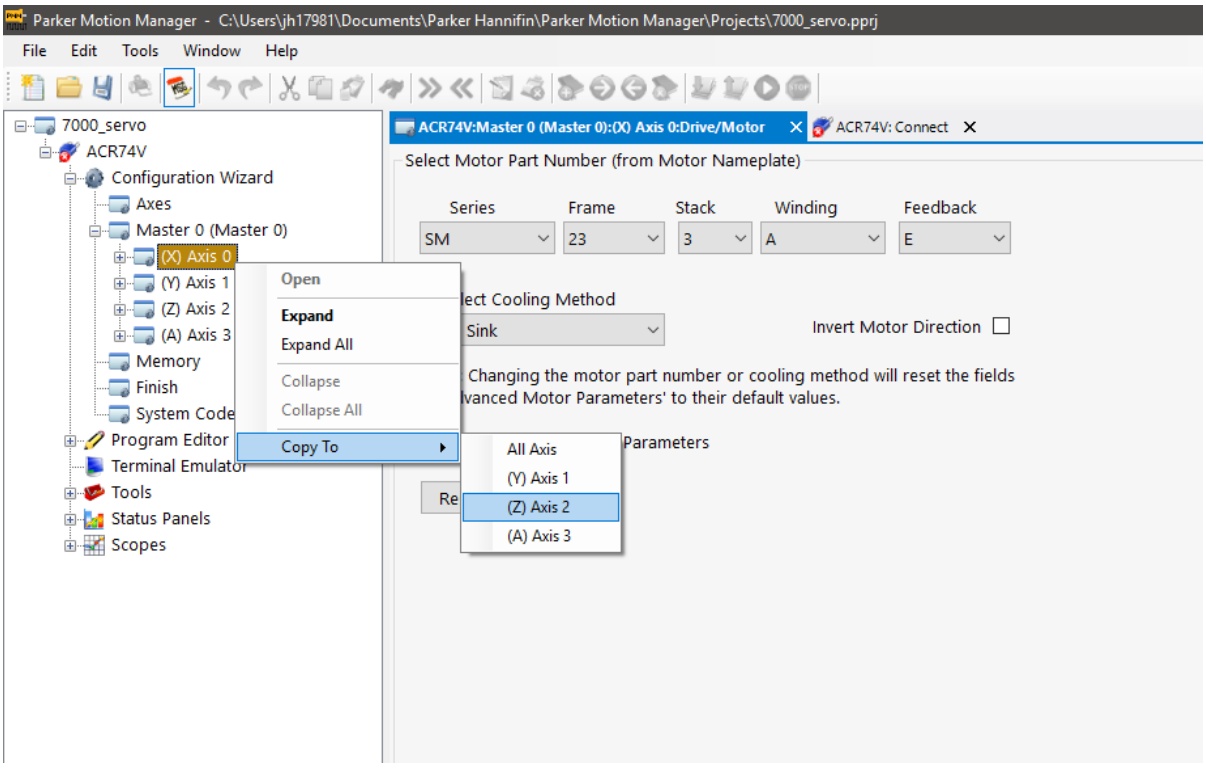

**Improvement 9**—PMM Windows can be docked, pinned or resized. They can also be popped out and placed anywhere on the screen, very useful for users with two monitors. This is a feature of PMM's new user interface engine.



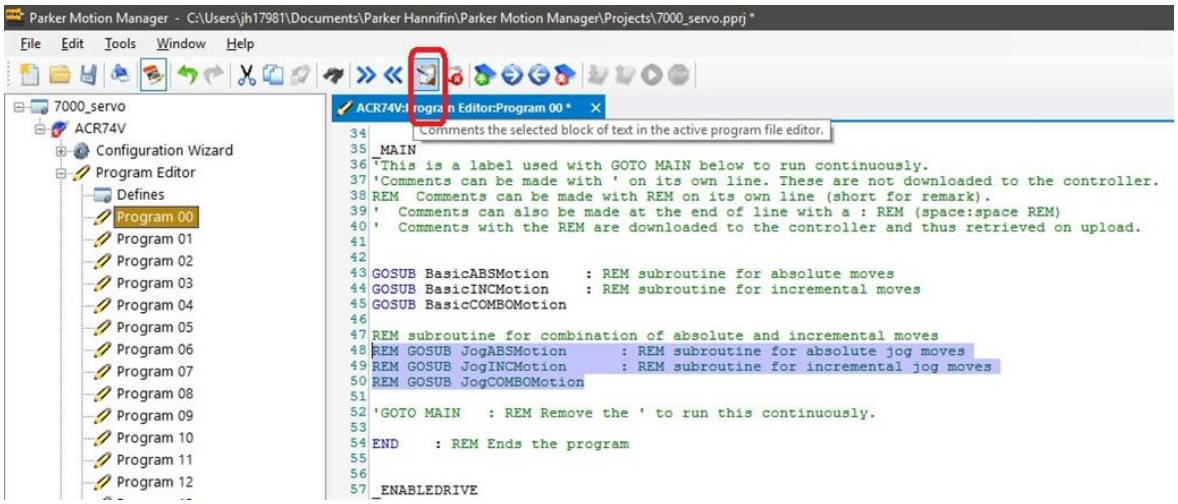

**Improvement 10**—The Configuration Wizard now includes Parker mechanics (actuators, precision tables and gearheads) for inches or millimeters. This saves the user from having to look up the stage and/or gearhead specifications in the catalog when configuring the unit scale.




**Improvement 11**—Same motor on multiple axis? Quickly populate the Configuration using *Axis Copy* (right-click on *Axis*).

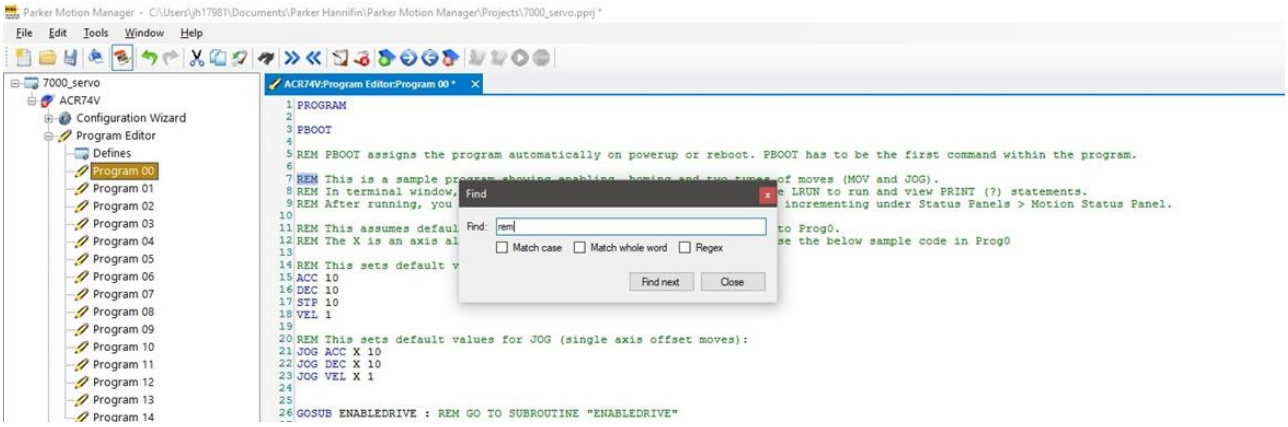



**Improvement 12**—Testing code and want to remove a section of code (comment out multiple lines)? Rather than typing `REM` or `'` at the beginning of each line, select the lines and use the Toolbar icon highlighted below.



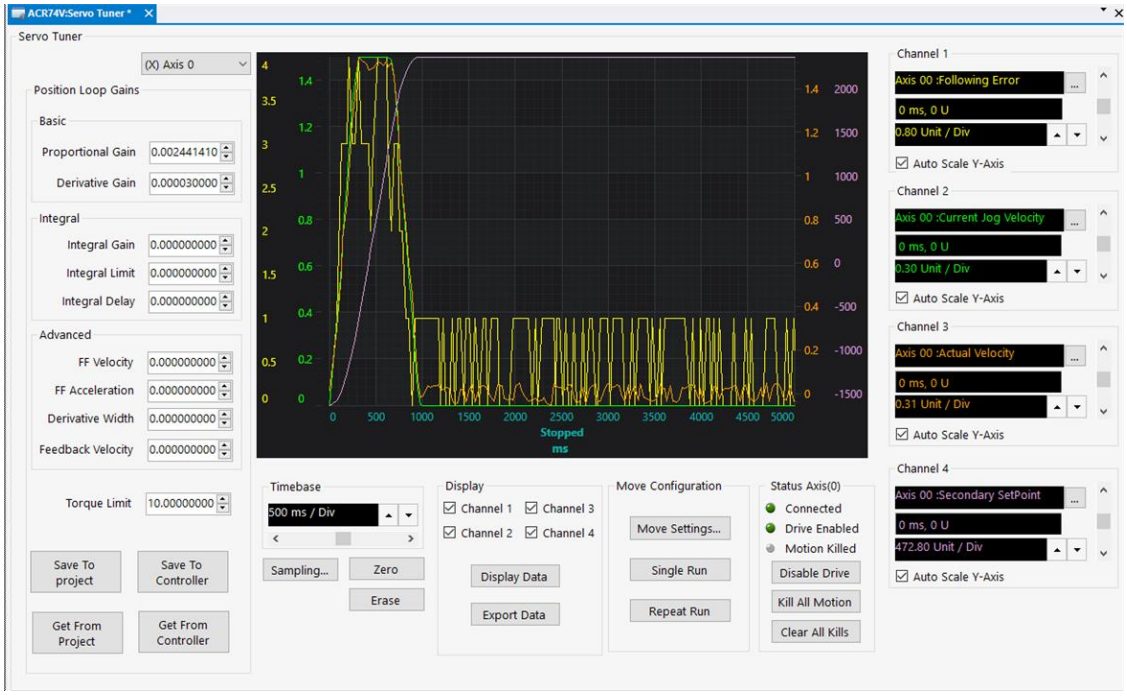

**Improvement 13**—In ACR-View, when doing a Find (CTRL+F), the first instance was found and then the focus was set on the program editor. So, when users pressed *Enter*, ACR-View would remove that text, inserting a carriage return and linefeed in its place.

In PMM, the focus remains on the Find Next window. So now when pressing *Enter*, it will move to the next instance, allowing users to quickly move through the code instead of clicking with the mouse. Not a big change, but a nice change for power users and heavy programmers.



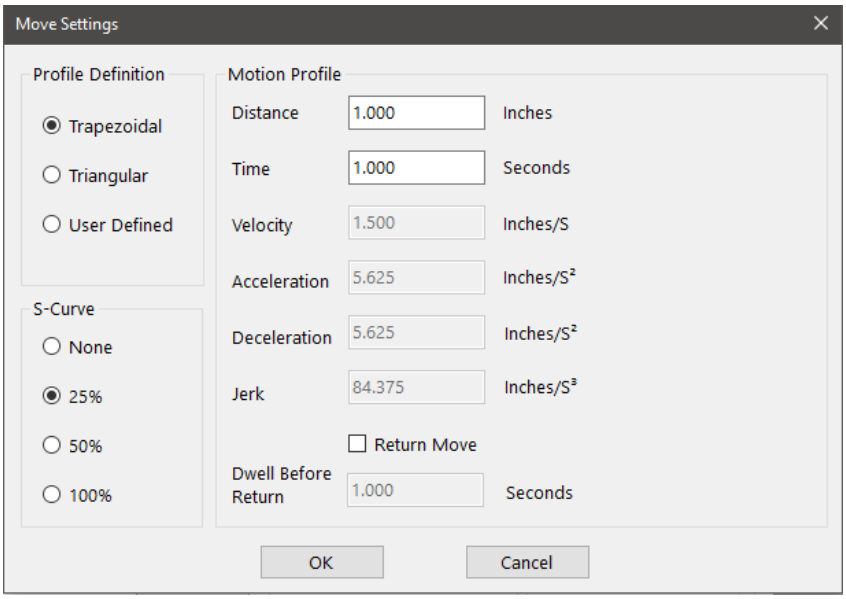

**Improvement 14**—Improved Servo Tuning screen and graphing!

Channel defaults are set for standard motion tuning. Auto-Scaling is turned on, so no more hunting for the signal.



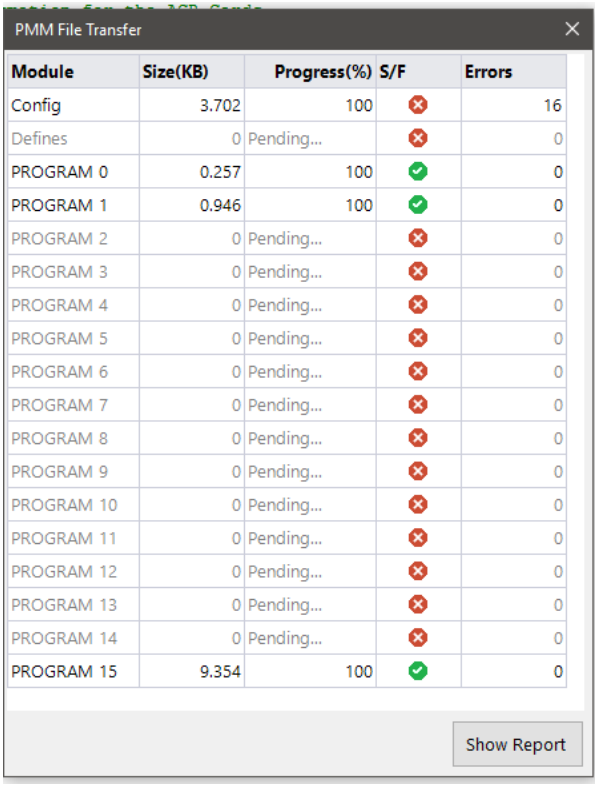

Note the Servo Tuner is a separate tool like in ACR-View, but there is not the simple step-tuning in the Configuration Wizard which was limited with high-resolution encoder systems. This was removed because tuning is best performed after configuration is complete.

Easy motion! Set the distance, time and move shape with optional S-curving and motion is generated automatically.

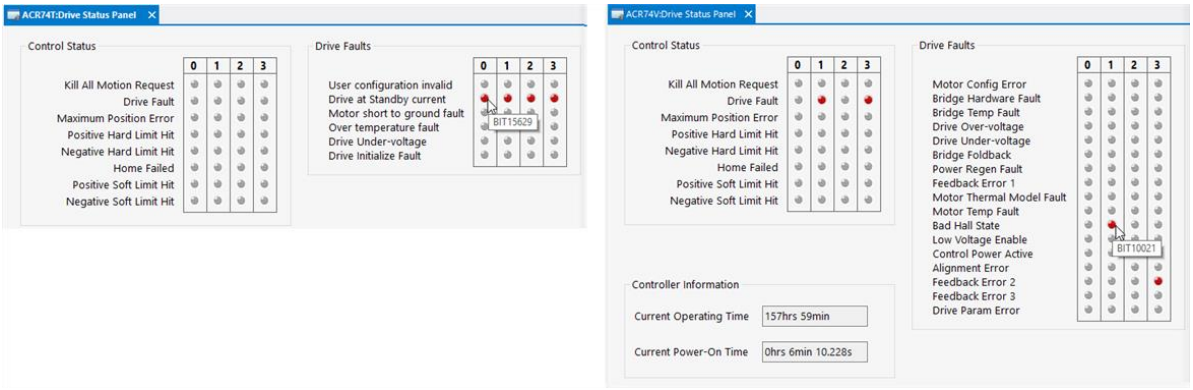



What is the value of each channel at a given point in time? Just mouse over! No need to export to Excel (though that is still available with the *Export Data* button).

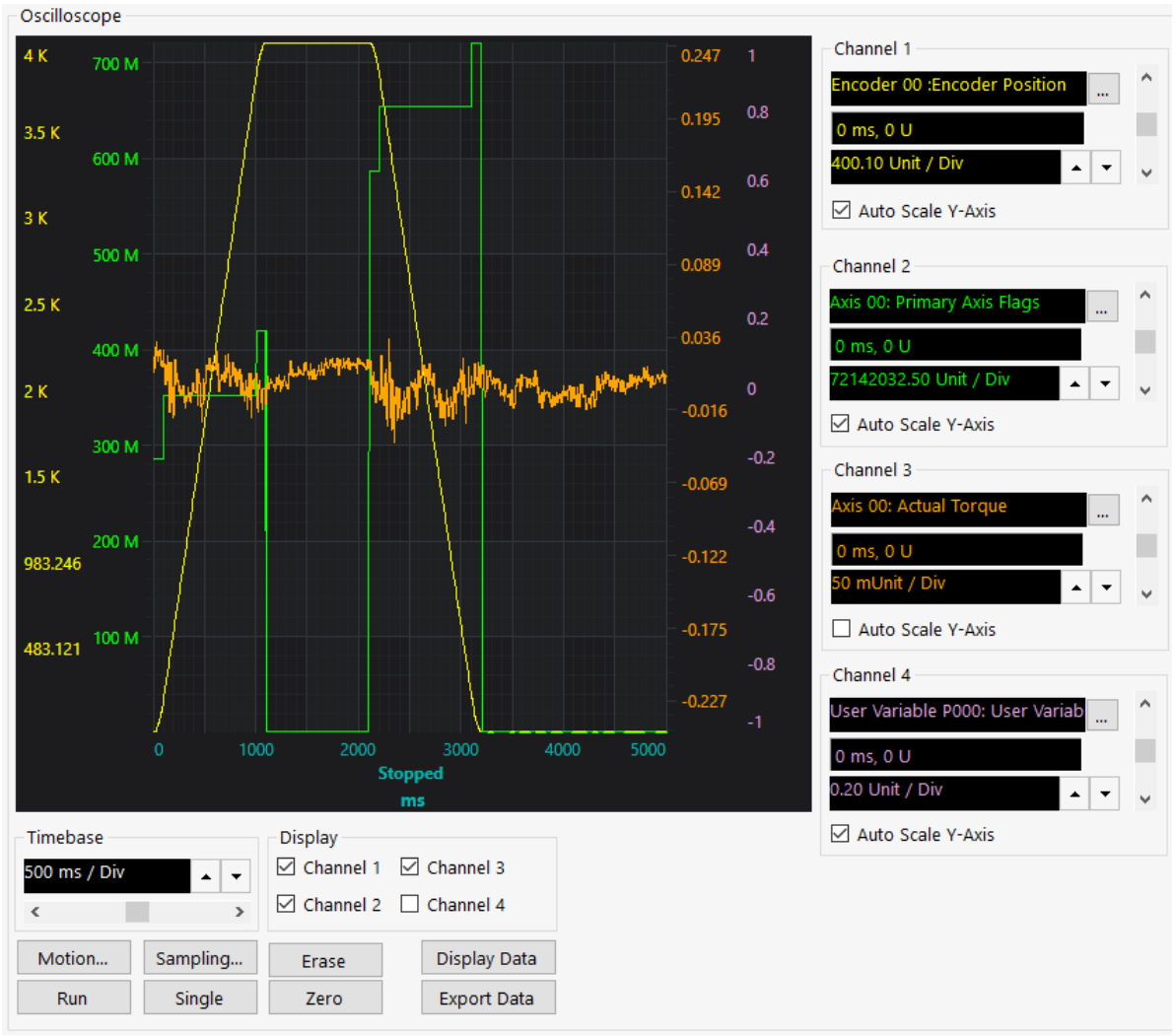

**Improvement 15**—PMM File Transfer provides status information during download. It will also highlight download errors.

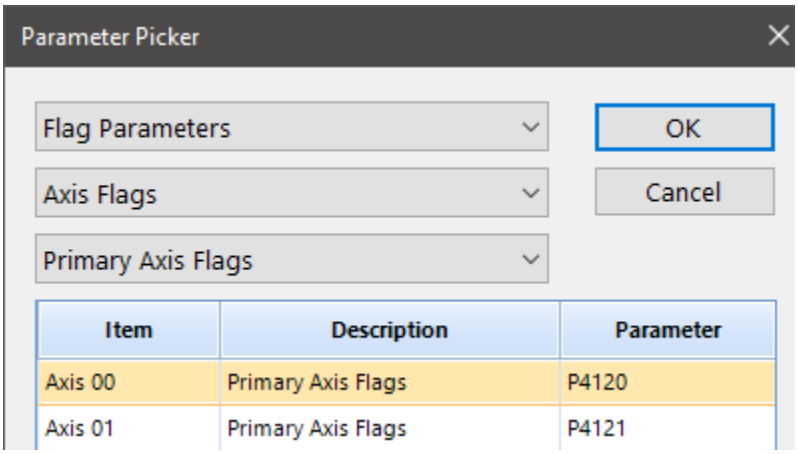



**Improvement 16**—Product-specific status panels.

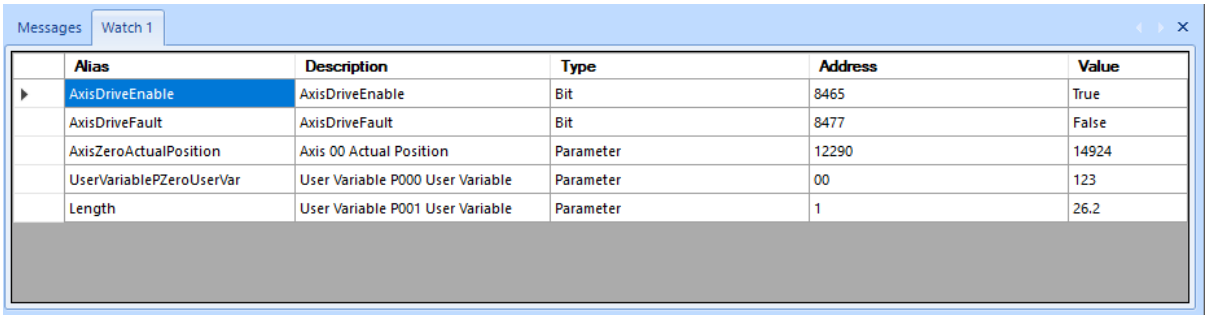



**Improvement 17**—Oscilloscope now has Flag Parameters available to graph. ACR-View had this function with a work-around, but now users can easily select Flag Parameters to visually graph these changes compared to other bits/parameters in their programs.



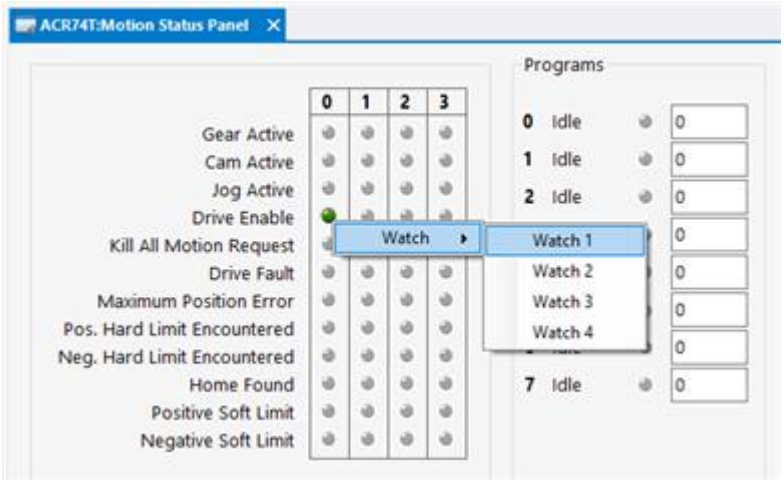




**Improvement 18**—Four Watch windows now available. Each Watch list can hold 20 lines of bits or parameters. No more switching between the Numeric and Bit Status panels!

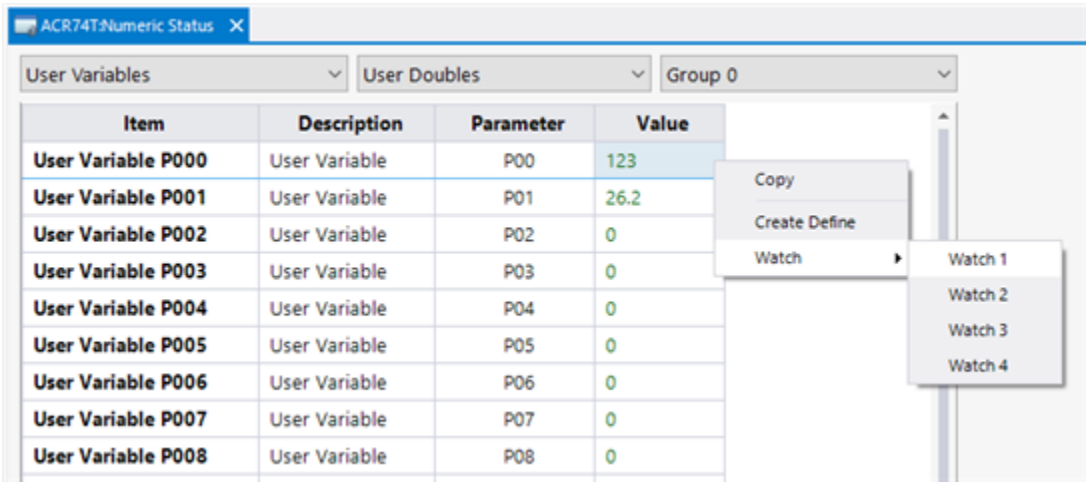



| | Alias | Description | Type | Address | Value |
|---|---|---|---|---|---|
| ▶ | AxisDriveEnable | AxisDriveEnable | Bit | 8465 | True |
| | AxisDriveFault | AxisDriveFault | Bit | 8477 | False |
| | AxisZeroActualPosition | Axis 00 Actual Position | Parameter | 12290 | 14924 |
| | UserVariablePZeroUserVar | User Variable P000 User Variable | Parameter | 00 | 123 |
| | Length | User Variable P001 User Variable | Parameter | 1 | 26.2 |

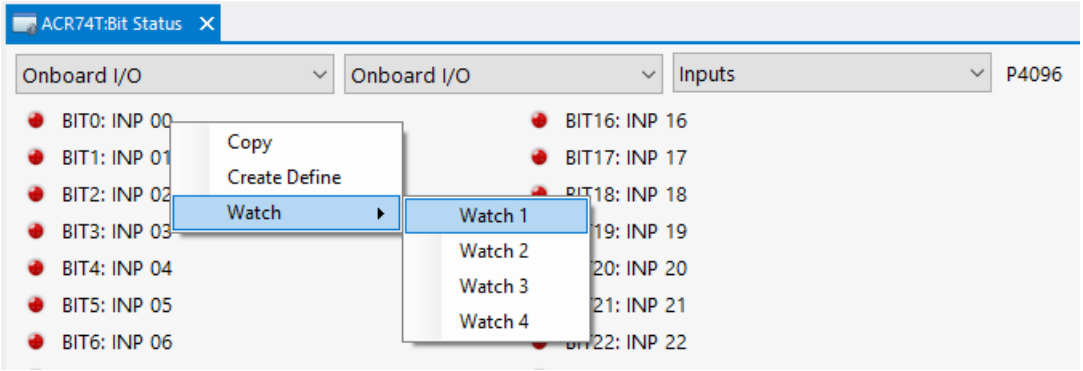
To populate the Watch lists, right-click on indicators within the Motion Status Panel.



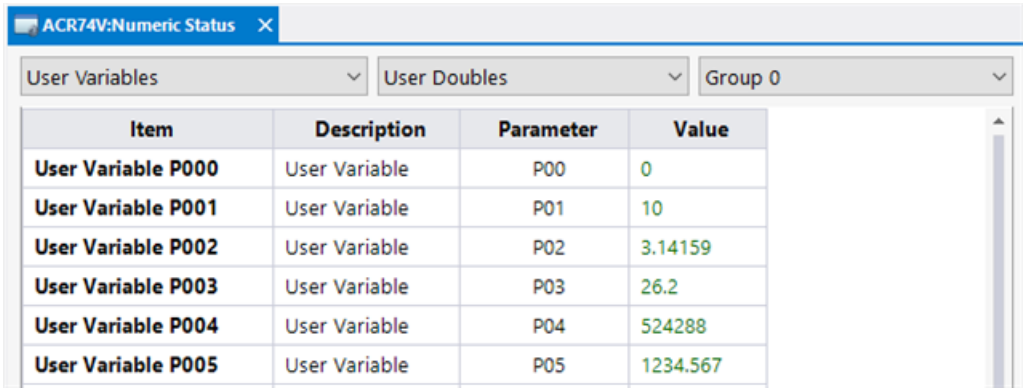

Or, from the Numeric Status Panel, Bit Status Panel or Defines editor, just right-click.




| Item | Description | Parameter | Value |
|---|---|---|---|
| **User Variable P000** | User Variable | P00 | 123 |
| **User Variable P001** | User Variable | P01 | 26.2 |
| **User Variable P002** | User Variable | P02 | 0 |
| **User Variable P003** | User Variable | P03 | 0 |
| **User Variable P004** | User Variable | P04 | 0 |
| **User Variable P005** | User Variable | P05 | 0 |
| **User Variable P006** | User Variable | P06 | 0 |
| **User Variable P007** | User Variable | P07 | 0 |
| **User Variable P008** | User Variable | P08 | 0 |

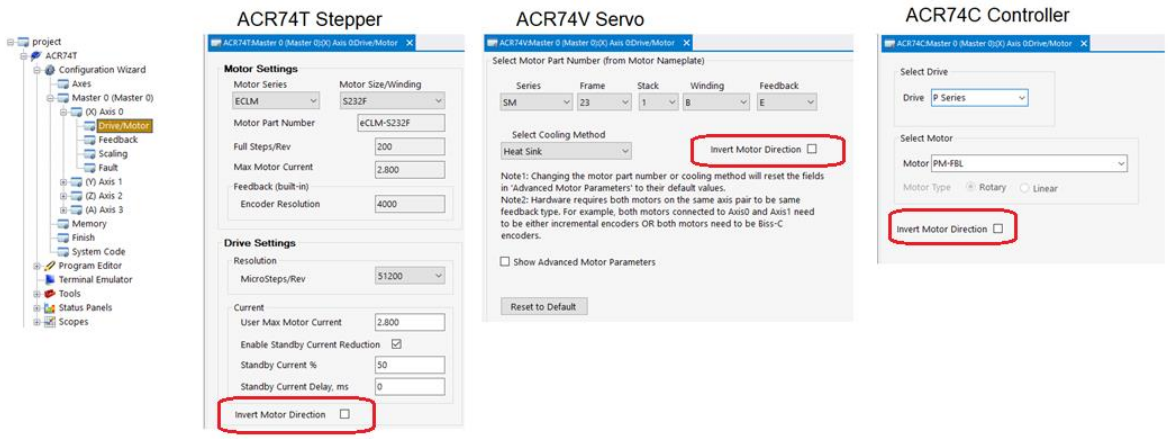



**Improvement 19**—User Parameters and User Bits are now in the Numeric and Bit Status panels! This includes user parameters P0-P4095, user non-volatile parameters (P39168-P39423) and non-volatile longs (P38912-P39167).



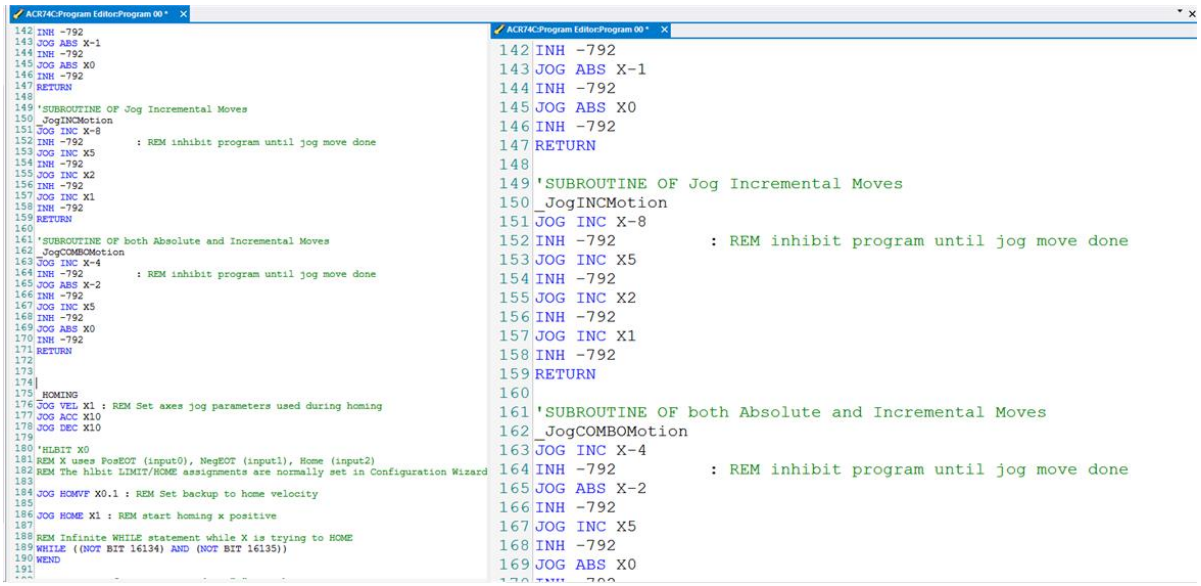

| Item | Description | Parameter | Value |
|---|---|---|---|
| **User Variable P000** | User Variable | P00 | 0 |
| **User Variable P001** | User Variable | P01 | 10 |
| **User Variable P002** | User Variable | P02 | 3.14159 |
| **User Variable P003** | User Variable | P03 | 26.2 |
| **User Variable P004** | User Variable | P04 | 524288 |
| **User Variable P005** | User Variable | P05 | 1234.567 |

**Improvement 20**—The positive direction on an axis is now easily changed with a check box in the Configuration Wizard.




**Improvement 21**—Save to Flash in ACR-View was 30-60 seconds. This has been greatly improved with PMM.

**Improvement 22**—New ease of use feature: zoom in the Program editor. Hold the CTRL key and use the mouse scroll wheel to zoom in and out.



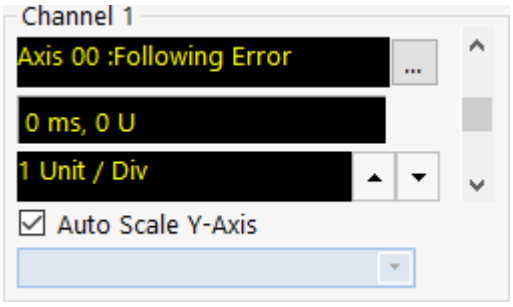
```
INH -792
JOG ABS X-1
INH -792
JOG ABS X0
INH -792
RETURN

'SUBROUTINE OF Jog Incremental Moves
_JogINCMotion
JOG INC X-8
INH -792          : REM inhibit program until jog move done
JOG INC X5
INH -792
JOG INC X2
INH -792
JOG INC X1
INH -792
RETURN

'SUBROUTINE OF both Absolute and Incremental Moves
_JogCOMBOMotion
JOG INC X-4
INH -792          : REM inhibit program until jog move done
JOG ABS X-2
INH -792
JOG INC X5
INH -792
JOG ABS X0
INH -792
RETURN


_HOMING
JOG VEL X1 : REM Set axes jog parameters used during homing
JOG ACC X10
JOG DEC X10

'HLBIT X0
REM X uses PosEOT (input0), NegEOT (input1), Home (input2)
REM The hlbit LIMIT/HOME assignments are normally set in Configuration Wizard

JOG HOMVF X0.1 : REM Set backup to home velocity

JOG HOME X1 : REM start homing x positive

REM Infinite WHILE statement while X is trying to HOME
WHILE ((NOT BIT 16134) AND (NOT BIT 16135))
WEND
```

**Improvement 23**—By default, PMM is set for 100 Defines, allowing users to name their bits/parameters. The default for ACR-View was 20.

**Improvement 24**—All programs have memory allocated by default, whereas in ACR-View, only Program 0 and Program 15 had memory allocated by default.

**Improvement 25**—Program 14 has been set for use with the oscilloscopes to prevent conflict with the user's arrays in their programs.

**Improvement 26**—Scope tools (Servo Tuner, Oscilloscope, etc.) can graph individual bits when configured to graph flag parameters.




**Improvement 27**—Scope tools (Servo Tuner, Oscilloscope, etc.) can now graph position and speed data in user units instead of encoder counts (at the user's option).

# APPENDIX E
# ACR7xC/ACR9000 Comparison

# ACR7xC/ACR9000 Comparison

This section covers similarities and differences between the ACR7xC controller and the ACR9000. The 7000 series controller is designed as a direct replacement for the ACR9000, supporting up to 8 servo and stepper axes in any combination.

The default IP address of all ACR7000 controllers is 192.168.100.1. The default for the ACR9000 was 192.168.10.40.

The ACR9000's Ethernet port did not auto-detect straight-through/crossover cables. The ACR7000 auto-detects straight-through or crossover ethernet cables. Both products support 10/100 Mbps.

The ACR7xC has the same axis connector pinout and discrete I/O pinout as the ACR9000.

The 7000 controller has two Ethernet ports available for use. The ports are configured as an unmanaged switch and have the same IP address. The ACR9000 had one ethernet port.

Like the ACR7xT and IPA, the ACR7xC can be an EtherNet/IP master to a Wago 750-363 EtherNet/IP bus coupler for expansion IO (or Wago's previous 750-352 or 750-341 bus coupler modules).

The ACR7xC can also be a slave on an EtherNet/IP network for an Allen Bradley or Omron PLC. This can be done at same time as being an EtherNet/IP master to a Wago 750 (although this is discouraged).

There is no serial or USB port, but there are five Ethernet streams available. The ACR9000 supported 4.

No battery! The ACR7000 is entirely flash-based. Using retentive variables such as P38912-P39167 (32-bit longs) or P39168-P39423 (32-bit floats)? No worries! The ACR7000 saves these in the background to behave the same as a battery-backed ACR9000 (9000PxUxBx), with the benefit of not having to periodically replace a battery or modify the program when upgrading.

No IEC support. Most users using the 9600 were using the AcroBASIC programs and not IEC, which was not expanded to work on Windows 10 anyway.

Powered from 24 VDC. The ACR9000 was powered from 120 VAC or 240 VAC.

The same AcroBASIC programming that has worked for 15+ years with the ACR9000 will work with the 7000.

Same ComACRServer: any PC applications (VB, LabVIEW, .NET, etc.) that a machine builder developed work the same with the 7000 as the 9000.

The 7000 controller has one 15-pin high density D-sub auxiliary encoder input that can only be used with incremental quadrature encoders. The ACR9000 had one (2/4-axis models) or two (6/8-axis models) 9-pin D-sub auxiliary encoder ports that supported incremental and SSI encoders.

The same axis I/O cables can be used, allowing users to easily upgrade systems. The same cables to connect to Parker drives all work with the 7000 series controller: P series, Aries, Compax3, Gemini Servo and Stepper, Zeta, E-AC, etc. The 7000 controller supports SSI feedback on its axis connectors just like the 9000.

Much smaller. A four-axis 9000P3U4B0 and a four-axis ACR74C-A0V2C1 are shown below.

New and improved software, Parker Motion Manager, supports the ACR7000 series controller, integrated stepper controller and integrated servo controllers. Anyone familiar with ACR-View will be able to pick it up very quickly since it has a similar look and feel, but we have taken out what is not needed and made drastic improvements to the usability.

The ACR7xC does not have RS485 on the axis connectors and thus do not support Drive Talk and those related commands (DTALK).

The ACR7000 does not have EthernetPOWERLINK and those related commands are not supported (EPLC).

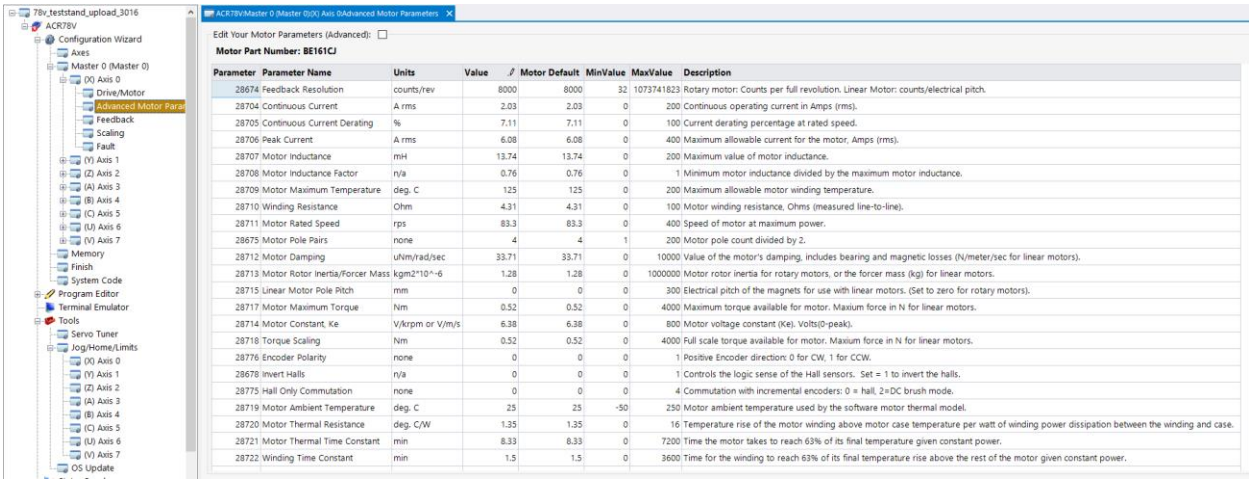
More form factors are available beyond just the ACR7xC. The ACR7000 integrated stepper (ACR7xT) and ACR7000 integrated servo (ACR7xV) are available for new applications with a multi-axis controller and multiple motor drives within a single package, saving space and cabling.

# APPENDIX F
# ACR7xV/IPA Differences

# ACR7xV/IPA Differences

The 7000 series integrated drive/controller products share aspects of the 9000 multi-axis controller and the IPA integrated servo drive/controller. All the 9000 commands, parameters, and flags that are used to control programs, masters and axes are present on the 7000 series. These include the position loop servo gain command and parameters specifically. There are also parameters and flags used for servo drive or stepper drive configuration and status reporting. Those for the servo drive are analogous to the "C" and "S" parameters of the IPA in terms of function but are consistent with the ACR parameter and flag model rather than the legacy Aries parameters and commands. Also, these values may be saved with the `ESAVE` command and will automatically be restored on power up, but they are not automatically saved like the Aries-style parameters of the IPA.

The motor parameters can be viewed in both the Numeric Status Panel and under Configuration Wizard → Drive/Motor → Show Advanced Motor Parameters.




| Parameter | Parameter Name | Units | Value | Motor Default | MinValue | MaxValue | Description |
|---|---|---|---|---|---|---|---|
| 28674 | Feedback Resolution | counts/rev | 8000 | 8000 | 32 | 1073741823 | Rotary motor: Counts per full revolution. Linear Motor: counts/electrical pitch. |
| 28704 | Continuous Current | A rms | 2.03 | 2.03 | 0 | 200 | Continuous operating current in Amps (rms). |
| 28705 | Continuous Current Derating | % | 7.11 | 7.11 | 0 | 100 | Current derating percentage at rated speed. |
| 28706 | Peak Current | A rms | 6.08 | 6.08 | 0 | 400 | Maximum allowable current for the motor. Amps (rms). |
| 28707 | Motor Inductance | mH | 13.74 | 13.74 | 0 | 200 | Maximum value of motor inductance. |
| 28708 | Motor Inductance Factor | n/a | 0.76 | 0.76 | 0 | 1 | Minimum motor inductance divided by the maximum motor inductance. |
| 28709 | Motor Maximum Temperature | deg. C | 125 | 125 | 0 | 200 | Maximum allowable motor winding temperature. |
| 28710 | Winding Resistance | Ohm | 4.31 | 4.31 | 0 | 100 | Motor winding resistance, Ohms (measured line-to-line). |
| 28711 | Motor Rated Speed | rps | 83.3 | 83.3 | 0 | 400 | Speed of motor at maximum power. |
| 28675 | Motor Pole Pairs | none | 4 | 4 | 1 | 200 | Motor pole count divided by 2. |
| 28712 | Motor Damping | uNm/rad/sec | 33.71 | 33.71 | 0 | 10000 | Value of the motor's damping, includes bearing and magnetic losses (N/meter/sec for linear motors). |
| 28713 | Motor Rotor Inertia/Forcer Mass | kgm2*10^-6 | 1.28 | 1.28 | 0 | 1000000 | Motor rotor inertia for rotary motors, or the forcer mass (kg) for linear motors. |
| 28715 | Linear Motor Pole Pitch | mm | 0 | 0 | 0 | 300 | Electrical pitch of the magnets for use with linear motors. (Set to zero for rotary motors). |
| 28717 | Motor Maximum Torque | Nm | 0.52 | 0.52 | 0 | 4000 | Maximum torque available for motor. Maxium force in N for linear motors. |
| 28714 | Motor Constant, Ke | V/krpm or V/m/s | 6.38 | 6.38 | 0 | 800 | Motor voltage constant (Ke). Volts(0-peak). |
| 28718 | Torque Scaling | Nm | 0.52 | 0.52 | 0 | 4000 | Full scale torque available for motor. Maxium force in N for linear motors. |
| 28776 | Encoder Polarity | none | 0 | 0 | 0 | 1 | Positive Encoder direction: 0 for CW, 1 for CCW. |
| 28678 | Invert Halls | n/a | 0 | 0 | 0 | 1 | Controls the logic sense of the Hall sensors. Set = 1 to invert the halls. |
| 28775 | Hall Only Commutation | none | 0 | 0 | 0 | 4 | Commutation with incremental encoders: 0 = hall, 2=DC brush mode. |
| 28719 | Motor Ambient Temperature | deg. C | 25 | 25 | -50 | 250 | Motor ambient temperature used by the software motor thermal model. |
| 28720 | Motor Thermal Resistance | deg. C/W | 1.35 | 1.35 | 0 | 16 | Temperature rise of the motor winding above motor case temperature per watt of winding power dissipation between the winding and case. |
| 28721 | Motor Thermal Time Constant | min | 8.33 | 8.33 | 0 | 7200 | Time the motor takes to reach 63% of its final temperature given constant power. |
| 28722 | Winding Time Constant | min | 1.5 | 1.5 | 0 | 3600 | Time for the winding to reach 63% of its final temperature rise above the rest of the motor given constant power. |

# APPENDIX G
# 6K to ACR Command Reference

# 6K to ACR Command Reference

This section covers common 6K commands and their closest ACR equivalents where applicable. The ACR uses a different architecture than the ACR, so this table should be taken as suggestion only.

**NOTE: "P" parameters and flags will show the parameter/flag number for the first axis or master. Others can be found in PMM help file.**

Most 6K programs only run one at a time. ACR programs can all be running at the same time. So, most 6K programs would be rewritten in ACR as subroutines with labels, unless the 6K is using multitasking.

***6K to ACR Command Crossover Table***

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| Scaling Setup | | | |
| SCLA, SCLV, SCLD | PPU | | Requires RES after changing PPU. Part of Configuration Wizard. |
| SCLMAS | GEAR PPU | | |
| Limits | | | |
| LH3,1 | HLIM X3 Y1 | | |
| LS2,3 | SLIM X2 Y3 | | |
| LSNEG -1: LSPOS +1 | SLM X1 | | Could also set differently using SLM X(-10,25). Axis 0 bits shown. SET and CLR bits as appropriate (Positive EOT bit shown is for Axis 0). |
| LIMLVL000 | SET 16144 : SET 16145 : CLR16146 | | |
| INFNC0-R, INFNC1-S, INFNC2-T | HLBIT X0 or HLBIT X(1,0,2) | | HLBIT X0: Negative and home automatically assigned as next contiguous inputs (1 and 2). Set within Configuration Wizard on Fault Screen. |
| LHAD100,200 | HLDEC X100 Y200 | | |
| TLIM | ?BIT16132 | | |
| | ?BIT16133 | | |
| | ?BIT 16130 | | |

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| Homing | | | |
| HOMZ | MSEEK | | Home to a Z-channel (mode 0). |
| HOM0, HOM1 | JOG HOME X1, JOG HOME X-1, MSEEK | | Home to a trigger input (mode 2). |
| HOMVF HOMA | JOG HOMVF, JOG ACC | | |
| HOMAA/HOMADA HOMAD | JOG JRK, JOG DEC | | |
| HOMBAC1, HOMDF1, HOMEDG1 | SET16152, SET16154, SET16153 | | Home Backup Enable bit must be on (BIT 16152). |
| PSET | RES REN | | Use to reset or preset the position counters for an axis. Zeroes the Current Position (MOV) register and adds parameter.<br><br>See also: JOG RES, GEAR RES, CAM RES JOG REN. |
| Non-Interpolated Motion | | | |
| Incremental Motion | | | |
| D+4:MC0:MA0:GO1 | JOG INC X4 | | See Also: JOG ACC, JOG DEC, JOG VEL |
| D+4, -3 : MC00 : MA00 : GO11 | JOG INC X4 Y-3 | | |
| Absolute Motion | | | |
| D : MC0 : MA1 : GO1 | JOG ABS X4 | | Move to the Jog Offset register's absolute position. |
| Continuous Motion | | | |
| D+ : MC1 : GO1 | JOG FWD X | SET 796 | Flags shown for Axis 0. |
| D- : MC1 : GO1 | JOG REV X | SET 797 | |
| S1 | JOG OFF X | CLR 796 : CLR 797 | |
| Non-Interpolate Motion Trajectory | | | |
| A | JOG ACC | | Scaled by PPU to user units/second$^2$. |

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| AD | JOG DEC | | Scaled by PPU to user units/second$^2$.<br><br>Scaled by PPU to user units/second<br><br>Scaled by PPU to user units/second^3. Pure S-Curve |
| V | JOG VEL | | Scaled by PPU to user units/second. |
| AA/ADA | JOG JRK | | Scaled by PPU to user units/second$^3$. |
| Interpolated Motion | | | |
| S | SET BIT 523 | SET 523 | Flags shown for Master 0. Uses DEC setting. |
| K | SET BIT 522 | SET 522 | Uses the HLDEC deceleration ramp. |
| !K | CTRL+X or CTRL+Z | | CTRL+X kill all motion for all programs. CTRL+Z kills motion and disables all drives. |
| KDRIVE | SET BIT 8471 | SET 8471 | BITS 8471, 8503, 8535, 8567, 8599, 8631, 8663, 869. |
| Linear Interpolated Motion | | | |
| D 2, 3 : MC00 : MA11 : GOL11 | MOV X2 Y3 | X2 Y3 | Absolute moves. |
| D 7, 8 : MC00 : MA00 : GOL11 | MOV X/7 Y/8 | X/7 Y/8 | Incremental moves. |
| D 4, 5 : MC00 : MA10 : GOL11 | MOV X4 Y/5 | X4 Y/5 | Mixed moves. |
| Circular 2D Interpolated | | | |
| PARCOM | CIRCCW | | Counter-clockwise. |
| PARCOP | CIRCW | | Clockwise. |
| PARCOM/PARCOP | SINE | | |
| PARCM | No equivalent | | |
| PARCP | No equivalent | | |
| Circular 3D Interpolated | | | |

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| No equivalent | TARC | | Axes must have same PPU. |
| Interpolated Motion Trajectory | | | |
| PA | ACC | | Scaled by PPU to user units/second$^2$ |
| PAD | DEC/STP | | Scaled by PPU to user units/second$^2$ |
| PV | VEL | | Scaled by PPU to user units/second |
| PAA/PADA | JRK | | Scaled by PPU to user units/second$^3$. Pure S-curve is ACC**2/VEL.<br><br>See also: IVEL, FVEL, F |
| PAXES | TANG | | |
| Interpolated Motion Trajectory | | | |
| FOLEN | GEAR ON/OFF | | LOCK can be used for gantry axes. |
| FOLMAS | GEAR SRC | | Can gear to any encoder or parameter. |
| FOLRN<br><br>FOLRD | GEAR RATIO | | GEAR RATIO sign determines direction.<br><br>Use ratio rather than decimal number, ex. "(1/10)". GEAR RATIO is a 64-bit floating point value. |
| FOLMD | GEAR ACC/DEC | | GEAR ACC/DEC is change in ratio over time. |
| SCLMAS | GEAR PPU | | |
| FMCLEN | No direct equivalent.<br>Use MOD. | | |
| NMCY | No direct equivalent. | | Can use simple division algorithm and use whole number. |
| Tuning | | | |
| All ACR gains are in volts. 6K gains are in volts, millivolts or microvolts. | | | |
| SGP (mV) | PGAIN | | PGAIN = SGP/1000 |
| SGV (uV) | DGAIN | | DGAIN = SGV/1e6 |

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| SGI (mV) | IGAIN | | IGAIN = SGI/1000 |
| SGILIM (mV) | ILIMIT | | ILIMIT = SGILIM/1000 |
| SGVF (uV) | FFVEL | | FFVEL = SGVF/1e6 |
| SGAF (uV) | FFACC | | FFACC = SGAF/1e6 |
| DACLIM (V) | TLM | | TLM = DACLIM |
| TDAC | PRINT P6400 | ?P6400 | ?Pnnnn prints the value in parameter nnnn. |
| DAC | P6400 | | |
| Communications | | | |
| NTADDR192,168,10,30 | IP "192.168.100.1" | | Defaults shown. Requires ESAVE and REBOOT to take effect |
| NTMASK255,255,255,0 | IP MASK "255.255.255.0" | | |
| Variables | | | |
| VARB1 | User Flag Parameters | | Bits 128-255 (P4100-4104) and bits 1920-2047 (P4156-4159) are user bits. |
| VARI1 | LV0 | | Must dimension local variables first. Example: DIM SV(10).<br><br>Variables start at "0" for ACR controllers. |
| VARS1 | $V0 | | Must dimension number and length of string variables. |
| VAR1-255 | SV0 or DV0 | | SV are 32-bit floating point local variables. DV are 64-bit floats. LV are 32-bit LONG integers. |
| VAR1-255 | P0-P4095 | | P0-P4095 are 64-bit global floating point user variables (already default in Config Wizard). |
| Position Counters | | | |
| 1TPC | ?P12295 | | Secondary Setpoint in raw counts. It is the sum of interpolated command, jog, gear, cam (Primary |

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| | | | Setpoint) with Backlash and Ballscrew Compensation. |
| 1TPE | ?P12290 | | Actual Position in raw counts. Depends upon ENC SRC Following Error in raw counts. |
| 1TPER | ?P12291 | | Hardware Position capture in raw counts. |
| !TPCE | ?P12292 | | See also: INTCAP, HSINT, GEAR ON/OFF TRG. |
| ENCPOL1 | ENC0 MULT -4 | | Valid values are "4" and "-4" for ACR/IPA. |
| SMPER 0.25,0.33 | EXC X0.25 Y0.33 | | EXC X0 does not disable excess error checking. |
| SMPER0 | CLR 8469 | | Disables error checking. |
| Program | | | |
| BREAK | END | | Used to terminate program. |
| COMMENT (;) | REM | | Comment is stored. |
| | Apostrophe ( ' ) | | Comment is stripped. MUST be on its own line. |
| DEL | NEW | | Automatically performed by PMM, not needed in |
| DEF | PROGRAM | | Starts program definition |
| END | ENDP | | Used to terminate program definition. |
| GOTO program | GOTO label | | You cannot GOSUB or GOTO to another program; use label in the same ACR program instead. |
| GOSUB program | GOSUB label | | |
| Drive Control | | | |
| DRIVE0 | DRIVE OFF X | CLR 8465 | AXIS0 DRIVE OFF |
| DRIVE1 | DRIVE ON X | SET 8465 | AXIS0 DRIVE ON |
| No Equivalent | DRIVE RES X | | Axis LED: Green = Enabled, Red = Disabled or Faulted |

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| | | | AXIS0 DRIVE RES: toggles axis' reset output. |
| IO Control | | | |
| OUT1 | SET BIT 32 | SET32 or BIT32 = 1 | Could also use P4097 = P4097 OR 1 |
| OUT0 | CLR BIT 32 | CLR32 or BIT32 = 0 | Could also use P4097 = P4097 AND 2**31 |
| OUTXXX1 | SET BIT 35 | SET 35 or BIT35 = 1 | Bit0 - 31 are inputs. Discrete outputs start at bit32 |
| OUTXXX0 | CLR 35 | CLR 35 or BIT35 = 0 | |
| TIN | ?P4096 | | Response is a decimal representation of binary bits |
| TIN.1 | ?BIT0 | | Reports back a 0 (zero) for inactive, -1 for active input |
| 1TIN | ?P4104 | | Response is a decimal representation of binary bits |
| TOUT | ?P4097 | | Response is a decimal representation of binary bits |
| 1TOUT | ?P4105 | | Response is a decimal representation of binary bits |
| TOUT.1 | ?BIT32 | | Reports back a 0 (zero) for inactive, -1 for active output |
| Other | | | |
| SSFR | PERIOD | | ACR7000 and IPA much faster than 6K / Gem6K. |
| BAUD | No serial communication. | | |
| TREV | VER | | Also can use P7042 and P7043 to retrieve VER and U. |
| TCOM | HELP | | Shows a list of reserved words that should not be used as aliases. |

| 6K Command | ACR Command | Shorthand | ACR Command Notes |
|---|---|---|---|
| ENCSND0 | ENC0 SRC0 | | Quadrature encoder mode. |
| ENCSND1 | ENC0 SRC1 | | Step and direction mode. |
| CMDDIR | BIT8455 | | Could also use ENC MULT and DAC GAIN together (required for ACR7xC). Already part of Config Wizard. |
| EPM | PM | | See Position Maintenance. |
| ANI | P6408 | | ADC inputs. |
| ANO | P6400 | | DAC outputs. |
| AS, ASX | P4120, 4168, 4296, 4360, 4600 | | There are five groups of axis flags in the ACR. |

# APPENDIX H
# ACR7000 Bits and Parameters

# ACR7000 Bits and Parameters

This section covers bits and parameters added to the ACR firmware specifically to handle new hardware features on the ACR7000. Relevant bits and parameters therefore vary by product.

## *ACR7xT Control and Status Bits*

The control, status, fault and warning bits are in the Stepper Flags. These bits will only have meaning for integrated steppers and will be ignored for other types of steppers. The bit numbers shown below are for Stepper 0, P4584. For each subsequent stepper, add 32 to the bit number.

| **Control Bits** | **Flag** | **Fault and Warning Bits** | **Flag** |
|---|---|---|---|
| Assert new configuration | 15616 | Motor short to ground fault | 15624 |
| Set parameters to factory default | 15617 | Over temperature warning | 15625 |
| Enable Auto Standby | 15618 | Over temperature fault | 15626 |
| Assert Standby current | 15619 | Stall threshold warning | 15627 |
| **Status Bits** | | Under voltage | 15628 |
| Driver chip configured | 15620 | Drive at Standby current | 15629 |
| Driver chip configuration underway | 15621 | General fault | 15630 |
| User configuration invalid, default installed | 15622 | Debug Control/Status Bits | |
| Change to Standby current underway | 15623 | Assert Debug write register | 15632 |
| | | Debug configuration underway | 15633 |

## *ACR7xT Latched Fault and Warning Bits*

The fault and warning bits listed above represent the instantaneous state of any fault or warning that is present in the drive hardware. But these bits will disappear if the underlying hardware condition disappears, so it may not be possible to diagnose a problem with the current state only. The bits below represent a sort of latched state of the fault and warning bits listed above. Every time the bits above are monitored, their state is OR'ed into the corresponding bits below. That allows a persistent record of any of the bits above having been present. The bits below are cleared when the drive is enabled from a previously disabled state.

| **Latched Fault and Warning Bits** | **Flag** | | | |
|---|---|---|---|---|
| | Axis 0 | Axis 1 | Axis 2 | Axis 3 |

| | P4584 | P4585 | P4586 | P4587 |
|---|---|---|---|---|
| Motor short to ground fault | 15640 | 15672 | 15704 | 15736 |
| Over temperature warning | 15641 | 15673 | 15705 | 15737 |
| Over temperature fault | 15642 | 15674 | 15706 | 15738 |
| Stall threshold warning | 15643 | 15675 | 15707 | 15739 |
| Under voltage | 15644 | 15676 | 15708 | 15740 |
| Not used | 15645 | 15677 | 15709 | 15741 |
| General fault | 15646 | 15678 | 15710 | 15742 |

# *ACR7xT Control and Status Parameters*

The control and status parameters are additional stepper parameters. These parameters will only have meaning for integrated steppers and will be ignored for other types of steppers. The parameter numbers shown below are for stepper 0. For each subsequent stepper, add 16 to the parameter number. Unlisted parameters are reserved. The parameters are listed first and then fully described individually in the paragraphs below.

| **Float Parameter Descriptions** | **Parameter** | **Default Value** | **Range** |
|---|---|---|---|
| Full scale current (Amps) | P7936 | N/A | Value reported as status |
| Product maximum motor current (Amps) | P7937 | N/A | Value reported as status |
| User selected maximum motor current (Amps) | P7938 | 0.5 | 0.0-4.0 |
| Motor resistance (ohms) | P7939 | 0.9 | 0.1-15.0 |
| Motor inductance (mH) | P7940 | 2.5 | 0.1-40.0 |
| Integer Parameter Descriptions | Parameter | Default Value | Range |
| Standby Current Percentage | P7944 | 100 | 3-100 |
| Time from full to standby current (msec.) | P7945 | 0 | 0-5000 |
| Micro-steps per full step (power of 2) | P7946 | 256 | 1-256 |
| Configuration error code | P7947 | N/A | Value reported as status |
| Drive Register write value (for debug) | P7948 | N/A | |

| | | | |
|---|---|---|---|
| Drive Register read value (for debug) | P7949 | N/A | Value reported as status |
| Drive Register tuning value (for tuning) | P7950 | 73765 | 0-131071 (0x1ffff) |
| Drive Register raw status read | P7951 | N/A | 16 or 20 bit number |

The table below shows the possible values returned in P7947 above and their meanings.

| Error description | Value |
|---|---|
| No error | 0 |
| Max Current setting range error | 1 |
| Motor resistance range error | 2 |
| Motor Inductance range error | 3 |
| Standby Current range error | 4 |
| Standby time range error | 5 |
| Micro-stepping setting value error | 6 |

# *ACR7xV Configuration Bits and Parameters*

The table below shows the parameters used for servo drive configuration. Note these are set from PMM's Configuration Wizard. These parameters occupy the same parameter space that had been used for Drive Talk on the ACR9000 and in most cases, have the same names and meanings as those parameters.

| Parameter | Parameter Name | Units | Value | Motor Default | Min Value | Max Value | Description |
|---|---|---|---|---|---|---|---|
| 28674 | Feedback Resolution | Counts/Rev | 524288 | 524288 | 32 | 1073741823 | Rotary motor: Counts per full revolution. Linear Motor: counts/electrical pitch. |
| 28704 | Continuous Current | A rms | 2.6 | 2.6 | 0 | 200 | Continuous operating current in Amps (rms). |
| 28705 | Continuous Current Derating | % | 10 | 10 | 0 | 100 | Current derating percentage at rated speed. |
| 28706 | Peak Current | A rms | 7.8 | 7.8 | 0 | 400 | Maximum allowable current for the motor, Amps (rms). |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 28707 | Motor Inductance | mH | 5.68 | 5.68 | 0 | 200 | Maximum value of motor inductance. |
| 28708 | Motor Inductance Factor | None | 0.75 | 0.75 | 0 | 1 | Minimum motor inductance divided by the maximum motor inductance. |
| 28709 | Motor Maximum Temperature | °C | 125 | 125 | 0 | 200 | Maximum allowable motor winding temperature. |
| 28710 | Winding Resistance | Ohm | 1.33 | 1.33 | 0 | 100 | Motor winding resistance, Ohms (measured line-to-line). |
| 28711 | Motor Rated Speed | RPS | 83.3 | 83.3 | 0 | 400 | Speed of motor at maximum power. |
| 28675 | Motor Pole Pairs | None | 4 | 4 | 1 | 200 | Motor pole count divided by 2. |
| 28712 | Motor Damping | µNm/rad/sec | 67 | 67 | 0 | 10000 | Value of the motor's damping, includes bearing and magnetic losses (N/meter/sec for linear motors). |
| 28713 | Motor Rotor Inertia/Forcer Mass | $kgm^2 \times 10^{-6}$ | 25 | 25 | 0 | 1000000 | Motor rotor inertia for rotary motors, or the forcer mass (kg) for linear motors. |
| 28715 | Linear Motor Pole Pitch | mm | 0 | 0 | 0 | 300 | Electrical pitch of the magnets for use with linear motors. (Set to zero for rotary motors). |
| 28717 | Motor Maximum Torque | Nm | 4.67 | 4.67 | 0 | 4000 | Maximum torque available for motor. Maximum force in N for linear motors. |
| 28714 | Motor Constant, Ke | V/krpm or V/m/s | 44.3 | 44.3 | 0 | 800 | Motor voltage constant (Ke). Volts(0-peak). |
| 28718 | Torque Scaling | Nm | 4.67 | 4.67 | 0 | 4000 | Full scale torque available for motor. Maximum force in N for linear motors. |
| 28776 | Encoder Polarity | None | 1 | 1 | 0 | 1 | Positive Encoder direction: 0 for CW, 1 for CCW. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 28678 | Invert Halls | None | 0 | 0 | 0 | 1 | Controls the logic sense of the Hall sensors. Set = 1 to invert the halls. |
| 28775 | Hall Only Commutation | None | 0 | 0 | 0 | 4 | Commutation with incremental encoders: 0 = hall, 2=DC brush mode. |
| 28719 | Motor Ambient Temperature | °C | 25 | 25 | -50 | 250 | Motor ambient temperature used by the software motor thermal model. |
| 28720 | Motor Thermal Resistance | °C/W | 1.4 | 1.4 | 0 | 16 | Temperature rise of the motor winding above motor case temperature per watt of winding power dissipation between the winding and case. |
| 28721 | Motor Thermal Time Constant | Min | 20 | 20 | 0 | 7200 | Time the motor takes to reach 63% of its final temperature given constant power. |
| 28722 | Winding Time Constant | Min | 0.7 | 0.7 | 0 | 3600 | Time for the winding to reach 63% of its final temperature rise above the rest of the motor given constant power. |
| 28679 | Disable Thermal Switches | None | 1 | 1 | 0 | 1 | Thermal Switch Checking: = enable=0 , disable=1. |
| 28716 | Motor Velocity Limit | RPS | 83.3 | 83.3 | 0 | 250 | Maximum velocity of motor in revs/s. Linear motor in meters/s. |
| 28725 | Encoder Commutation Offset | Counts | -0.45 | -0.45 | -1 | 1 | Encoder commutation offset. 1=180 degrees. |
| 28769 | Serial Encoder Valid Bits | | 35 | 35 | 0 | 214783647 | Number of valid bits for serial encoder. Single and multi-turn total. |
| 28771 | Feedback Type | | 5 | 5 | 0 | 10 | Feedback type. 1 = incremental encoder, 5=BiSS |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 28772 | Serial Encoder Valid Turns | | 65535 | 65535 | 1 | 214783647 | Number of supported multi turns for a serial encoder |
| 28800 | BiSS Single Turn bits | | 21 | 21 | 0 | 64 | Number of single turn bits in BiSS frame |
| 28801 | BiSS Multi Turn bits | | 16 | 16 | 0 | 20 | Number of multi turn bits in BiSS frame |
| 28802 | BiSS Status Bit Offset | | 0 | 0 | 0 | 16 | Status bit offset in BiSS frame |
| 28803 | BiSS CRC_Invert | | 0 | 0 | 0 | 1 | Set = 1 if BiSS CRC is not inverted |
| 28804 | BiSS CRC | | 0 | 0 | 0 | 1 | Set =1 to skip BiSS CRC check |
| 28805 | BiSS Status Bit Inversion | | 1 | 1 | 0 | 1 | BiSS Status bit polarity (0=inverted) |
| 28806 | BiSS Position Bits Offset | | 2 | 2 | 0 | 32 | Position data offset in BiSS frame |
| 28807 | BiSS Protocol Type | | 0 | 0 | 0 | 1 | BiSS Protocol type. BiSS C=0, BiSS B=1 |
| 28686 | Brake Relay Delay on Enable | ms | 50 | 0 | 0 | | Delay in milliseconds after DRIVE ON. Brake will remain engaged/holding for delay time. |
| 28687 | Brake Delay after disable | ms | 50 | 0 | 0 | | Delay in milliseconds after DRIVE OFF. Servo loop will continue operating, allowing time for brake to engaged/hold. |

Brake Relay Delay on Enable: Specifies the amount of time that the brake relay will remain asserted after the current is applied to the motor windings when the drive is enabled. This allows torque to build up in the motor while the brake output is off (brake set). This is important in vertical applications where the motor must be able to support the load before the brake is released. This should be set for the disengage/release time of the brake.

Brake Delay after disable: Specifies the amount of time that current will remain in the motor windings after DRIVE OFF issued. This setting is intended to be used in vertical applications, where the brake must be enabled while the motor still has torque so that the load is always supported. This is the complement to the Brake Relay Delay on Enable setting. This should be set for the engage/set time of the brake. This delay will not occur in error condition, only when axis is disabled.

# ACR7xV Status Parameters

The usual scheme for P parameters applies for eight axes here. So Axis1 will be Axis 0 +256, axis2 is +512, etc.

| Parameter Name | Axis0 | Axis1 | Axis2 | Axis3 | Axis4 | Axis5 | Axis6 | Axis7 |
|---|---|---|---|---|---|---|---|---|
| Drive Continuous Current Rating | P28736 | P28992 | P29248 | P29504 | P29760 | P30016 | P30272 | P30528 |
| Drive Maximum Current Rating | P28737 | P28993 | P29249 | P29505 | P29761 | P30017 | P30273 | P30529 |
| Commanded Current | P28738 | P28994 | P29250 | P29506 | P29762 | P30018 | P30274 | P30530 |
| Commanded Torque | P28739 | P28995 | P29251 | P29507 | P29763 | P30019 | P30275 | P30531 |
| Actual Torque | P28740 | P28996 | P29252 | P29508 | P29764 | P30020 | P30276 | P30532 |
| Actual Velocity | P28741 | P28997 | P29253 | P29509 | P29765 | P30021 | P30277 | P30533 |
| Shaft Power, watts | P28742 | P28998 | P29254 | P29510 | P29766 | P30022 | P30278 | P30534 |
| Drive Temperature | P28743 | P28999 | P29255 | P29511 | P29767 | P30023 | P30279 | P30535 |
| Motor Temperature | P28744 | P29000 | P29256 | P29512 | P29768 | P30024 | P30280 | P30536 |
| Bus Voltage | P28745 | P29001 | P29257 | P29513 | P29769 | P30025 | P30281 | P30537 |
| Thermistor temperature | P28746 | P29002 | P29258 | P29514 | P29770 | P30026 | P30282 | P30538 |
| | | | | | | | | |
| Fan On | P28691 | P28947 | P29203 | P29459 | P29715 | P29971 | P30227 | P30483 |
| Custom Product ID | P28692 | P28948 | P29204 | P29460 | P29716 | P29972 | P30228 | P30484 |
| Encoder Position | P28693 | P28949 | P29205 | P29461 | P29717 | P29973 | P30229 | P30485 |
| Current Hall State | P28694 | P28950 | P29206 | P29462 | P29718 | P29974 | P30230 | P30486 |
| Operating Hours | P28695 | P28951 | P29207 | P29463 | P29719 | P29975 | P30231 | P30487 |
| Operating Minutes | P28696 | P28952 | P29208 | P29464 | P29720 | P29976 | P30232 | P30488 |
| Operating Milliseconds | P28697 | P28953 | P29209 | P29465 | P29721 | P29977 | P30233 | P30489 |

# ACR7xV Status 1 Flags

All eight axes have their status parameters in a row rather than offset by increments of 256.

| Parameter Name | Mask | 0x01 | 0x02 | 0x04 | 0x08 | 0x010 | 0x20 | 0x40 | 0x80 |
|---|---|---|---|---|---|---|---|---|---|
| Flag Parameter Code=0x10; Index=0x04 | | P4392 | P4393 | P4394 | P4395 | P4396 | P4397 | P4398 | P4399 |
| | | Axis Number | | | | | | | |
| Status Flags | Bit Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Motor Configuration Warning | 0 | 9472 | 9504 | 9536 | 9568 | 9600 | 9632 | 9664 | 9696 |
| Motor Configuration Error | 1 | 9473 | 9505 | 9537 | 9569 | 9601 | 9633 | 9665 | 9697 |
| Invalid OS Loader | 2 | 9474 | 9506 | 9538 | 9570 | 9602 | 9634 | 9666 | 9698 |
| Max Inductance = 0 | 3 | 9475 | 9507 | 9539 | 9571 | 9603 | 9635 | 9667 | 9699 |
| Rated Speed = 0 | 4 | 9476 | 9508 | 9540 | 9572 | 9604 | 9636 | 9668 | 9700 |
| DPOLE = 0 | 5 | 9477 | 9509 | 9541 | 9573 | 9605 | 9637 | 9669 | 9701 |
| Resistance = 0 | 6 | 9478 | 9510 | 9542 | 9574 | 9606 | 9638 | 9670 | 9702 |
| Ke = 0 | 7 | 9479 | 9511 | 9543 | 9575 | 9607 | 9639 | 9671 | 9703 |
| Continuous Current = 0 | 8 | 9480 | 9512 | 9544 | 9576 | 9608 | 9640 | 9672 | 9704 |
| Peak Current = 0 | 9 | 9481 | 9513 | 9545 | 9577 | 9609 | 9641 | 9673 | 9705 |
| Cont Motor Current > Drive | 10 | 9482 | 9514 | 9546 | 9578 | 9610 | 9642 | 9674 | 9706 |
| Torque Rating > Drive | 11 | 9483 | 9515 | 9547 | 9579 | 9611 | 9643 | 9675 | 9707 |
| Peak Current > Drive | 12 | 9484 | 9516 | 9548 | 9580 | 9612 | 9644 | 9676 | 9708 |
| Inertia = 0 | 13 | 9485 | 9517 | 9549 | 9581 | 9613 | 9645 | 9677 | 9709 |
| Damping = 0 | 14 | 9486 | 9518 | 9550 | 9582 | 9614 | 9646 | 9678 | 9710 |
| Reserved | 15 | 9487 | 9519 | 9551 | 9583 | 9615 | 9647 | 9679 | 9711 |
| Reserved | 16 | 9488 | 9520 | 9552 | 9584 | 9616 | 9648 | 9680 | 9712 |
| Reserved | 17 | 9489 | 9521 | 9553 | 9585 | 9617 | 9649 | 9681 | 9713 |
| Reserved | 18 | 9490 | 9522 | 9554 | 9586 | 9618 | 9650 | 9682 | 9714 |
| Reserved | 19 | 9491 | 9523 | 9555 | 9587 | 9619 | 9651 | 9683 | 9715 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Reserved | 20 | 9492 | 9524 | 9556 | 9588 | 9620 | 9652 | 9684 | 9716 |
| Reserved | 21 | 9493 | 9525 | 9557 | 9589 | 9621 | 9653 | 9685 | 9717 |
| Reserved | 22 | 9494 | 9526 | 9558 | 9590 | 9622 | 9654 | 9686 | 9718 |
| Reserved | 23 | 9495 | 9527 | 9559 | 9591 | 9623 | 9655 | 9687 | 9719 |
| Reserved | 24 | 9496 | 9528 | 9560 | 9592 | 9624 | 9656 | 9688 | 9720 |
| Reserved | 25 | 9497 | 9529 | 9561 | 9593 | 9625 | 9657 | 9689 | 9721 |
| Drive Faulted | 26 | 9498 | 9530 | 9562 | 9594 | 9626 | 9658 | 9690 | 9722 |
| Bridge Hardware Fault | 27 | 9499 | 9531 | 9563 | 9595 | 9627 | 9659 | 9691 | 9723 |
| Bridge Temperature Fault | 28 | 9500 | 9532 | 9564 | 9596 | 9628 | 9660 | 9692 | 9724 |
| Drive Over-voltage Fault | 29 | 9501 | 9533 | 9565 | 9597 | 9629 | 9661 | 9693 | 9725 |
| Drive Under-voltage Fault | 30 | 9502 | 9534 | 9566 | 9598 | 9630 | 9662 | 9694 | 9726 |
| Bridge Foldback Warning | 31 | 9503 | 9535 | 9567 | 9599 | 9631 | 9663 | 9695 | 9727 |

# *ACR7xV Status 2 Flags*

All eight axes have their status parameters in a row rather than offset by increments of 256.

| **Parameter Name** | **Mask** | **0x01** | **0x02** | **0x04** | **0x08** | **0x010** | **0x20** | **0x40** | **0x80** |
|---|---|---|---|---|---|---|---|---|---|
| Flag Parameter Code=0x10; Index=0x04 | | P4408 | P4409 | P4410 | P4411 | P4412 | P4413 | P4414 | P4415 |
| | | Axis Number | | | | | | | |
| Status Flags | Bit Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Power Regeneration Fault | 0 | 9984 | 10016 | 10048 | 10080 | 10112 | 10144 | 10176 | 10208 |
| Reserved | 1 | 9985 | 10017 | 10049 | 10081 | 10113 | 10145 | 10177 | 10209 |
| Drive Temperature Fault | 2 | 9986 | 10018 | 10050 | 10082 | 10114 | 10146 | 10178 | 10210 |
| Motor Thermal Model Fault | 3 | 9987 | 10019 | 10051 | 10083 | 10115 | 10147 | 10179 | 10211 |
| Motor Temperature Fault | 4 | 9988 | 10020 | 10052 | 10084 | 10116 | 10148 | 10180 | 10212 |
| Bad Hall State | 5 | 9989 | 10021 | 10053 | 10085 | 10117 | 10149 | 10181 | 10213 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Feedback Failure | 6 | 9990 | 10022 | 10054 | 10086 | 10118 | 10150 | 10182 | 10214 |
| Drive Disabled | 7 | 9991 | 10023 | 10055 | 10087 | 10119 | 10151 | 10183 | 10215 |
| Over Current Fault | 8 | 9992 | 10024 | 10056 | 10088 | 10120 | 10152 | 10184 | 10216 |
| Power Regeneration Warning | 9 | 9993 | 10025 | 10057 | 10089 | 10121 | 10153 | 10185 | 10217 |
| Shaft Power Limited Warning | 10 | 9994 | 10026 | 10058 | 10090 | 10122 | 10154 | 10186 | 10218 |
| Reserved | 11 | 9995 | 10027 | 10059 | 10091 | 10123 | 10155 | 10187 | 10219 |
| Reserved | 12 | 9996 | 10028 | 10060 | 10092 | 10124 | 10156 | 10188 | 10220 |
| Reserved | 13 | 9997 | 10029 | 10061 | 10093 | 10125 | 10157 | 10189 | 10221 |
| VQ exceed bus voltage | 14 | 9998 | 10030 | 10062 | 10094 | 10126 | 10158 | 10190 | 10222 |
| Low Voltage at Enable | 15 | 9999 | 10031 | 10063 | 10095 | 10127 | 10159 | 10191 | 10223 |
| Control Power Active (IPA) | 16 | 10000 | 10032 | 10064 | 10096 | 10128 | 10160 | 10192 | 10224 |
| Alignment Error | 17 | 10001 | 10033 | 10065 | 10097 | 10129 | 10161 | 10193 | 10225 |
| Hardware Error | 18 | 10002 | 10034 | 10066 | 10098 | 10130 | 10162 | 10194 | 10226 |
| Internal Error | 19 | 10003 | 10035 | 10067 | 10099 | 10131 | 10163 | 10195 | 10227 |
| Encoder Read Fault | 20 | 10004 | 10036 | 10068 | 10100 | 10132 | 10164 | 10196 | 10228 |
| Reserved | 21 | 10005 | 10037 | 10069 | 10101 | 10133 | 10165 | 10197 | 10229 |
| Encoder Loss Fault | 22 | 10006 | 10038 | 10070 | 10102 | 10134 | 10166 | 10198 | 10230 |
| Reserved | 23 | 10007 | 10039 | 10071 | 10103 | 10135 | 10167 | 10199 | 10231 |
| Drive Param Error | 24 | 10008 | 10040 | 10072 | 10104 | 10136 | 10168 | 10200 | 10232 |
| Torque Enable Fault (IPA) | 25 | 10009 | 10041 | 10073 | 10105 | 10137 | 10169 | 10201 | 10233 |
| Torque Enable Open (IPA) | 26 | 10010 | 10042 | 10074 | 10106 | 10138 | 10170 | 10202 | 10234 |
| Torque Enable Health Event (IPA) | 27 | 10011 | 10043 | 10075 | 10107 | 10139 | 10171 | 10203 | 10235 |
| Reserved | 28 | 10012 | 10044 | 10076 | 10108 | 10140 | 10172 | 10204 | 10236 |
| Reserved | 29 | 10013 | 10045 | 10077 | 10109 | 10141 | 10173 | 10205 | 10237 |
| Reserved | 30 | 10014 | 10046 | 10078 | 10110 | 10142 | 10174 | 10206 | 10238 |
| Reserved | 31 | 10015 | 10047 | 10079 | 10111 | 10143 | 10175 | 10207 | 10239 |