QuickBuilder™ Reference Guide

Doc. No. 951-530020-012

© 2018 Control Technology Corp.

25 South Street
Hopkinton, MA 01748

Phone: 508.435.9595
Fax: 508.435.2373

Thursday, April 12, 2018

Control Technology Corp.

Table of Contents

1 Chapter 1: QuickBuilder Overview ... 8
- Target Systems ... 9
- QuickBuilder projects ... 11
- Resource Manager (RM) ... 11
- SFC Window ... 14
- Step Editor & Assistant ... 14
- Task Editor ... 15
- Translation ... 19
  - Modes ... 20
  - Target Platforms ... 21
  - Program Download ... 22
  - Reserved Words ... 22
- Debugger Mode ... 23
  - Watch Windows ... 24
  - Online Variable Monitoring ... 24
  - Online Status & Control Monitor/Debugger ... 27
  - Breakpoints ... 29
  - MSB Status/Control Monitor Fault Processing ... 31
  - MSB Monitor ... 33
- Project Manager ... 37
- FTP Explorer ... 37
- Global and Local Resources ... 39
- Intelligent Prompting ... 42
- Tech Tips ... 45

2 Chapter 2: Library Manager ... 47
- Creating a Library ... 47

3 Chapter 3: QuickStep 4 (QS4) ... 52
- QS4 SFC Graphical Constructs ... 52
  - SFC Diagram ... 53
  - QS4 Task Definition ... 54
  - QS4 Event Definition ... 54
  - QS4 Function Definition ... 56
  - QS4 Step ... 56
  - QS4 Goto ... 57
  - QS4 Do Step ... 57
  - QS4 Begin Step ... 59
  - QS4 Decision Step ... 59
  - QS4 Done Step ... 60
  - QS4 'C' Step ... 60
  - Motion Overview & Sequence Blocks ... 61
- QS4 Resources ... 65
  - Symbolic Names and Resources ... 65
  - Resource Declarations ... 66
  - Vector and Table Declarations (Arrays) ... 66
  - Constants & Literals ... 67

© 2018 Control Technology Corp.

Indirect Variables ... 69
Tasks and Steps ... 71
QS4 Functions and Expressions ... 72
Expressions ... 72
Numerical Functions ... 73
String Functions ... 76
Bit Functions ... 77
Special Functions ... 77
QS4 System Variables ... 79
$TASKTIMER ... 79
$DINPUTS[ ] ... 80
$DOUTPUTS[ ] ... 80
$REGISTERS[ ] ... 81
$TRIGGER ... 81
$CBITS[ ] ... 82
$CVARS[ ] ... 82
$TASKPRIORITY ... 82
$CURRENT_TASKPRIORITYLEVEL ... 83
$TASKHANDLE ... 84
MSB System Variables ... 84
QS4 Statements ... 86
QS4 Statement syntax ... 86
QS4 Editor Color Codes ... 86
Assignment (numeric) ... 86
Assignment (string) ... 87
Store ... 88
Set ... 88
Setbit, Clrbit ... 88
Goto ... 89
Call, Return ... 89
If/Then/Else ... 90
While ... 92
Repeat/Until ... 92
For ... 93
Break ... 94
Continue ... 94
Delay ... 95
Timeout ... 95
When ... 95
Enable, Disable (Event) ... 96
Do ... 96
Begin ... 97
Cancel ... 97
Done ... 98
Start ... 98
Stop ... 98
Soft Counters ... 98
Rotate, Shift Flags ... 99
4 Chapter 4: Importing QuickStep 2/3 Projects ... 100
Datatables ... 101
Motion Control ... 102
Variables ... 105
Soft Counters ... 105
Reserved Words Error Search ... 106

© 2018 Control Technology Corp.

Importing ... 107
5 Chapter 5: BACnet/IP ... 117
BACnet Volatile Tables ... 117
BACnet System Variables ... 120
BACnet Explorer ... 123
6 Chapter 6: EtherCAT Explorer ... 126
Status Window ... 126
Properties ... 131
Log Buffer Timings ... 133
User Options ... 134
File Over EtherCAT (FOE) ... 137
7 Chapter 7: Windows 7 and 10 Support ... 141
8 Appendix A: Shortcut Keys ... 144
9 Appendix B: Known Anomolies & Warnings ... 145
10 Appendix C: Training ... 147

1 Chapter 1: Introduction and Overview ... 150
Guide to Symbols ... 150
Brief Overview of Motion Control ... 151
Servo Motor Applications ... 151
Stepper Motor Applications ... 152
Brief Overview of M3-40/41 Motion Module Features ... 153
M3-40 & M3-41/IncentiveECAT Motion Module Features ... 155
Special M3-40 I/O Functions ... 156
Drives & M3-41 IO ... 156
QuickBuilder Motion Control Features ... 157
IO Assignments ... 158
IO Assignments - M3-40A ... 158
IO Assignments - M3-40B ... 159
IO Assignments - M3-40C ... 160
IO Assignments - M3-41A ... 161
2 Chapter 2: Motion Architecture ... 162
QuickBuilder ... 163
QuickStep ... 164
QuickMotion ... 165
Adding Motion to the 5300/Incentive Application ... 166
The Axis Module ... 167
The Axis Object ... 168
The Motion Sequence Block ... 169
Controlling Motion from QuickStep ... 170
QS4 start Statement ... 170
QS4 stop Statement ... 170
Motion Architecture Summary Diagram ... 171
3 Chapter 3: QuickMotion Axis Setup ... 174
Axis Properties ... 175
Basic Tuning ... 176
Fine Tuning ... 177
Tuning an axis (5300 M3-40 Only) ... 178
4 Chapter 4: QuickMotion Programming ... 180

© 2018 Control Technology Corp.

Operating Modes ... 180
Expressions ... 181
Utility Statements ... 182
Program Flow Statements ... 187
Set Statements ... 191
Common bits and variables ... 194
I/O Statements ... 197
Simple Motion ... 205
Gearing ... 215
Position Capture & Registration ... 219
S-Curve ... 221
Linear and Circular Interpolation (Vectors) ... 223
5 Chapter 5: Camming and Data Tables ... 224
Loading Tables ... 227
Using Tables for Spline/CAM ... 231
Accessing Table Data ... 235
Diagnosing Table Issues ... 236
Microsoft Excel as Table Data ... 237
Virtual Master ... 238
Broadcasting (M3-40 only) ... 239
Segmented Moves and Examples ... 240
Concept ... 240
Commands ... 241
Examples ... 243
6 Chapter 6: Motion Variables ... 248
QuickMotion User-defined Variables ... 248
QuickMotion Pre-defined Variables ... 250
Host Register Access ... 280
7 Chapter 7: Quickstep Support ... 282
Registers ... 283
Quickstep Variables ... 288
Input Mapping (M3-40 Only) ... 291
8 Chapter 8: Fault Codes & MSB Debugging ... 292
Fault Codes ... 293
MSB Status/Control Monitor Fault Processing ... 297
MSB Monitor ... 299
9 Appendix: Sample Code ... 302
10 Appendix: Command Hyperlinks ... 306

1 Chapter 1: Overview ... 311
2 Chapter 2: The QB PID Object ... 313
Features ... 313
PID Loop Algorithm ... 314
PID Object Setup ... 315
PID Object Properties ... 318
Accessing Properties in QS4 code ... 319
3 Appendix A: PID Loop Tuning ... 321

© 2018 Control Technology Corp.

1 Chapter 1: Overview 324
2 Chapter 2: QuickScope and QuickView Features 325
Invoking QuickScope 325
Toolbar Summary 327
Status Bar Summary 327
Connecting to a controller 328
Setting up traces 329
Capturing Data 331
Evaluating Data 332
Zoom 333
A and B Cursors 334
Creating a PDF file 335
Creating an Excel Spreadsheet 336
QuickView 336
Multiple Windows 338
Index 339

© 2018 Control Technology Corp.

QuickBuilder Reference Guide

Copyright © 2004 - 2018 Control Technology Corp. All Rights Reserved.

Control Technology Corp.
25 South Street
Hopkinton, MA 01748
Phone: 508.435.9595 • Fax 508.435.2373

Document No. 951-530020-010

⚠ **WARNING:** Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.

The information in this document is subject to change without notice. The software described in this document is provided under license agreement and may be used and copied only in accordance with the terms of the license agreement. The information, drawings, and illustrations contained herein are the property of Control Technology Corporation. No part of this manual may be reproduced or distributed by any means, electronic or mechanical, for any purpose other than the purchaser's personal use, without the express written consent of Control Technology Corporation. Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions.

See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

1 Chapter 1: QuickBuilder Overview

Introduction to the QuickBuilder architecture and terminology

QuickBuilder is CTC's innovative graphical development environment for the Incentive (5300 and embedded PC) series of automation controllers. It combines all the aspects of an automation project into one easy to use desktop application. This holistic approach to solving automation projects leads to quicker machine startups and simpler ongoing maintenance.

QuickBuilder is based upon concepts from the IEC 61131 (1131) standard and CTC's proven automation state language called QuickStep. A QuickBuilder project contains all of the relevant information pertaining to a particular automation application. The five main elements of the QuickBuilder desktop are:

1. The Resource Manager (RM), which allows the automation developer to define how and where data and physical I/O exist in one or more CTC automation controllers. It features a hierarchical, resource definition view that is used to describe the data used in the project, where the data exists (physical-to-logical mapping) as well as "typing" information for the data.

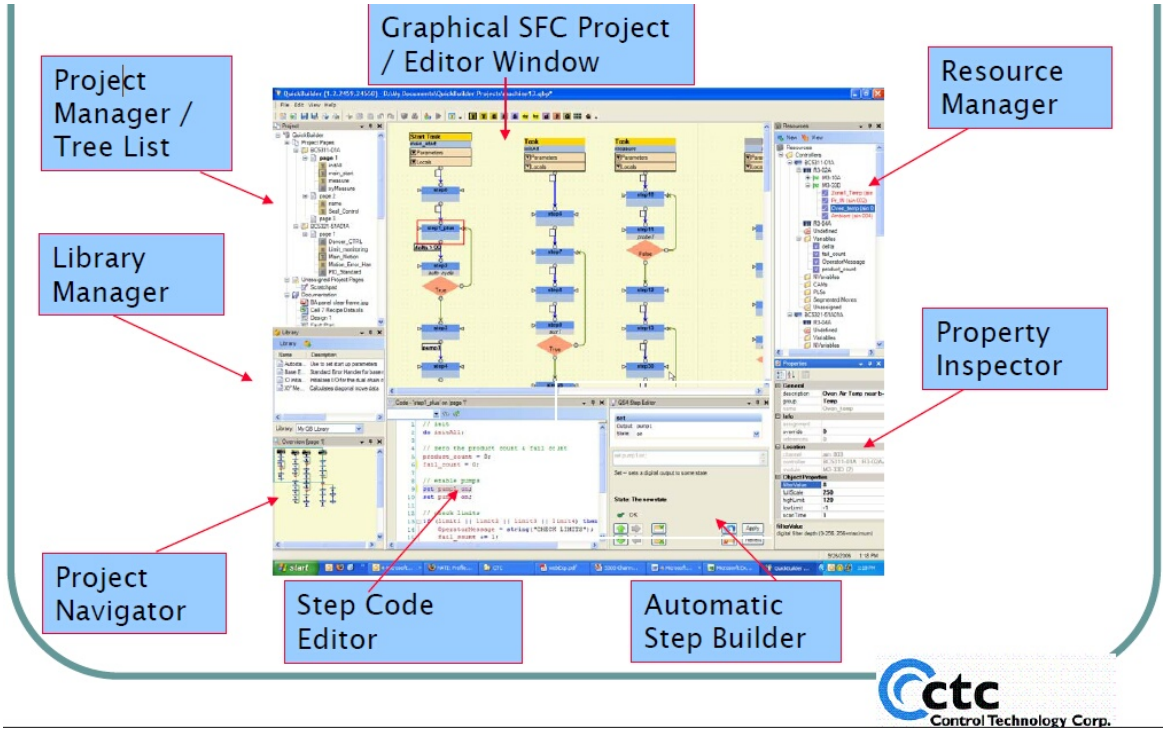
2. The graphical Sequential Function Chart (SFC) that describes the overall logic flow of the project. The SFC is composed of logically connected graphical constructs parented from tasks, functions and events, as well as their underlying logic steps.

3. The QuickStep Editor is where logic is entered or edited within the steps of a QuickBuilder project. A structured text language known as QuickStep4 (QS4) is used to program the logic of the steps that are contained within a task function or event. The QS4 text can either be freely typed into the editor or automatically entered into the editor from menu-driven pick lists.

4. The Project Manager provides a hierarchical view of the major program elements in a familiar tree structure. Using the tree's drill-down capability, even large multi-controller projects are easy to navigate. The project manager also organizes other useful information such as debug watch windows and project documentation files.

5. The Library Manager allows individuals and teams to easily share and re-use common code elements. Any portion of a project from a step, to a series of steps, to a task, to an entire page can be saved as a library element.

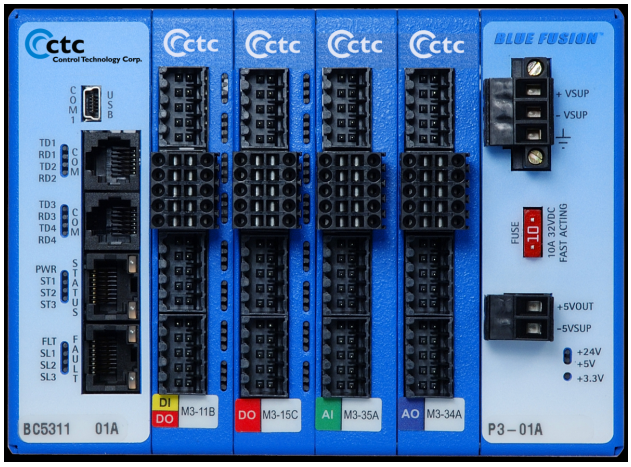
Note: Appendix C contains training slides which can enhance your understanding of QuickBuilder. These slides are only available when using the 'Help' menu option within QuickBuilder, not when this document is a separate PDF.



1.1 Target Systems

QuickBuilder is used for program development on two main automation platforms that are fully compatible; the CTC 5300 PLC controller and IncentivePLC/ECAT for embedded PC automation.

Series 5300 programmable automation controllers are self-contained compact control systems featuring high-density I/O and an impressive suite of capabilities. With such powerful integrated capabilities as high-performance motion control, strong communications and data handling provisions, and a high-level automation-specific programming language, this series of product has become a favorite with fast-moving technology companies for their automation projects.



These systems are programmed using the QuickBuilder programming environment, a comprehensive programming tool encompassing programming, library creation, system configuration, debugging and monitoring. It fully

supports all QuickBuilder language capabilities, which include multitasking, asynchronous events, user functions with parameters, and a modified state language structure that greatly improves program readability and performance. The Series 5300 can operate standalone or in a distributed network, and is supported by a broad range of over 40 I/O modules, allowing you to tailor your I/O complement to specific device requirements without additional external hardware.

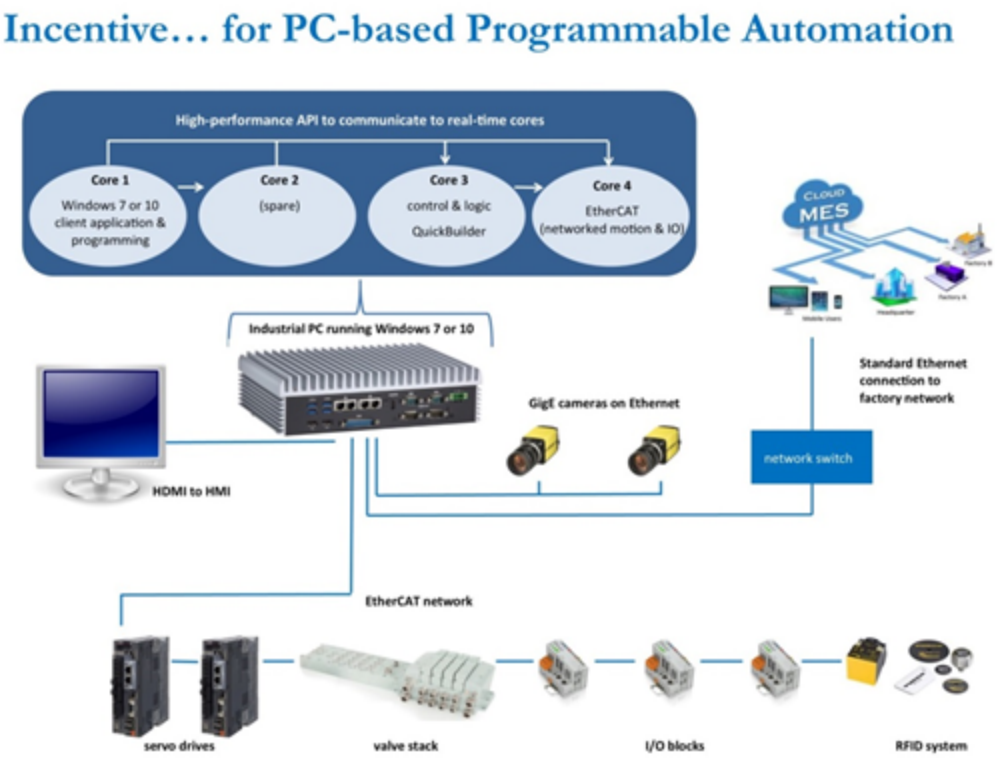
The 5300 Controller QuickBuilder run-time was ported to the Windows® PC where it is known as Incentive. A virtual, soft 5300 PLC (Incentive) can now execute on multiple platforms allowing for versatility in your automation decisions. The Incentive family of software components runs on your standard PC hardware and creates a powerful and extensible control system for your automation project. The three primary members of this family are:

• **IncentivePLC** – a fast, multitasking controller offering powerful constructs to quickly implement a control program. IncentivePLC is programmed using the QuickBuilder programming environment, which can run either locally or on a remote system.

• **IncentiveAPI** – this .Net Managed API of high-level automation functions lets your code, written in C#, C++ or VB.Net, interact with your IncentivePLC control program or directly with external sensors and actuators via IncentiveECAT, both locally and remotely over a network.

• **IncentiveECAT** – a software-based EtherCAT master controller, IncentiveECAT runs on one core of your multicore PC platform and uses a standard Ethernet port to communicate with up to 64 servo axes, or a combination of motion drives and I/O. Use it in conjunction with IncentiveAPI to give your own code easy access to controlling the physical world, or add IncentivePLC to build a complete PC-based control system.

These modules run as processes on dedicated real-time CPU cores in parallel with normal Windows programs. Unlike the 5300 controller, Incentive is dedicated to controlling all motion and IO over an EtherCAT network via its EtherCAT Master, IncentiveECAT. IncentiveECAT originated from the 5300 M3-41 EtherCAT Master module, ported to a PC environment. Since both the 5300 and Incentive originated from the same code base they are both fully compatible and use the same QuickBuilder programming environment. QuickBuilder generates 'C' code when translated, programs loaded on the PC are then compiled using Microsoft Visual Studio tools, while the 5300 uses an embedded ARM gnu compiler, thus a single setting within your QuickBuilder program determines the target system.



1.2 QuickBuilder projects

A QuickBuilder project is composed of:

1. Global and local definitions that specify symbolic representations of storage ("registers") or physical resources such as digital inputs, digital outputs or the like;

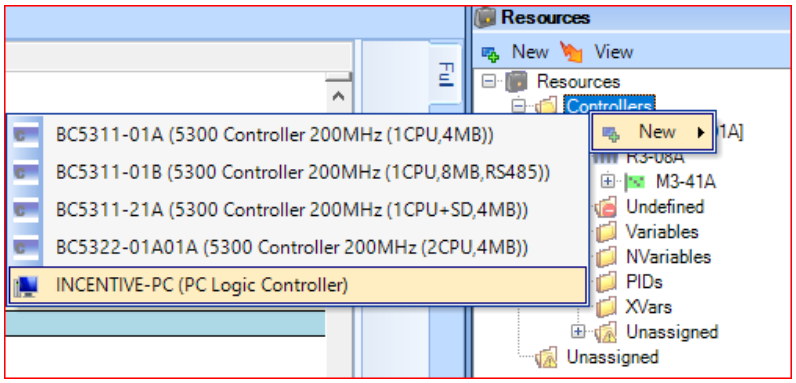
2. Global constant definitions that specify symbolic representations of a value that do not change during program execution;

3. One or more controllers – each of which has one or more pages containing graphical diagrams that define logic and multi-tasking relationships. These diagrams are parented from tasks, functions and events and are programmed in QS4.

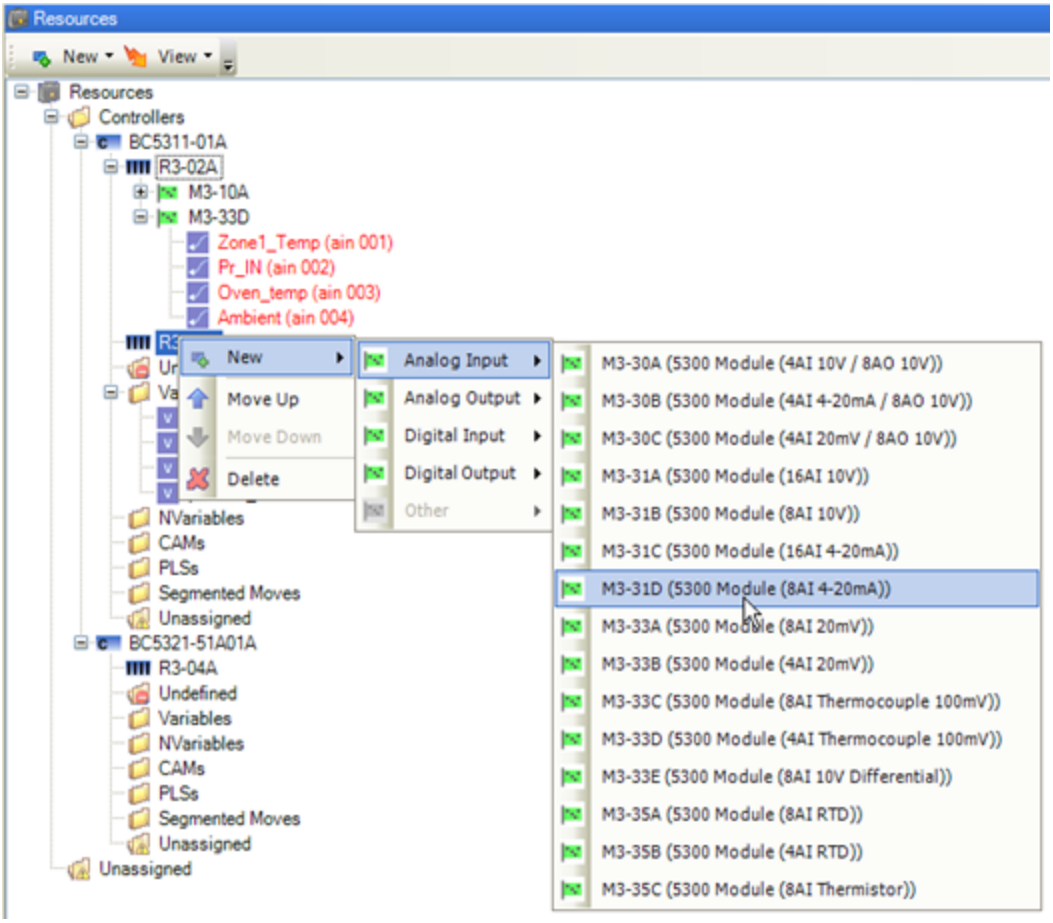
1.3 Resource Manager (RM)

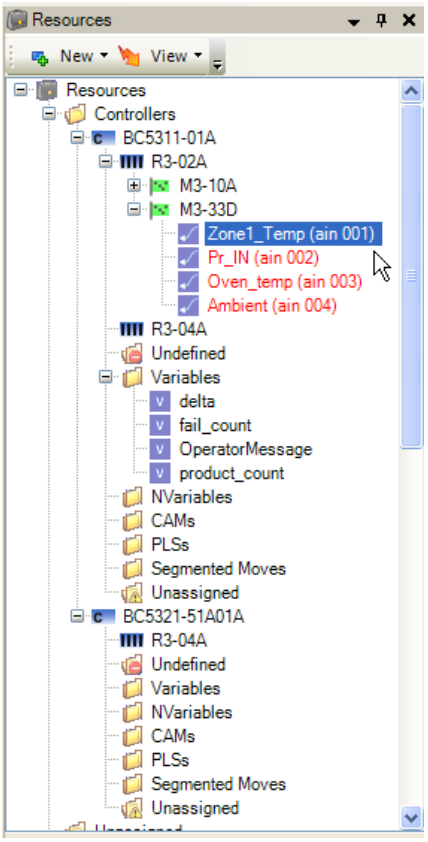
All of the physical and logical attributes of the controller are contained in the Resource Manager (RM). Instead of burying this important information within the Project Manager, QuickBuilder uses a dedicated window for resource management. This not only provides a clearer view of the project, but it significantly speeds program development. Using the menus and right click functionality, it's easy to set up and configure controllers for the application.

Multiple controller types are available. That selected is only for reference as each can be translated for the other's platform. If only PC based solutions will be deployed it is recommended that the INCENTIVE-PC be selected.



Resources are added to a controller starting with the rack and then filling the rack with the proper IO modules.





Maximum capacities for the Resource Manager are:

Parameter	Max capacity
Digital Inputs	1024
Digital Outputs	1024
Analog Inputs	256
Analog Outputs	256
PID loops	256
Modules per CPU	32
Racks per CPU	4
Active tasks (including Tasks, Functions and Events)	96
Volatile integers	485
Volatile floats, strings and arrays: Array cells holding integers or floats Array cells holding strings	600 > 250,000 >40,000

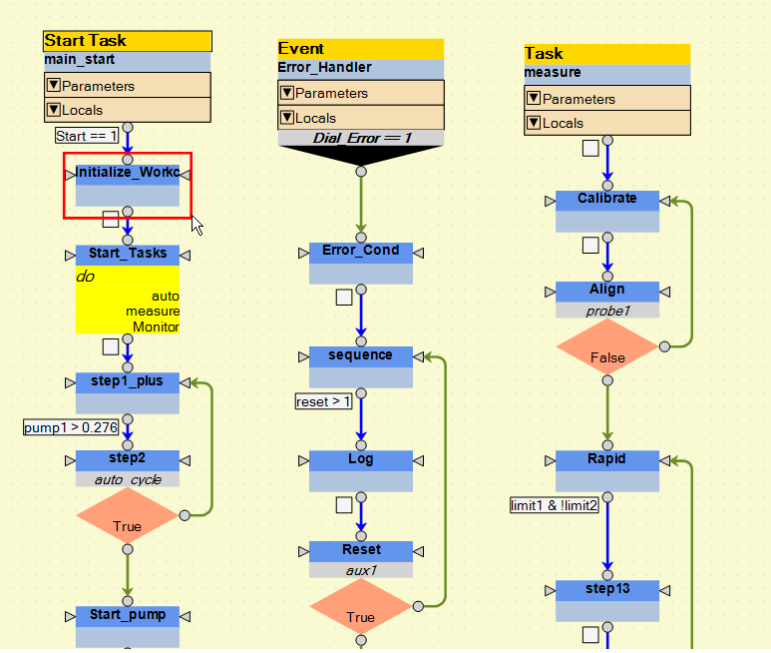
Non-volatile integers	4000
Non-volatile floats, strings, and arrays[1]	100

⚠ Non-volatile arrays are stored to NVRAM as part of the file system. It is suggested you use the save datatable script commands for larger arrays and tables since each element or cell in an array is 256 Bytes. (a 10 x10 table would occupy 25K of NVRAM space).

⚠ Integers - 32 bit signed, Floats - float32 (32 bit single precision float) float64 (64 bit double precision), strings 223 bytes maximum.

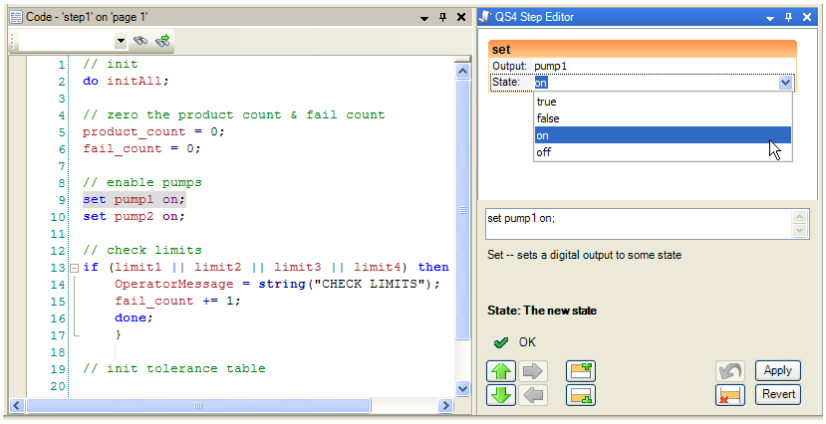
1.4 SFC Window

The SFC Window is where the program development takes shape. Using flowcharting techniques, the major application elements are arranged according to task. Under the tasks are steps that can easily be altered, moved, cloned, or deleted. When a step is highlighted, it is instantly linked to the code window.

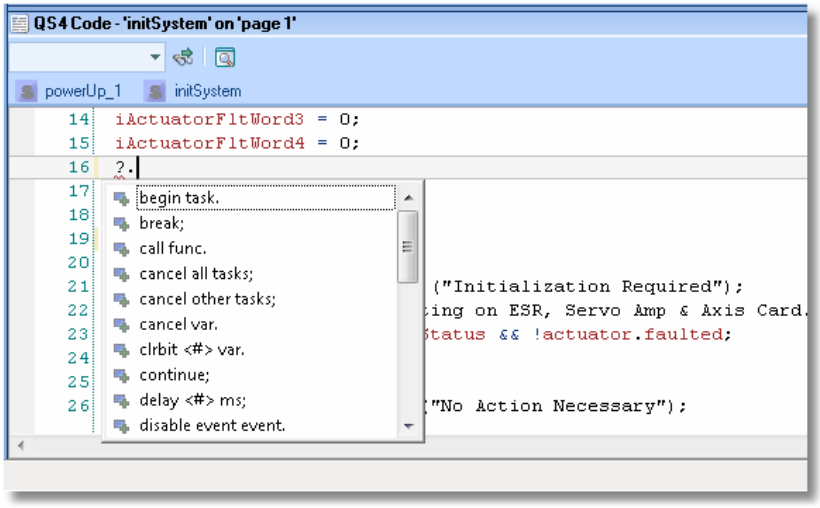


1.5 Step Editor & Assistant

Once a step is highlighted in the graphical SFC Window, the actual instructions and logic for that step can be created and/or edited in the editing window. Here you have two options: For novice programmers there is an auto Step Assistant Editor that walks the user through the command selection and completion. The resulting code is automatically inserted into the left side code editor window. The Step Assistant Editor is typically not shown in the default configuration and has be included for legacy users. A more powerful full screen editor is available, detailed later in this manual.



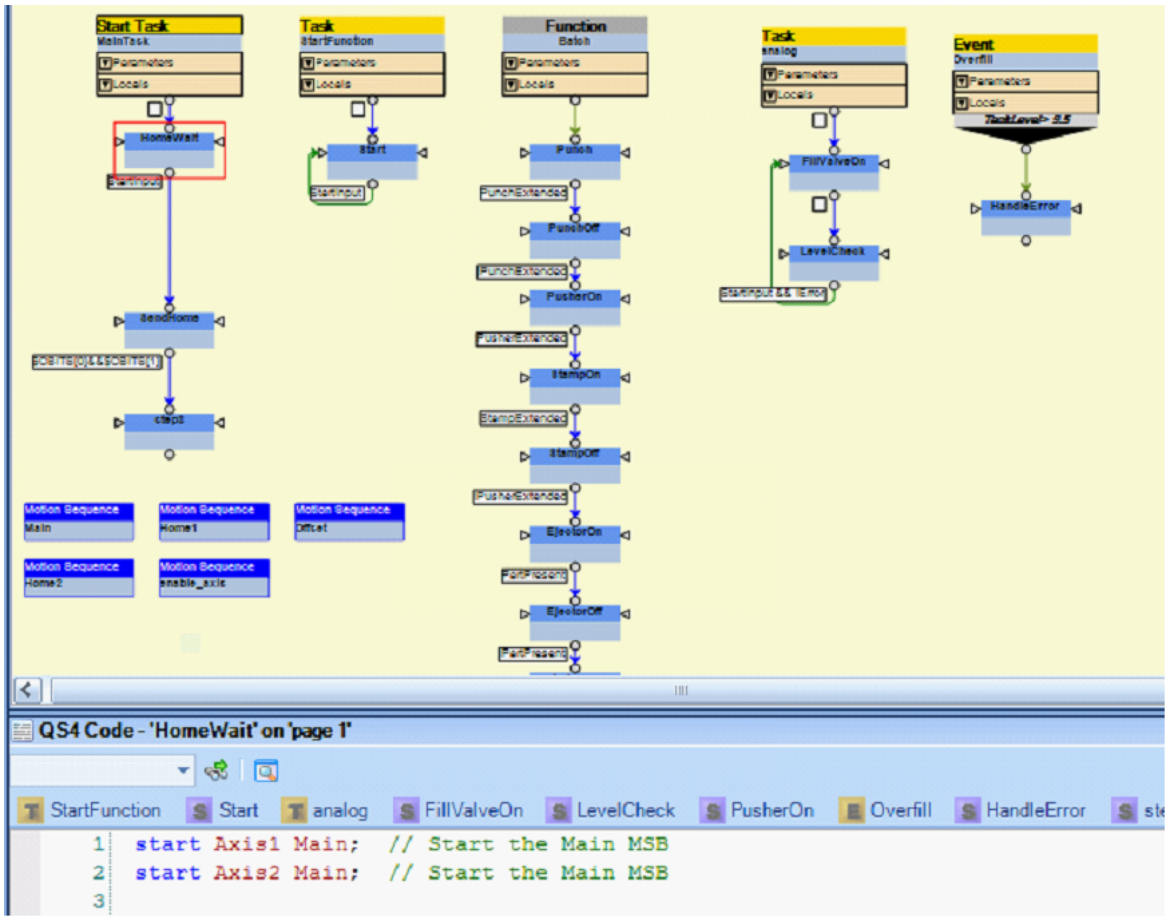
The Step Assistant Editor It may be opened using the View->Step Editor menu option. As an alternative intellihelp is available which allows you to work solely in the text editor and use special key sequences to view available commands, variables, etc. Reference the Intelligent Prompting section for further details.



⚠ Refer to the 'Editor & Debugger Mode' section for enhanced editing features.

1.6 Task Editor

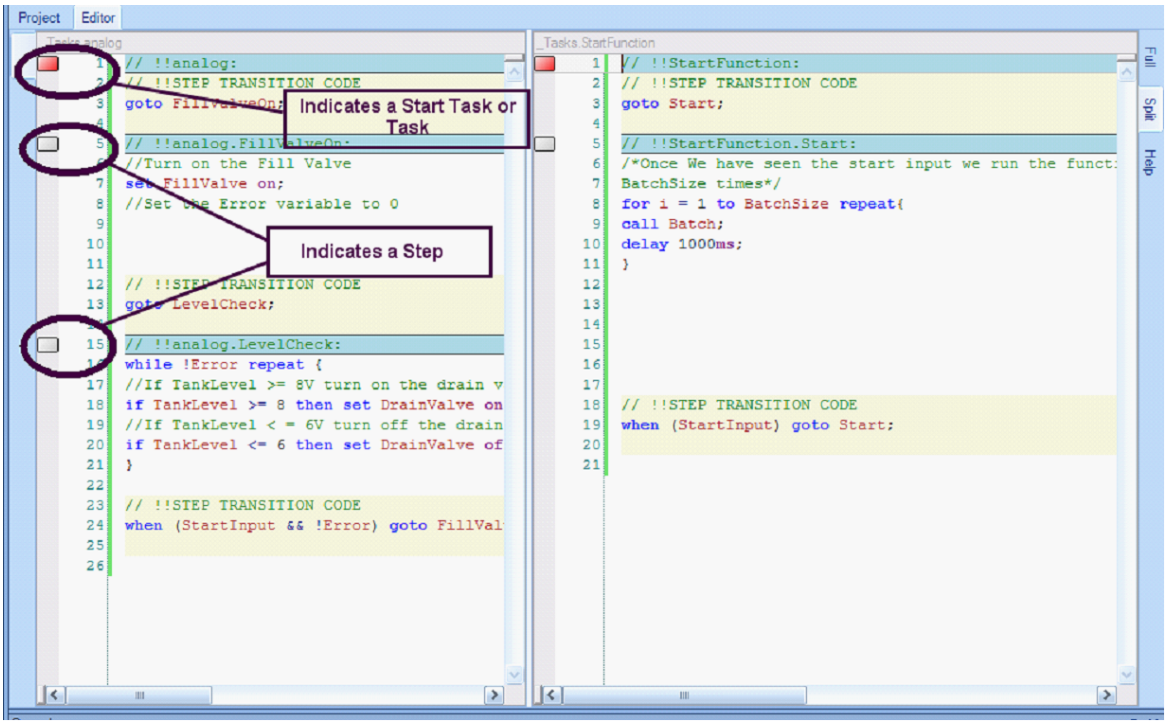
QuickBuilder consists of two editor modes, Project and Editor. Project mode is similar to the screen shown below where the graphical flow chart representation is depicted, you can click on a step and then edit the code in the window at the bottom of the screen.



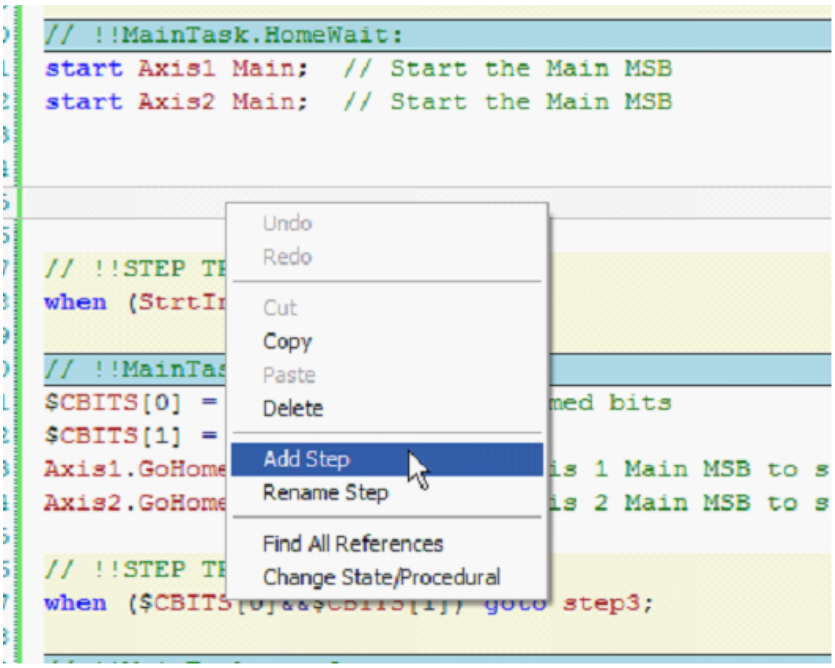
The tab at the top of the flow chart window also shows Editor. Selecting that converts you to Editor mode. Editor mode offers some very powerful features:

1. the ability to view the code from two different tasks at the same time;
2. view help by simply double clicking an instruction, in the help tab;
3. online monitoring of variables;
4. MSB online monitoring and debug.

Below shows the Editor with the 'Split' tab selected:



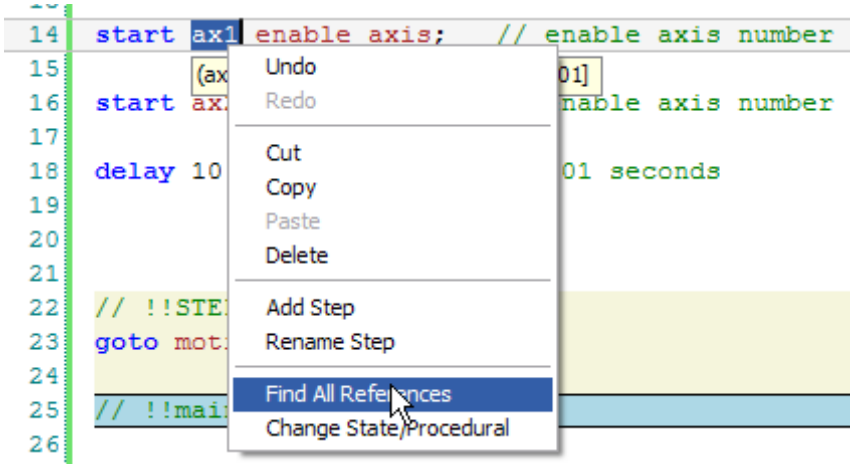
You can even add steps while in the Editor by issuing a right click in the Editor Step. Selecting 'Add Step' inserts a new step below the step you have selected.



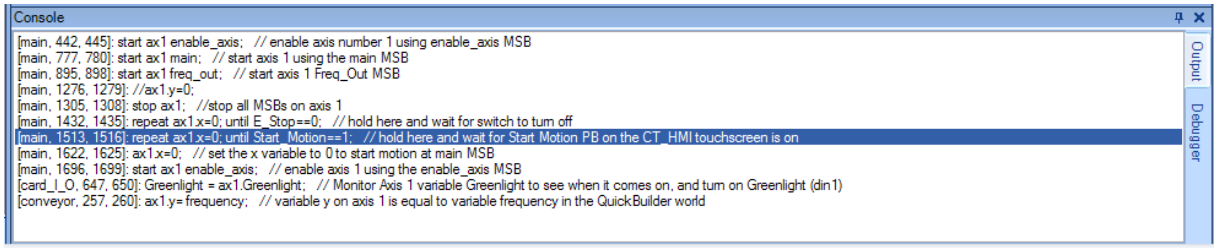
You can create code in the new step and rename it as desired.

```
10  // !!MainTask.HomeWait:                                    <Procedural>
11  start Axis1 Main;  // Start the Main MSB
12  start Axis2 Main;  // Start the Main MSB
13
14
15
16
17  // !!STEP TRANSITION CODE
18  when (StartInput) goto SendHome;
19
20  // !!MainTask.step1:                                       !![QuickBuilde
21  set StatusLight on;
22  set Actuator off;
23  delay 500 ms;
24
25  // !!MainTask.SendHome:                                    <Procedural>
26  $CBITS[0] = 0;   // clear the homed bits
27  $CBITS[1] = 0;
28  Axis1.GoHome = 0;  //Tell the Axis 1 Main MSB to start homing Axis 1
```

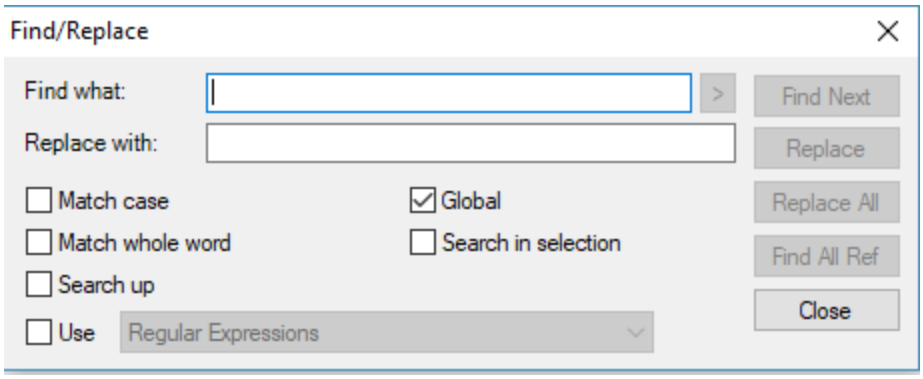
Other useful features such as 'Find All References' allows you to find all references to the highlighted text.



The result is displayed in the console window below the editor. Double clicking on a line of the result will make that step current an place your cursor within that area of the editor.

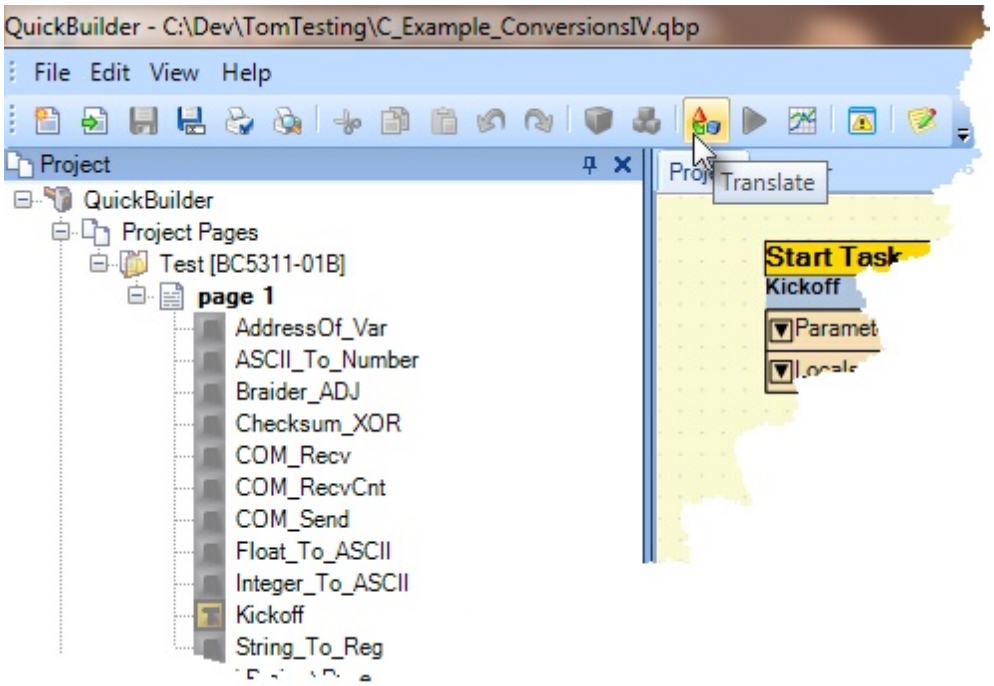


You can also use the Cntrl-F keyboard sequence to invoke the find form, further refining a search based on case, local, global, etc.

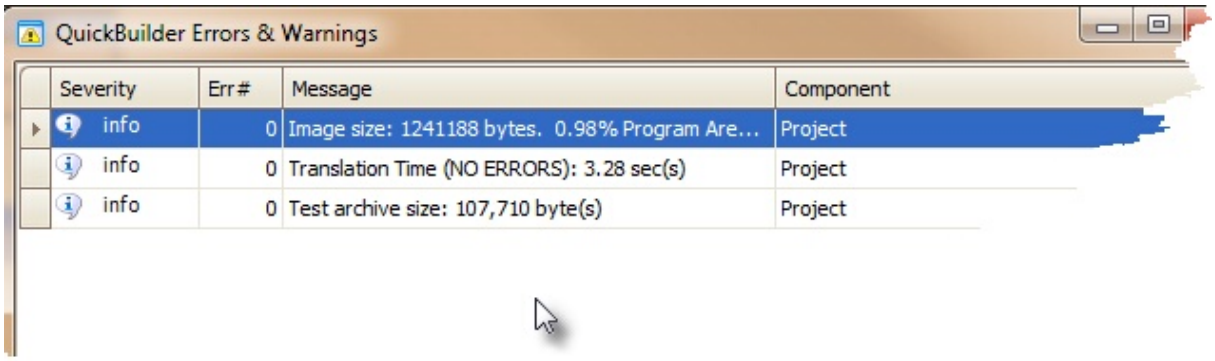


1.7 Translation

After programs are created they must be translated. The translation generates the compiled 'C' code that is downloaded to the controller for execution. All errors must be corrected prior to download. To translate a program select the translate button:



A window will appear in the center of the screen listing any errors or none as shown below:



1.7.1 Modes

There are 5 translation modes available within QuickBuilder:

Legacy – The original translation mode used by older QuickBuilder programs, prior to 11/2010. This mode generated 'C' code and interpreted Quickstep code. It is no longer supported and provided for legacy applications that need to be built. It is recommended that all Legacy programs be built with 'Optimized_debug' for a significant performance improvement. Many new instructions are not supported in Legacy mode and an error will be generated during translation.

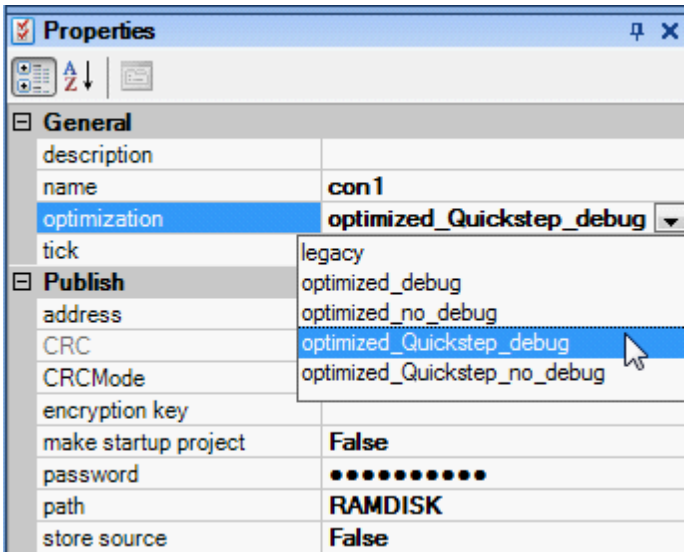
Optimized_debug – The **default** QuickBuilder translation mode which generates full 'C' compiled code along with debug information useful for breakpoints and monitoring program execution. Step instructions execute sequentially as in a procedural program.

Optimized_no_debug – Same as "Optimized_debug" except the debug information has been removed in order to make a smaller program image. Some large programs have problems fitting in the base RAMDISK size and need to remove the debug information as a cost reduction versus expanding to a larger RAMDISK.

Optimized_Quickstep_debug – The default Quickstep Import translation mode (automatically set when imported). State programming where the assignment instructions are executed only on the first loop of the step and a branch out of the step must be by a specified 'goto' instruction. Debug information is included in the program file image.

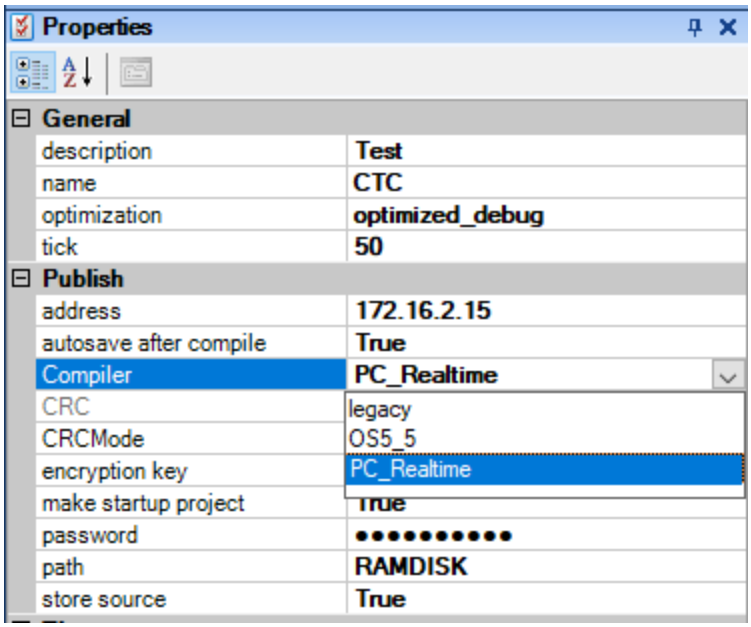
Optimized_Quickstep_no_debug – The same as "Optimized_Quickstep_debug" except the debug information has been removed to reduce file storage requirements.

The desired translation mode is selected under the controller properties:



1.7.2 Target Platforms

Currently there are three possible target platforms available within QuickBuilder, each using a different compiler setting.



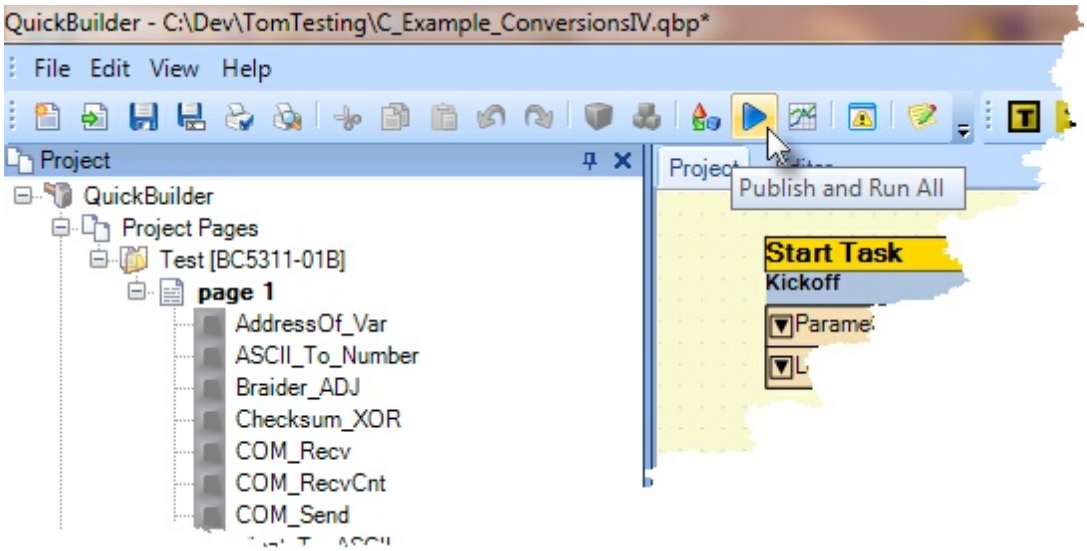
legacy - This is for very old programs and not used anymore.

OS5_5 - This is for the 5300 Controller which is ARM based and uses the gnu 3.4 compiler.

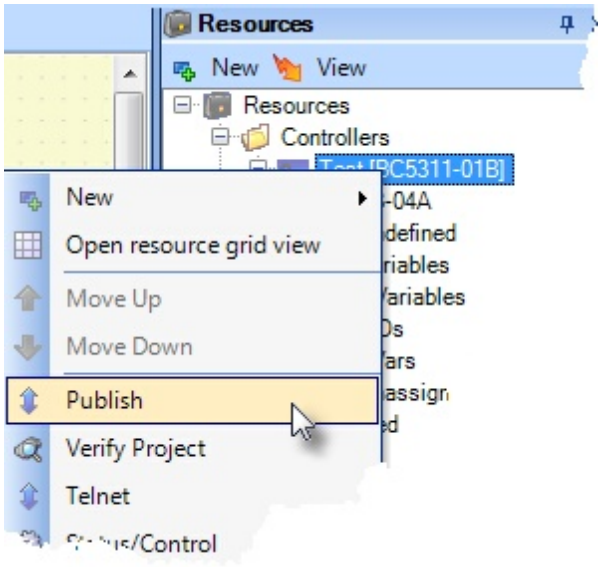
PC_Realtime - This is for the Incentive embedded PC platform and used to compile programs for x86 processors using the Microsoft Visual Studio Community compiler 2013 and greater.

1.7.3 Program Download

Once a program have been successfully translated it can be downloaded to the controller using the 'Publish and Run All' button. The IP address provided in the properties section of the controller will be used as the destination.



If more than one controller is listed all will be downloaded. If only one is desired, right click the controller and select the 'Publish' option on the menu.



1.7.4 Reserved Words

When entering instructions or naming steps, within QuickBuilder, there are a number of reserved word tokens which if used will generate a Translation error. The tokens currently are as follows:

Instructions:

all begin break by
call cancel clrbit clrebit continue scount const
delay disable do done timeout
else enable event
for goto if next other
repeat return
set setbit setebit start stop store sync FG BG slewed
tasks then to until when while with
softstop hardstop maxspeed position up down cw ccw
profile zero motion search and accel P I D servo at reset turn deadband of rotate shift

Functions:

sin cos tan asin acos atan sinh cosh tanh atan2
abs exp log log10 sqrt pow hypot
ceil floor frac sign sign2 len
min max fmod
left right trim mid padl padr compare find
bit addr isdone string
_servoInfo _rol _ror

Constants:

false true FALSE TRUE on off ON OFF pi PI ms
BIT0 BIT1 BIT2 BIT3 BIT4 BIT5 BIT6 BIT7 BIT8 BIT9 BIT10 BIT11 BIT12 BIT13 BIT14 BIT15
BIT16 BIT17 BIT18 BIT19 BIT20 BIT21 BIT22 BIT23 BIT24 BIT25 BIT26 BIT27 BIT28 BIT29
BIT30 BIT31

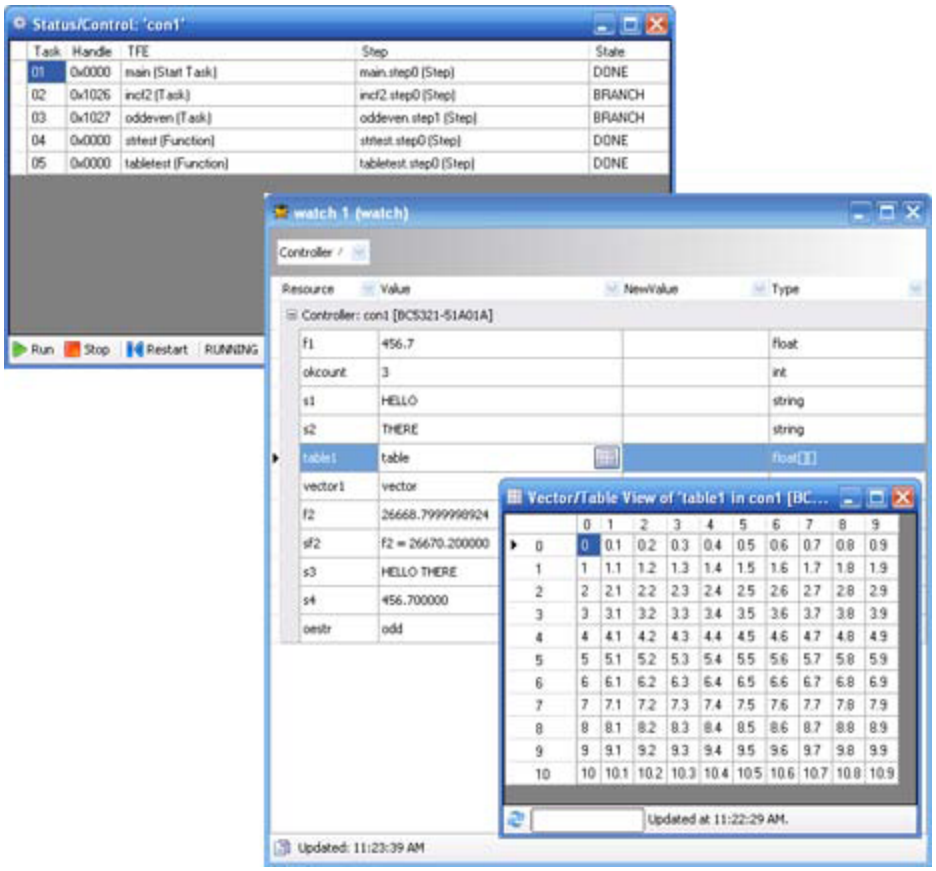
Note: Each space is a separator between the unique words/tokens.

1.8 Debugger Mode

QuickBuilder has extensive debugging capabilities which include variable watch windows, program status display, multiple task breakpoints, real-time variable value display by mouse over, and MSB instruction monitoring.

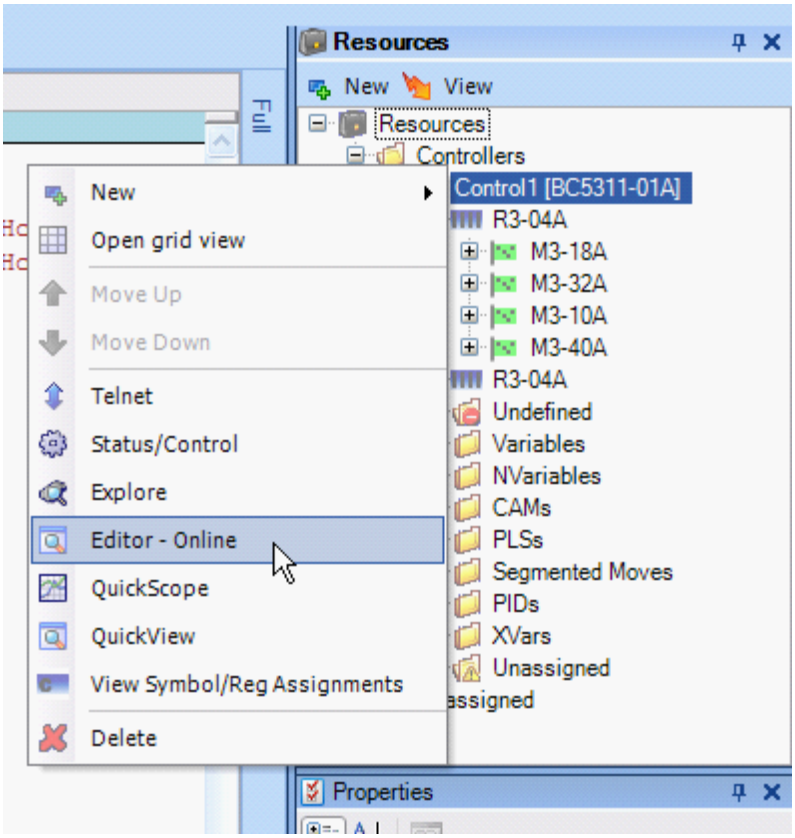
1.8.1 Watch Windows

Watch Windows are created via the Project Manager and allow you to monitor variables or resources with a simple drag and drop from the Resource Window. You can also write to any "writeable" resources, view arrays, and do some simple data-logging. You can create as many Watch Windows as you like and their configuration is saved as part of the project.

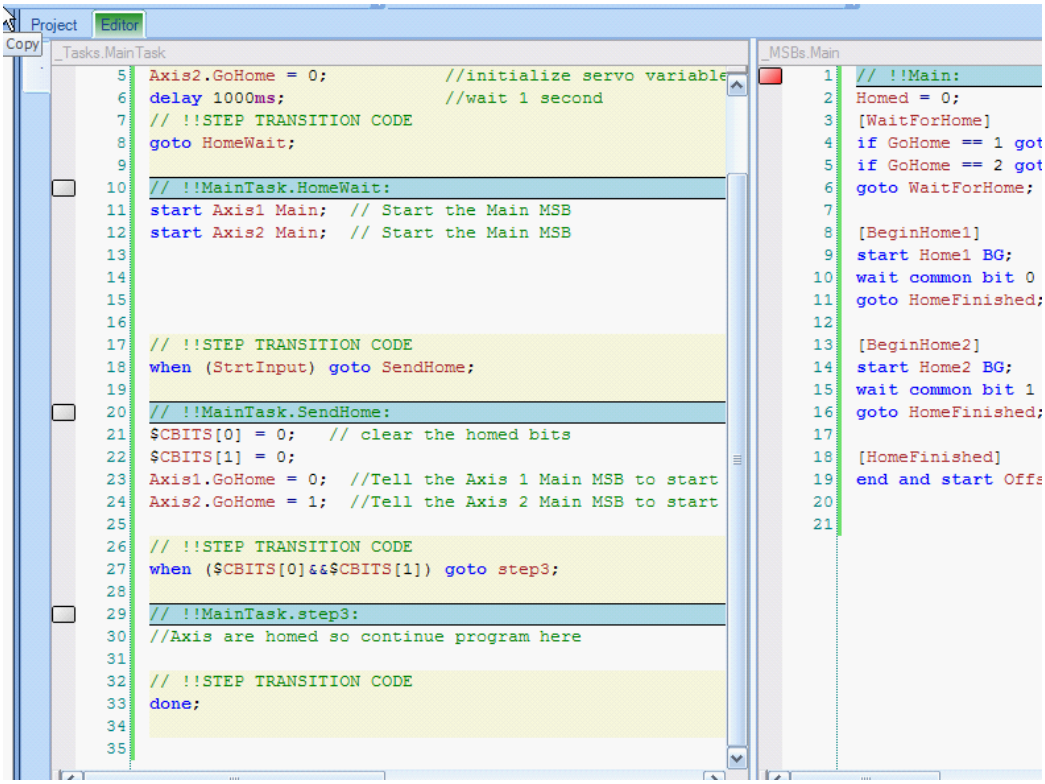


1.8.2 Online Variable Monitoring

QuickBuilder has the ability to view "online" values of any given resource using the Editor-Online feature. This allows you to run your program on a controller and then simply hover your mouse over certain variables within your program editor to determine their current values and type.

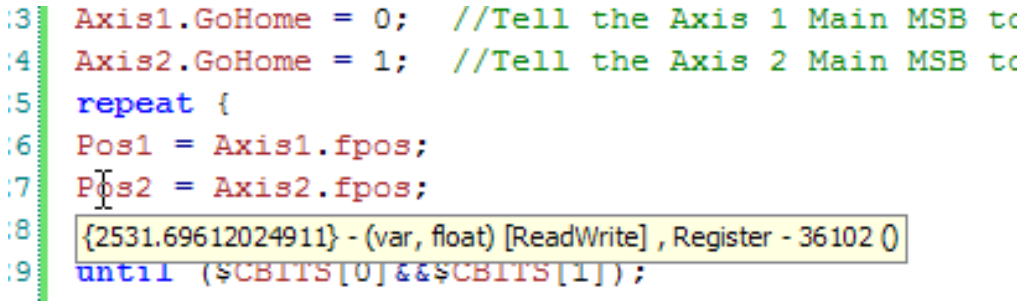


Once you are Online you will see the Editor highlighted in green.



This feature allows you to mouse over any resource and see its value. The value is shown within the curly brackets {}. This also shows you details about the resource, such as type and location.

If you mouse over a resource while the Editor is not online you will still see the resource details, but the value reports back as {?}. Below you can see the value of Pos2, which is a read/write float variable. This value is a snapshot of the resource taken at the time you mouse over it. If you want to update it simply move the mouse away and then back over the resource.

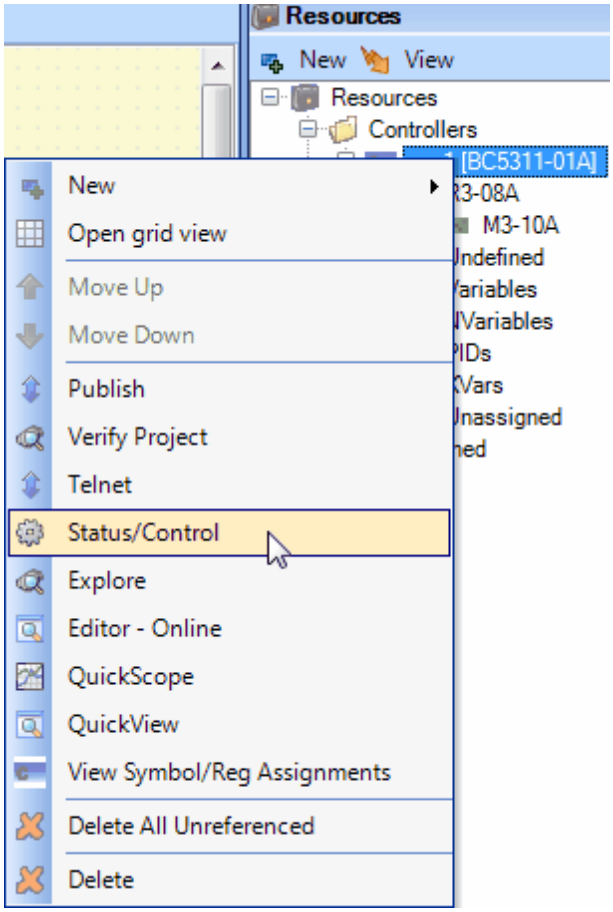


To exit online mode simple return to the resource menu and select Editor - offline. Translating a program will also automatically cause the online mode to be disconnected. Note that it is not recommended that you modify source code while you are online, but you are not prevented from doing so.

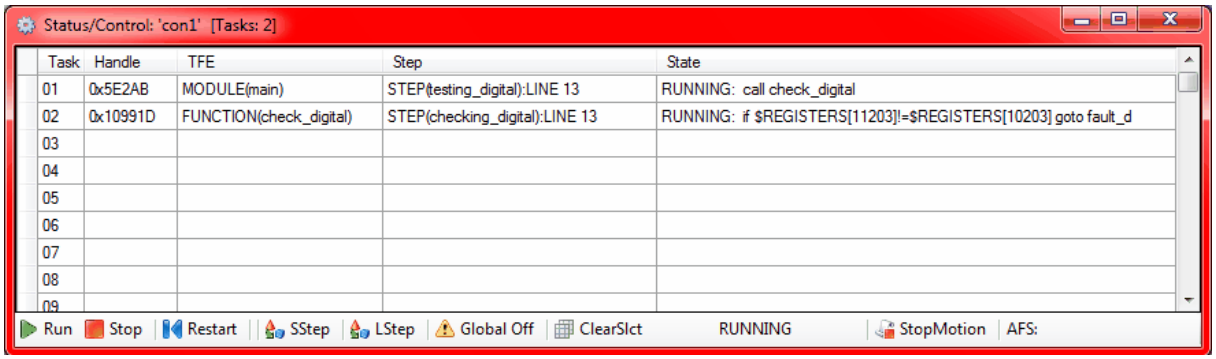
⚠ The 'Editor Online/Offline' option will automatically be activated if the Status/Control window is open while in the 'Editor Mode' or transitioned from 'Project' to 'Editor' mode.

1.8.3 Online Status & Control Monitor/Debugger

The "Status & Control Monitoring" window allows you to monitor the execution of all the tasks in a controller. Additionally, if the program is translated and executed in 'optimize' mode, a source level debugger is available with breakpoints, single stepping and the ability to monitor source code execution. The window is invoked from the controller Resources panel using the Status/Control menu selection:



Selection of "Status/Control" causes the window to open and attempt a connection to the controller. Assuming a controller is online then all the executing tasks and their state of execution will appear:



Note: If the window does not appear as above then the program is translated in legacy mode. In legacy mode SStep, LStep, Global Off, and ClearSlct buttons will be disabled, the line number will not reflect actual SFC execution, and the instruction line will not appear within 'State'.

The headers across the tops of the columns are:

Task - Up to 96 tasks can be run, 1 to 96 is listed. Double clicking the Task cell of an active task will activate that task for direct control by the Execution Buttons at the bottom of the window. Up to 16 tasks can be highlighted. Double click again to remove the highlight or click the 'ClearSlct' button to clear all highlighted.
Handle - Each task, function, or event is assigned a unique handle identifier upon startup of execution. The 'Handle' is used to control individual tasks. It is also referenced in QuickBuilder for task cancellation.
TFE - Task, Function, Event. The type of task and name of it appears here. If MODULE(??????) appears then the task is being started and none has been assigned yet.
Step - Step name and current line execution number. The first number references the SFC line number as viewed within the Project Editor. If the Full Editor is open then an /## will appear after the Project Editor line number, which references the line within the Full Editor. Double clicking this cell, within the Full Editor, will cause that source code to appear in the Editor with the line highlighted and also turn on line tracking during single stepping, for this task. Line Tracking will cause the step that appears in the 'Step' column, and line in the State column, to become current in the editor. During single & line stepping the source code will track with the controller execution. Note that double clicking the column title 'Step' will disable line tracking. Also clicking a different task Step will then cause that task to be in line tracking mode.
State - The current state of the task as well as the instruction being executed. Possible states are:

RUNNING - Executing normally.
STOPPED - Task temporarily stopped.
S_STEP - Stopped in single step mode.
S_RUN - Executing a single step.
L_STEP - Stopped in line step mode.
L_RUN - Executing a single line execution.
L_BREAK - Breakpoint has occurred.
HALTED - May be displayed at task start or while ending, task preparing to start or stop.

⚠ An **'*'** will appear to the right of the 'State' if a breakpoint is active anywhere within the task. This is useful to prevent random unused breakpoints from being left activated or if for some reason you loose sync with the controller.

⚠ The instruction line that appears within the State cell is what is about to be executed in Line Step mode and is what was last executed in Single Step mode. Also in Single Step mode it is what is about to be executed for a conditional test or loop instruction such as an 'if', 'while', 'do', etc. This is due to the fact that Single Step mode maintains the same atomicity as a running program and yields as a normal program does, allowing other tasks to run. Line Step is useful for more detailed debugging but does allow other tasks to run after each line execution, which is not the same as a fully running task. This can result in slightly different results if the variable is shared with other tasks.

The buttons across the bottom are used for task control. Double clicking the 'Task' column will highlight individual rows/tasks. If highlighted then Run, Stop, SStep, and LStep will effect only those tasks. If none are highlighted than all tasks are effected. Buttons are defined as:

Run - Run tasks normally, full speed.
Stop - Stop the task, temporarily.
Restart - Restart the program and begin execution.
SStep - Single step to the point where a program yields control to another task.
LStep - Step each individual instruction and yield task control after each execution (SStep is normal execution where task execution is not yielded until the end of a step or the beginning of a conditional test or end of a loop (while, do...)).
Global Off - This button displays the current mode of the SStep and LStep keys. The possibilities are 'Global Off' and 'Global On'. When on all tasks may be stepped at once. **Be careful as stepping all tasks, or any for that matter, can cause damage to your equipment if not careful.**
ClearSlct - Clear all selected (highlighted) tasks.

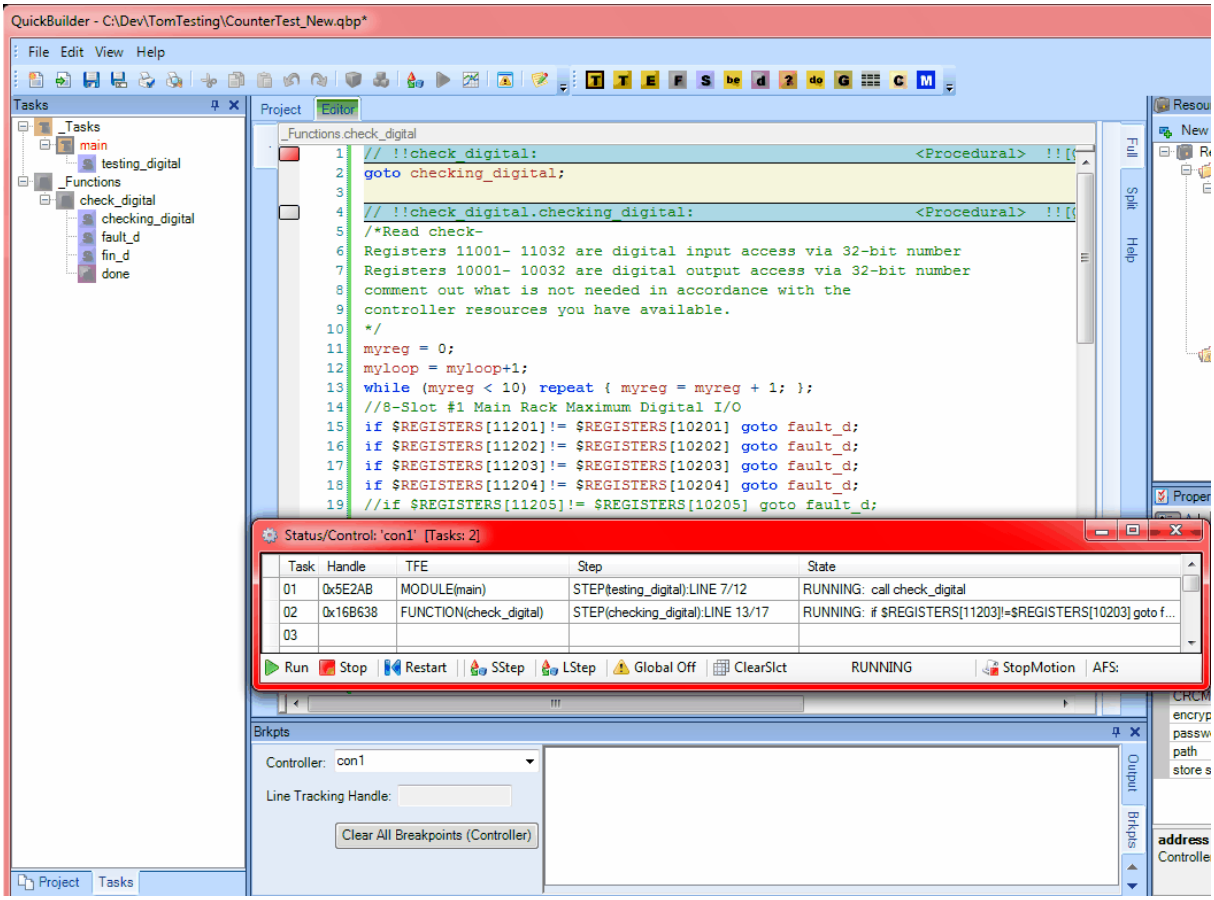
⚠ Reference the Tasks and Steps section for information regarding Legacy and Optimized execution.

1.8.4 Breakpoints

⚠ Breakpoints are only supported in controllers with the optimization level set to 'optimized'. Also it is recommended that programs be translated and newly downloaded to ensure they are in sync. If you set your project CRCMode property to 'common' instead of 'unique' then you will not have to download after translation unless the project is actually different (controller must be initially downloaded with the 'common' project version). The debugger will check the CRC of the last translated project to that in the controller and if they do not match you will be warned. **As is the normal procedure with any debugging, be careful with breakpoints since once you stop executing you will loose control of your equipment and depending upon its implementation damage or injury could result.**

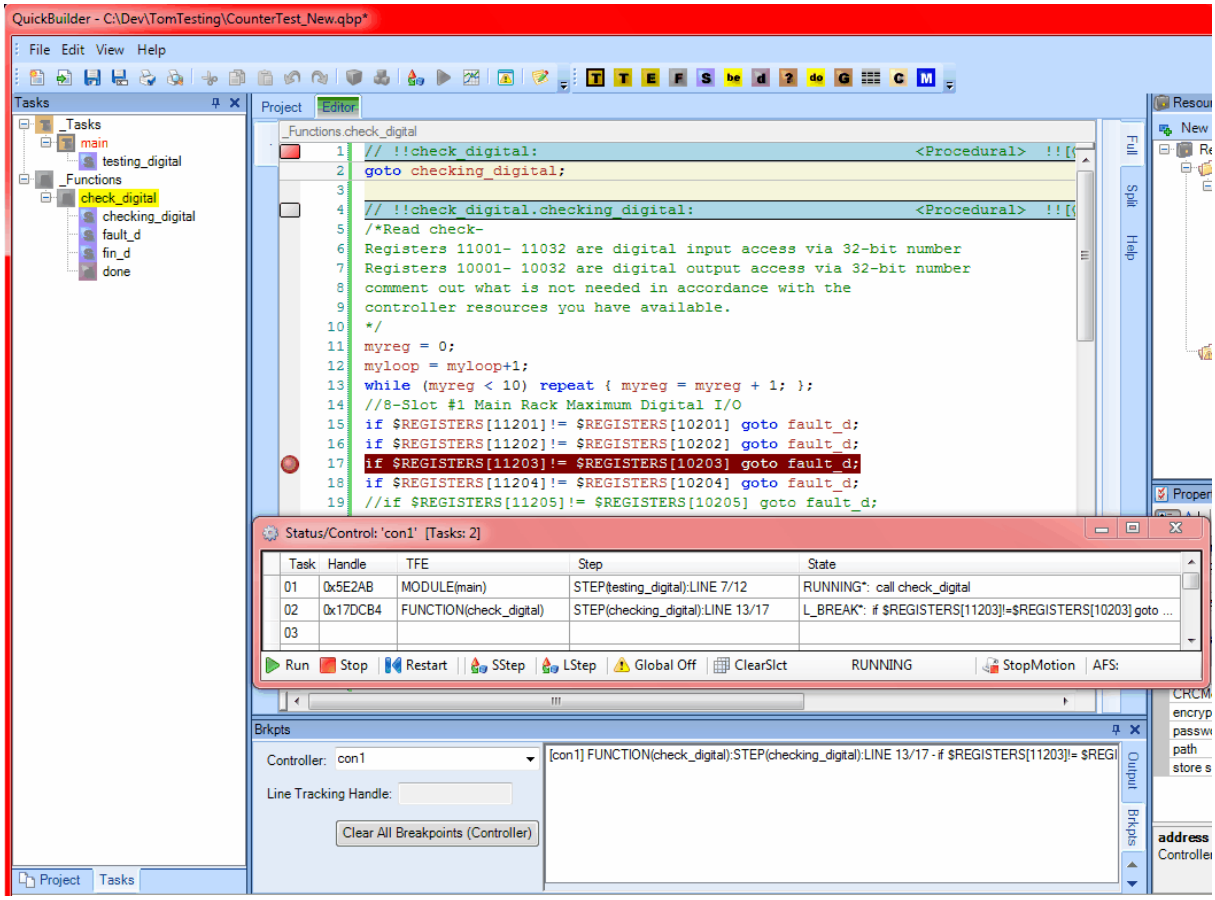
The 'Editor' supports up to 16 source code level breakpoints per task. Breakpoints can be set on either a global program or task basis. To activate the use of breakpoints simply view your program in the 'Editor' mode and open a 'Status/Control' window (from the controller resource menu). The debug environment will automatically activate online variable monitoring as well as breakpoint capabilities.

Below is a sample editor session monitoring the controller execution. Note the 'Editor' tab is green indicating variable monitoring is also active:



There are no '*' next to the RUNNING status, thus indicating breakpoints are not active. We will now set a global breakpoint in all tasks at line 17. Note that task 2 is currently displaying LINE 13/17. The 13 is the line when referenced within the 'Project' view and line 17 is the current 'Editor' view. Since the controller is running full

speed we are only seeing a moment of execution. Upon setting a breakpoint the task will stop when it executes that line within the step. To set a breakpoint double click to the left of line 17, in the gray column, a maroon bullet and highlighted line will appear for each set:



When a breakpoint is set the round maroon indicator will stay in your program as long as you are in the Editor. Also the list at the lower right will show all the breakpoints that are currently set. You can double click on any breakpoint in the list to move to that location within your editor. The combo box that says 'Controller' lists the currently active controller. You can open another Status/Controller window and by selecting different controllers in the combo box you can interact with both.

If you reference task 02 in the Status/Control window you'll see the current status is now L_BREAK. This means the task has stopped running and has hit a breakpoint. The instruction shown in the state window was about to execute. You may either click the RUN button to continue to the next breakpoint or click LStep to single step to the next line.

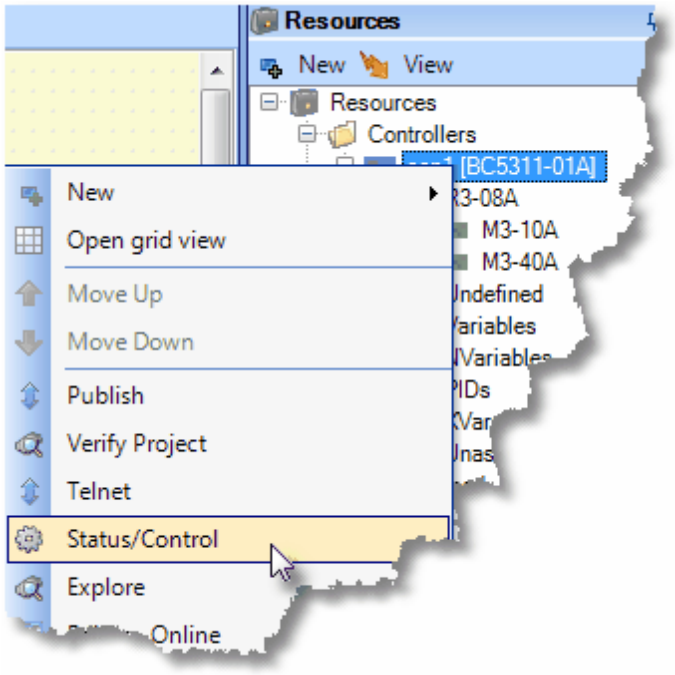
To clear a breakpoint simply double click the round maroon indicator and it will be removed, alternately click the 'Clear All Breakpoints' button if you wish to clear all in the controller. You may also set breakpoints for individual tasks by highlighting those tasks in the Status/Control window and then setting a breakpoint. The breakpoint will only be active on the tasks highlighted. Note that if you clear breakpoints, they are cleared for that program location on all tasks.

⚠ Up to 16 global breakpoints are allowed, or 16 different one's per task. Breakpoints are stored on a task basis thus global breakpoints are all tasks and each counts as one.

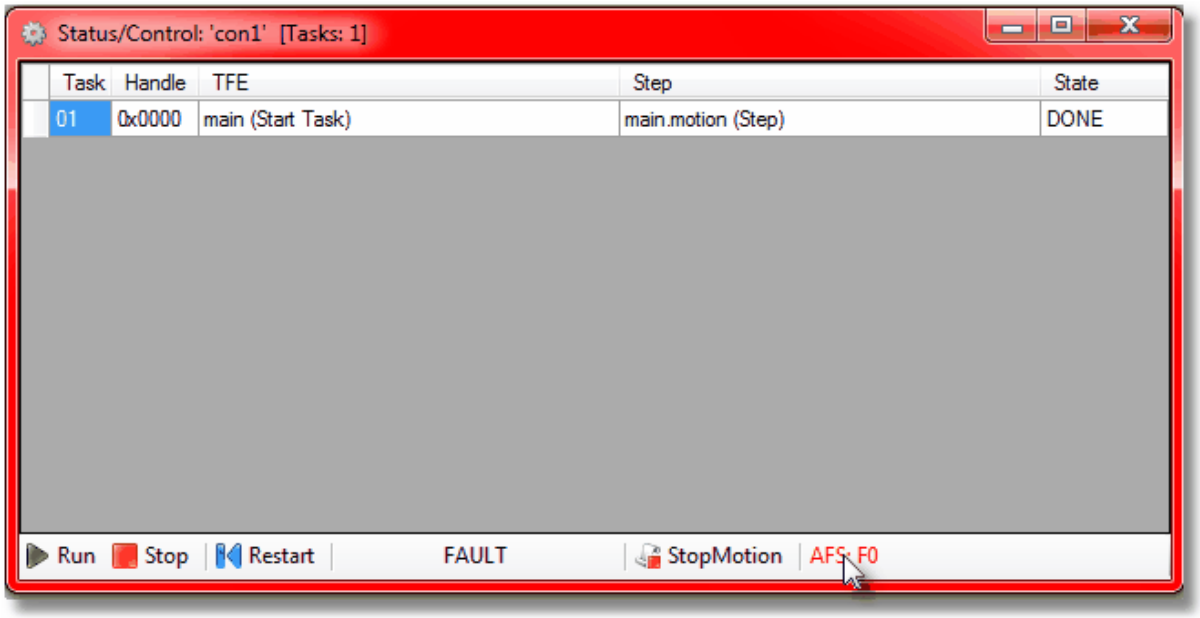
⚠ Double clicking on an item in the 'Step' column will make that line current in the source editor and also enable line tracking for that Handle task. the "Line Tracking Handle" text box will contain the task handle of the active task with tracking enabled.

1.8.5 MSB Status/Control Monitor Fault Processing

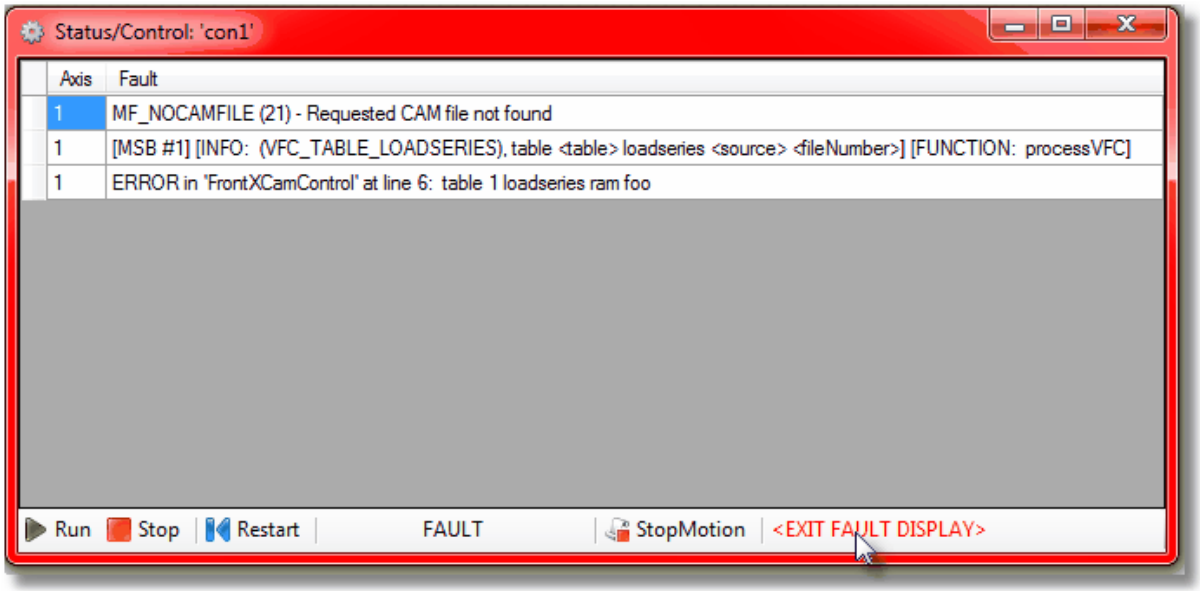
There are a number of features within QuickBuilder to enable the debugging of QuickMotion MSB's. This can be either during normal operation or should a fault occur. A fault is indicate by a flashing FLT LED on the controller CPU. To observe a QuickMotion fault the Status/Control monitor can be viewed:



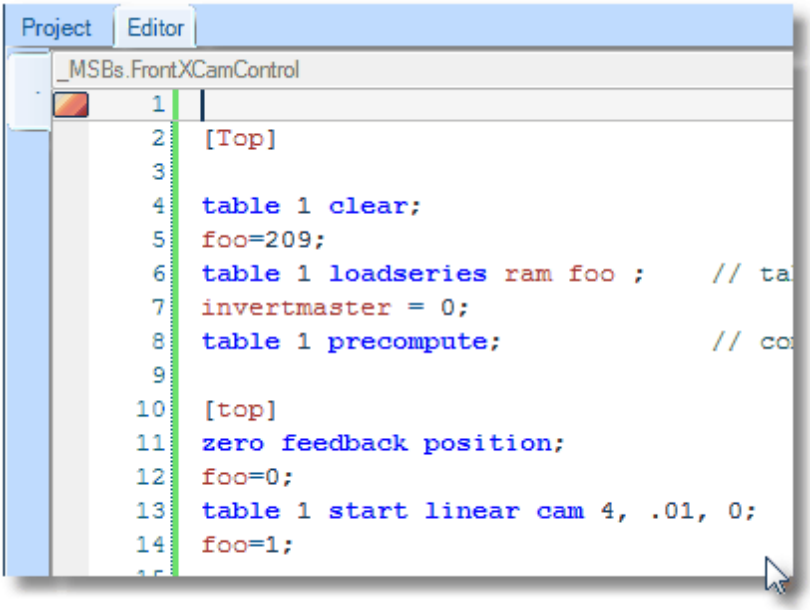
Once the Status/Control window appears observe and click the AFS text. Note that each character represents an axis, with the first on the far left. In the example below a 0 means the axis is OK, F that there is a fault. Below shows a fault on axis 1 since it is 'F'.



Once clicked detailed information about the fault will be shown, if available:



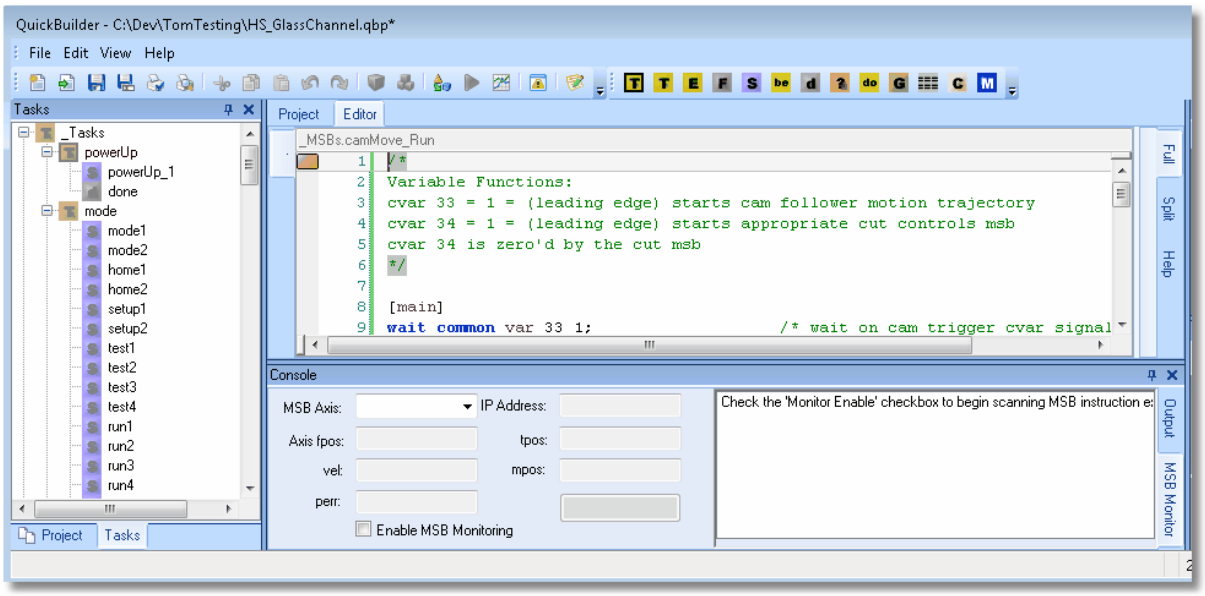
Note that the error occurred at line 6 of the source code of the FrontXCamControl MSB. In referencing that MSB we can see the line listed, 'table 1 loadseries ram foo' as being the problem. In this case there was no camtable209 file present within the controller flash disk.:



1.8.6 MSB Monitor

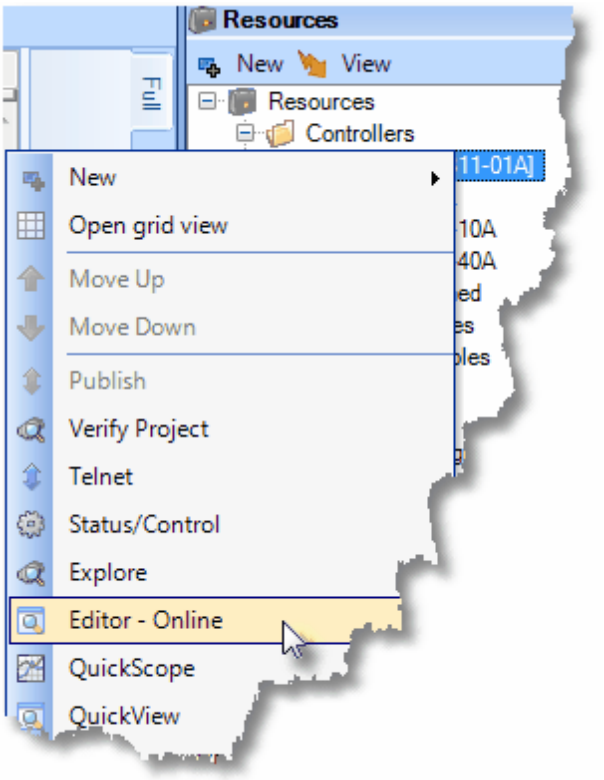
QuickMotion MSB blocks may be monitored the same as regular steps. Since a single MSB can be run by more than one axis, the default axis to request information must be selected while in Editor Mode from the MSB Monitor window:

.



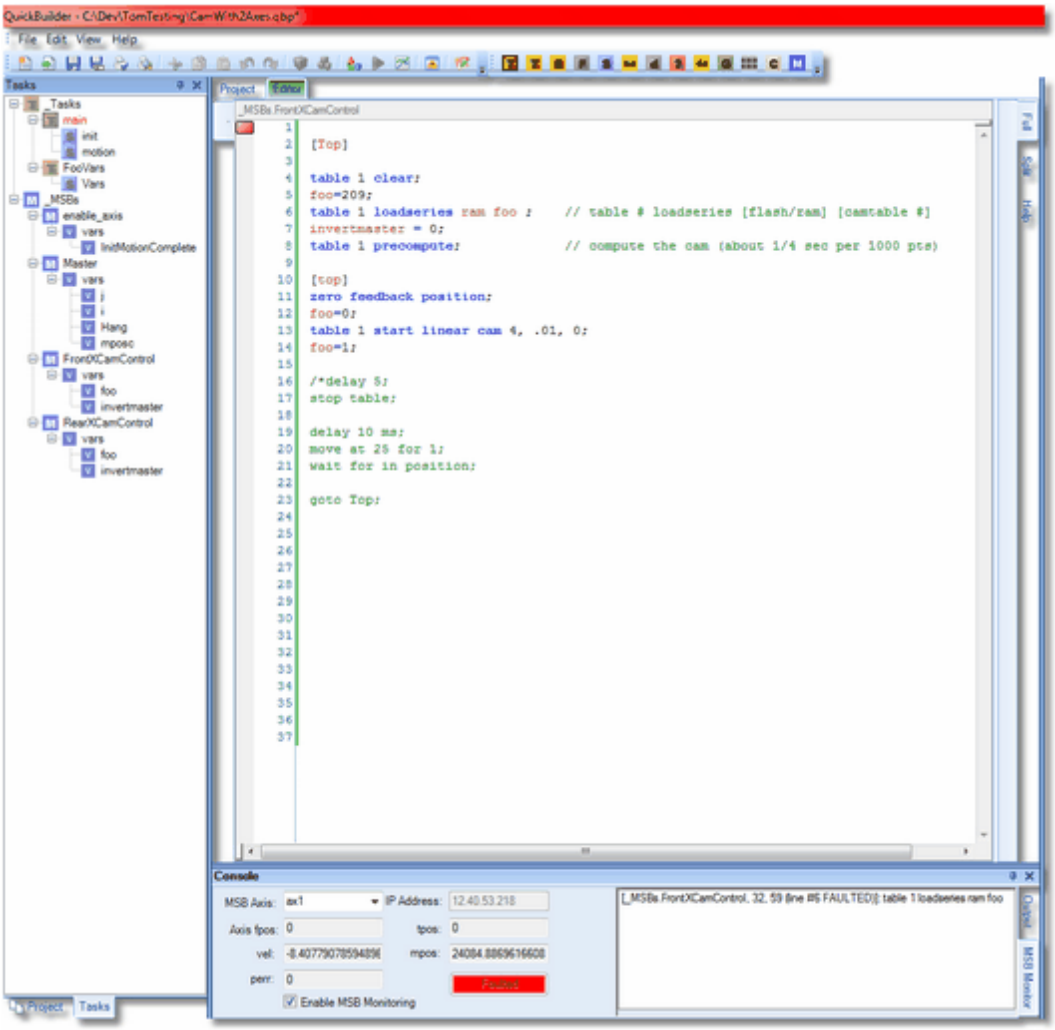
Once online the IP Address will be filled out with that to which you are connected and the MSB Axis pull down list will contain all available axis. That which is seelcted is what will be active for mouse hover data monitoring. Clicking the 'enable MSB Monitoring' check box will cause the axis to be periodically scanned and both position and MSB execution information shown. Double-clicking on the MSB instruction information will bring you to that line of code in the Editor.

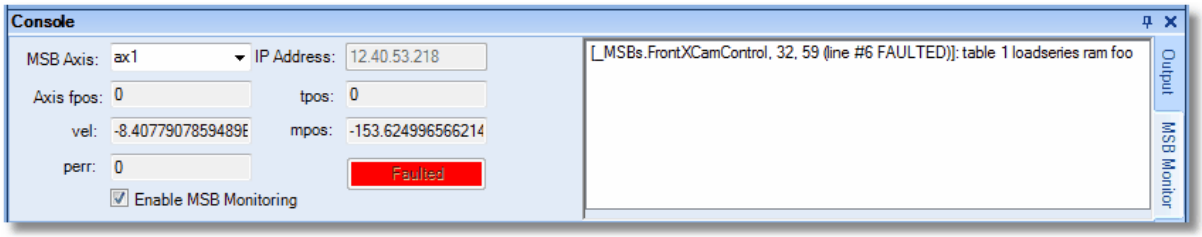
QuickBuilder offers a MSB Monitor when online in the Editor mode.



This monitor periodically (about every second) refreshes axis information for display. Current fpos, mpos, vel, tpos and perr are available as well as the instruction and state of MSB's that are executing. A pull down combo box lists all available axis, that selected is what will be automatically refreshed.

If the axis is faulted, using the example from the 'MSB Status/Control Monitor Fault Processing' section, the following will appear:

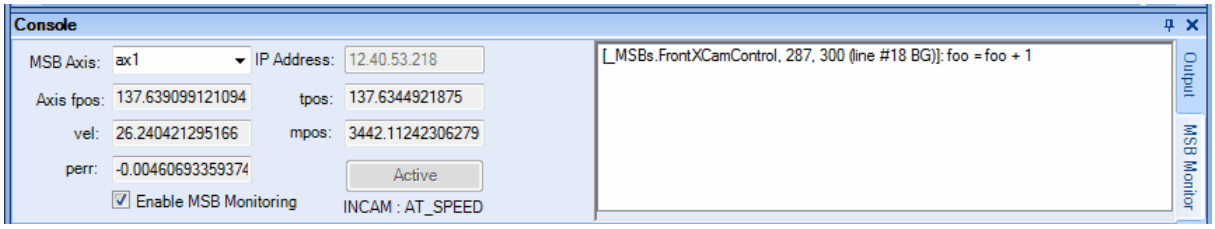




⚠ Note that the 'Enable MSB Monitoring' check box must be checked for monitoring to be active. Also the Editor tab should be green to indicate online debug mode.

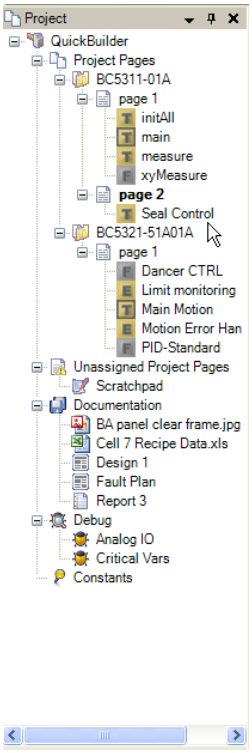
⚠ Double clicking on the MSB line appearing in the list box will automatically make that code and line current in the Editor.

In situations where a fault had not occurred multiple MSB's would appear executing, as well as their line number and axis motion status:


Console
MSB Axis:
ax1
IP Address:
12.40.53.218
Axis fpos:
137.639099121094
tpos:
137.6344921875
vel:
26.240421295166
mpos:
3442.11242306279
perr:
-0.00460693359374
Active
Enable MSB Monitoring
INCAM : AT_SPEED
[_MSBs.FrontXCamControl, 287, 300 (line #18 BG)]: foo = foo + 1
Output
MSB Monitor

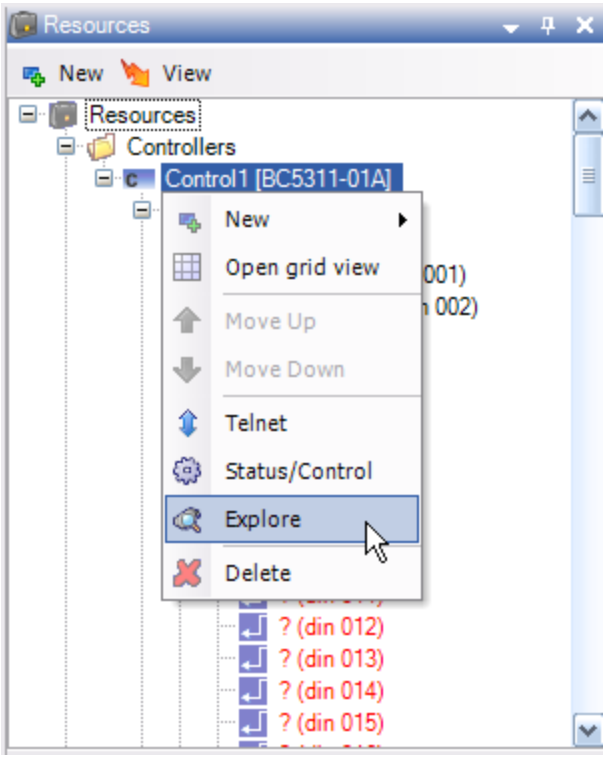
1.9 Project Manager

The Project Manager provides a hierarchical view of the major program elements in a familiar tree structure. Using the tree's drill-down capability, even large multi-controller projects are easy to navigate. At the top of the tree are the controllers used in the project. The program for each controller is built on logical pages. The page concept offers a convenient way to logically break up the program. A controller can have as many pages as desired. Clicking on a page activates it in the graphical SFC window. The pages contain the flowchart view of the application, including all of the tasks, events, and functions used. In addition to Controller pages, there is a scratchpad area that is not associated with any particular controller, but rather can be used in developing modules that might be used in multiple places.

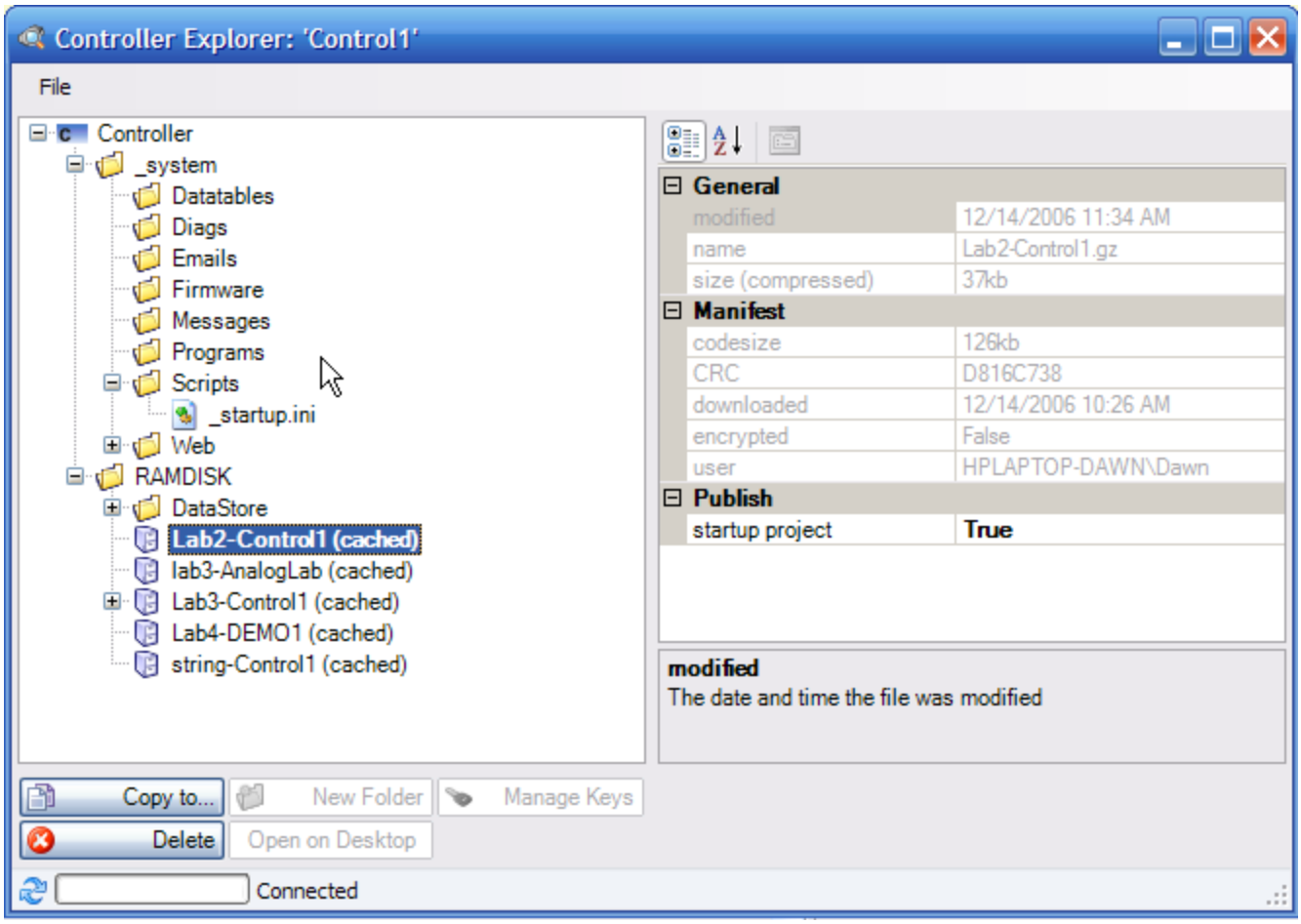


1.10 FTP Explorer

The FTP Explorer allows you to organize the projects that are stored in your controller. It also allows you to set up which project will run when the controller is initially powered up. FTP Explorer can be accessed by right-clicking on the controller as shown.



FTP File Manager:



1.11 Global and Local Resources

Global and *Local* resources are defined and grouped in a tree structure. Global resources are shared by all QS4 *tasks*, *events*, and *functions* (and in the future, all of 1131) – Local resources are created and used on a per-QS4 *task/event/function* basis.

Logical resources are what used to be called registers, except they are no longer numbered and can take on multiple *types* depending on their current assigned value (Boolean, integer, string and double-precision floating-point).

Variables (Volatile & Non-Volatile)

Data Types:

- 32-bit Integer (“int”)
- 32-bit Single Precision Floating Point ("float32")
- 64-bit Double Precision Floating Point (“float64”)
- String (“string”)223-byte maximum length
- Boolean (“boolean”)
- Any (“any”)

Storage Types:

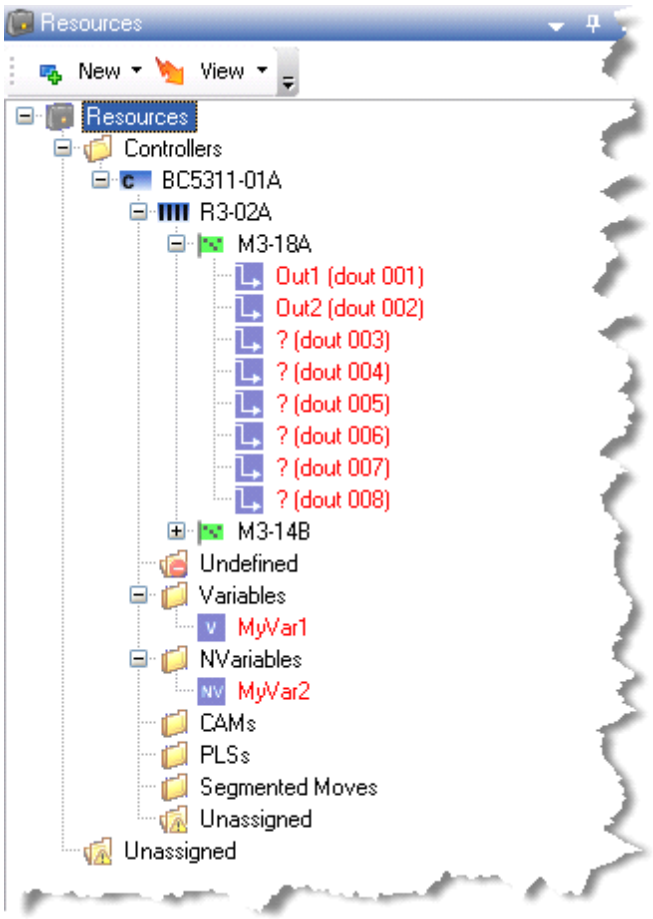
- Scalar (“scalar”) - Holds a single value
- One dimensional array (“vector”)
- Two dimensional array (“table”)

⚠ QS2 Users: Arrays are used in place data tables. They are much more powerful then data tables also since two-dimensional arrays allow the use of multiple variable types.

⚠ Variable names are case sensitive. Spaces and special characters are not allowed. You may find it useful to routinely use lower-case characters when writing your program, because reserved system variables are typically upper-case and it will help differentiate them.

Physical resources represent controller physical entities which usually connect in some way to the outside world, such as inputs, outputs and the like.

For example:

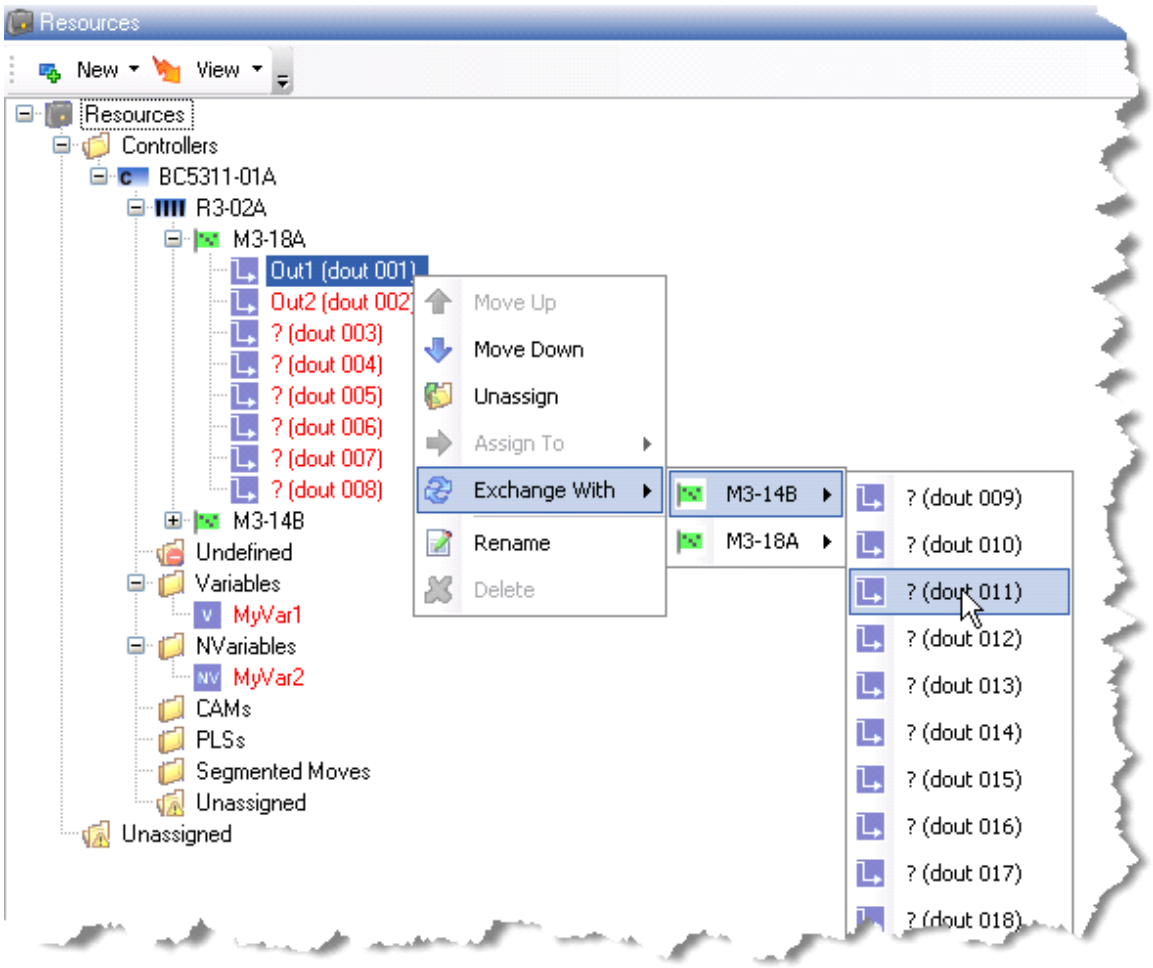


This example shows four resources defined: two logical and two physical.

The first logical resource, *MyVar1* is defined as a simple volatile *Variable*. The second, *MyVar2* is defined as a non-volatile *NVariable*.

The first physical resource, *Out1*, is defined as a digital output and is assigned physically to the first rack's (R3-02A) module (M3-18A) and the first output on the module. The second physical resource, *Out2* , is defined as another digital output and is assigned physically to the first rack's (R3-02A) module (M3-18A) and the second output on the module.

Physical reconfiguration can be performed at any time by context (right-mouse click) menu:



1.12 Intelligent Prompting

While in the editor window certain key sequences can be used to help remember available commands, defined variables, functions, axis, etc.

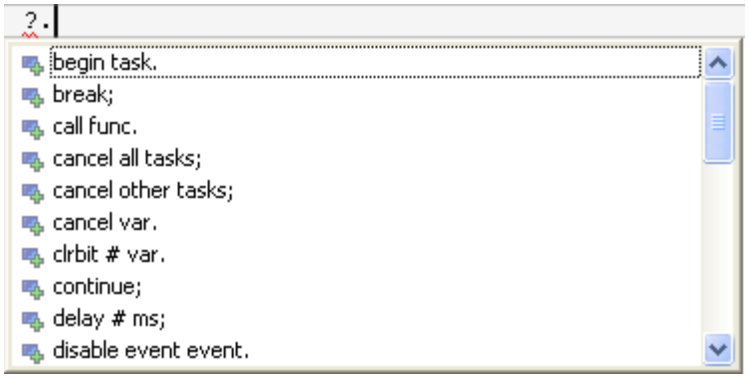
STEP Intelligent Prompting Editor Keys:

Summary:

?. – Display available commands.
ain. – Display available analog inputs.
aout. – Display available analog outputs.
axis. – Display available motion axis.
din. – Display available digital inputs.
dout. – Display available digital outputs.axis. – Display available motion axis.
“selected axis”. – Display motion variables and property for specific axis.
func. – Display available function calls.
evt. – Display available events.
msb. – Display available MSB blocks.

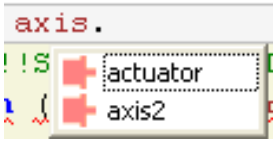
nvar. – Display defined non-volatile variables
pid. – Display available PID definition blocks.
step. – Display available steps.
task. – Display available tasks.
var. – Display defined volatile variables
xvar. - Display defined xvar variables (locals)

?. – Display available commands

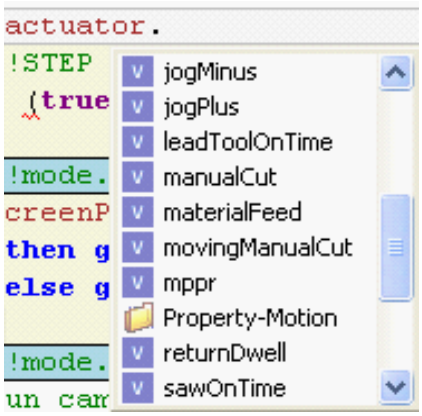


Note: All items enclosed in < > must be replaced by valid references.

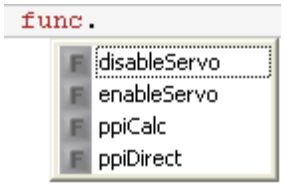
axis. – Display available motion axis



"selected axis". – When axis. is satisfied the resulting axis name may be used to reference all created MSB variables as well as axis properties. For example if the axis name was 'actuator' then actuator. would cause a prompt with MSB variables and axis properties.



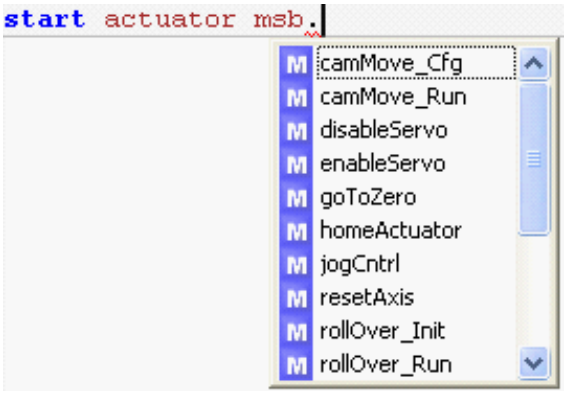
func. – Display available function calls.



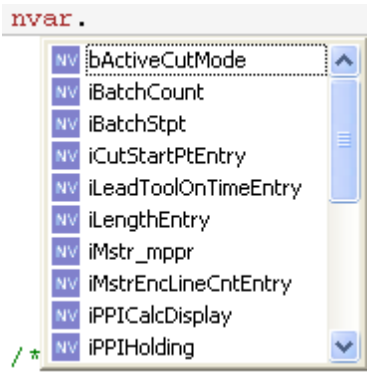
evt. – Display available events.



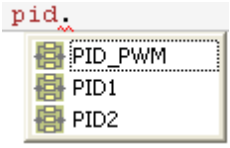
msb. – Display available MSB blocks



nvar. – Display defined non-volatile variables



pid. – Display available PID definition blocks.



1.13 Tech Tips

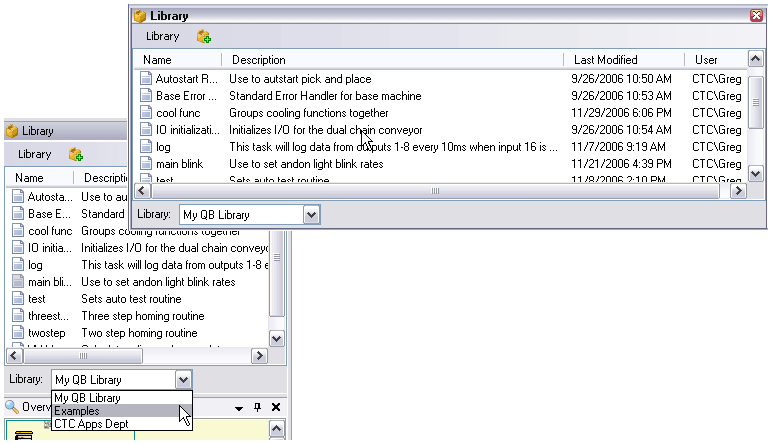
Numerous programming tips are available online at Control Technology's web site, TechTips.

A knowledge base of sample code is also available.

2 Chapter 2: Library Manager

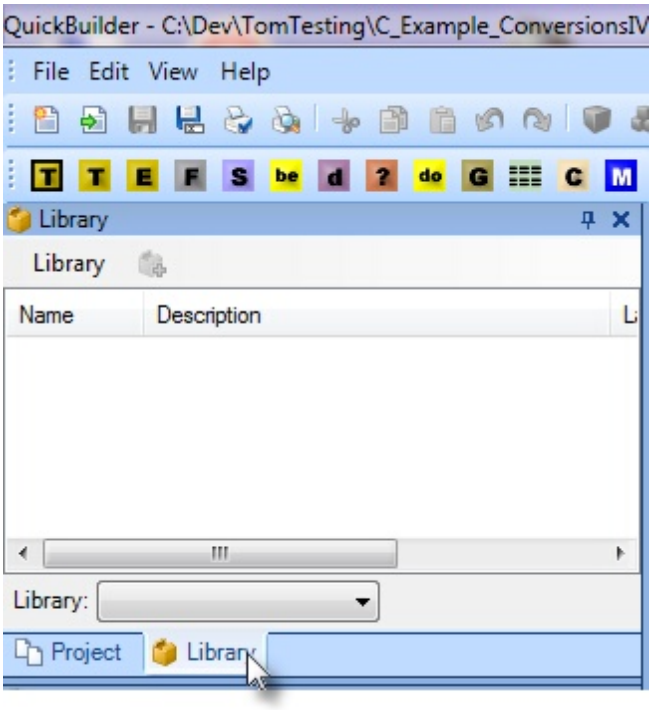
QuickBuilder's powerful Library Manager allows individuals and teams to easily share and re-use common code elements. Any portion of a project from a step, to a series of steps, to a task, to an entire page can be saved as a library element. Library elements are stored in a folder that can be located on the local PC or shared server. Multiple libraries can be open simultaneously.

- Stores and retrieves snippets of logic
- Multiple libraries are supported.
- Encourages code re-use
- Minimizes debug time when proven logic is inserted into a new project
- Standardize Projects and Programs
- Corporate (networked) Libraries as well as User Libraries are supported

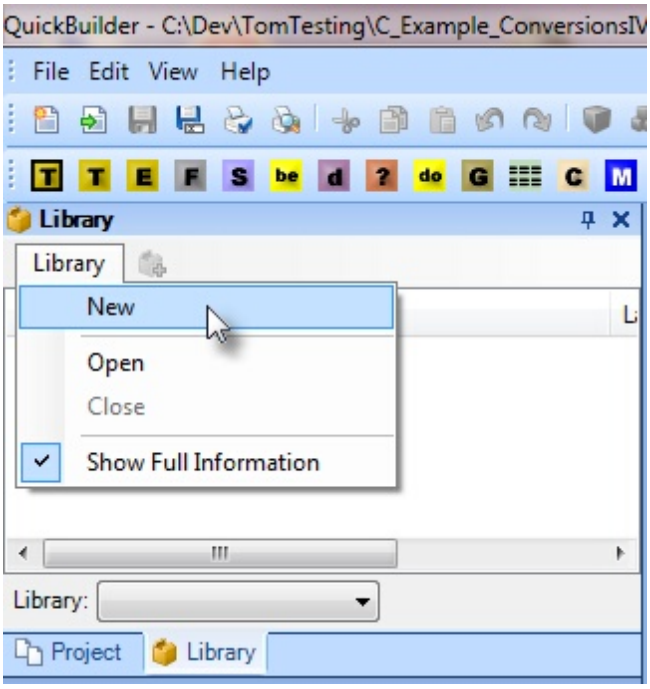


2.1 Creating a Library

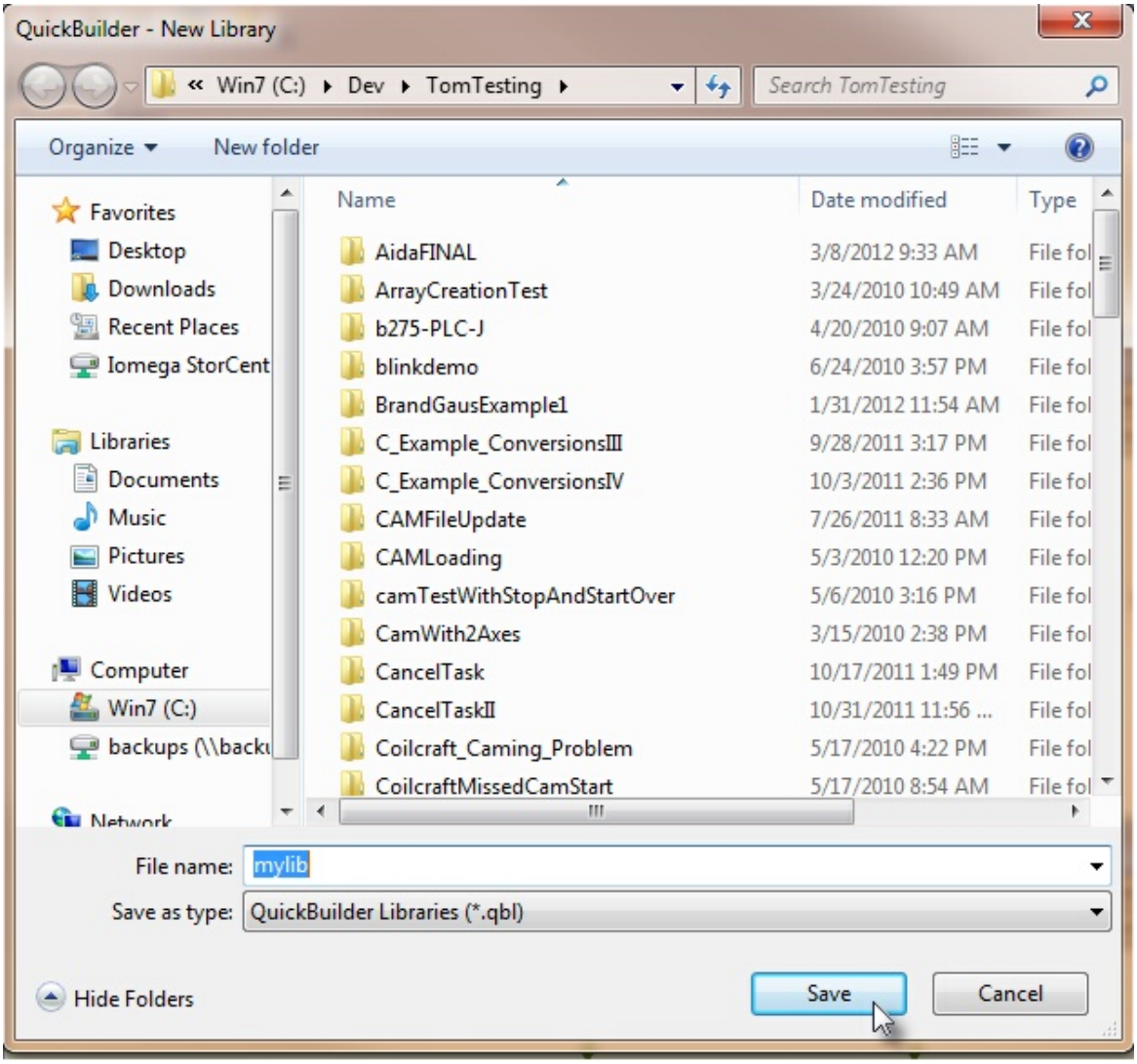
In order to create a library simply select the 'Library' tab:



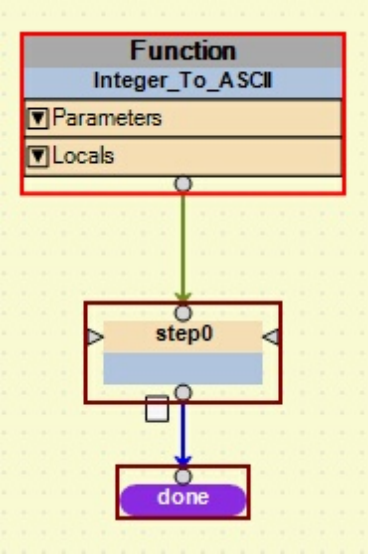
From the menu select the 'New' item:



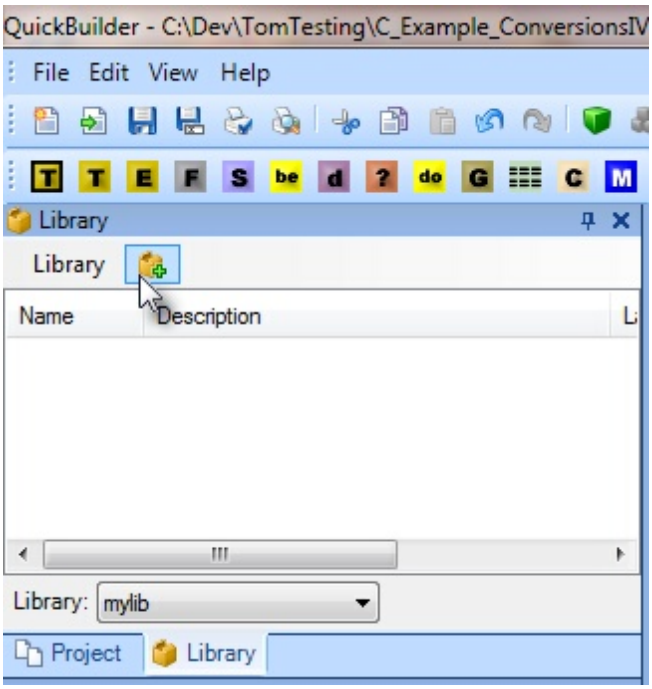
A file dialog box will open, enter the name for the library and modify the path as needed if it is to be stored elsewhere:



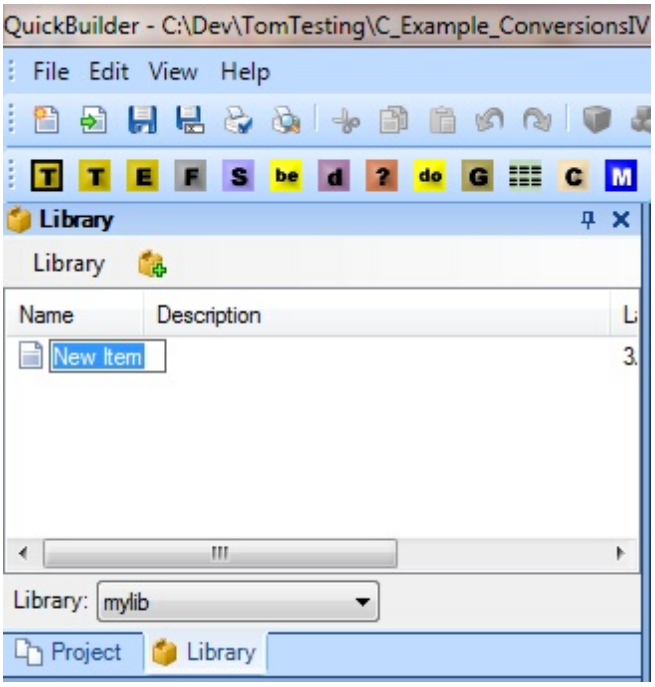
Select the blocks to store in the library by holding the 'CNTL' key down and clicking the steps. Below shows 3 steps being selected.



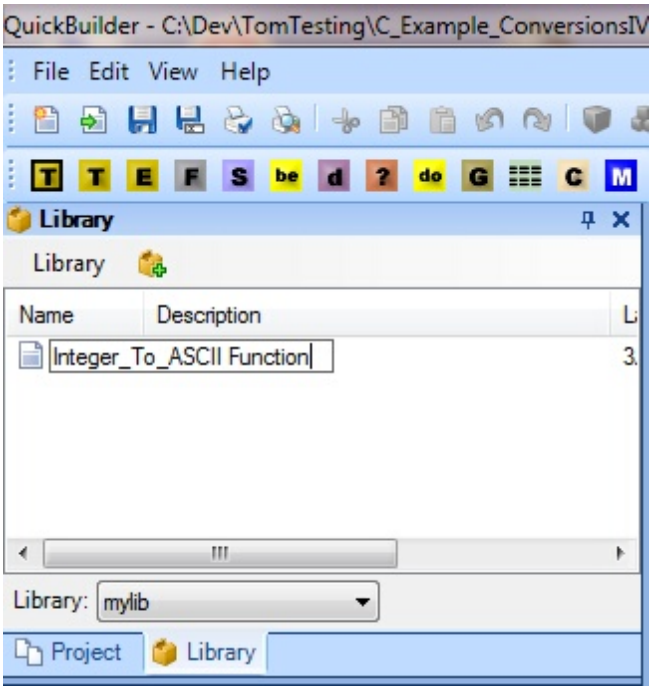
Click the Add icon to store the selected items as a Library entry. Each time Add is selected a new entry will be made.



Edit the 'New Item' name to that you wish to call the Library entry:



Below shows the new item named:



In order to use the item from the library simply select it and drag and drop it to the flowchart area.

3 Chapter 3: QuickStep 4 (QS4)

QuickStep 4 is CTC's *next-generation automation programming language*.

QuickStep 4 is the next-generation programming language for CTC automation controllers. Although different from earlier versions of QuickStep, it is similar enough to allow seasoned automation engineers to easily transition to.

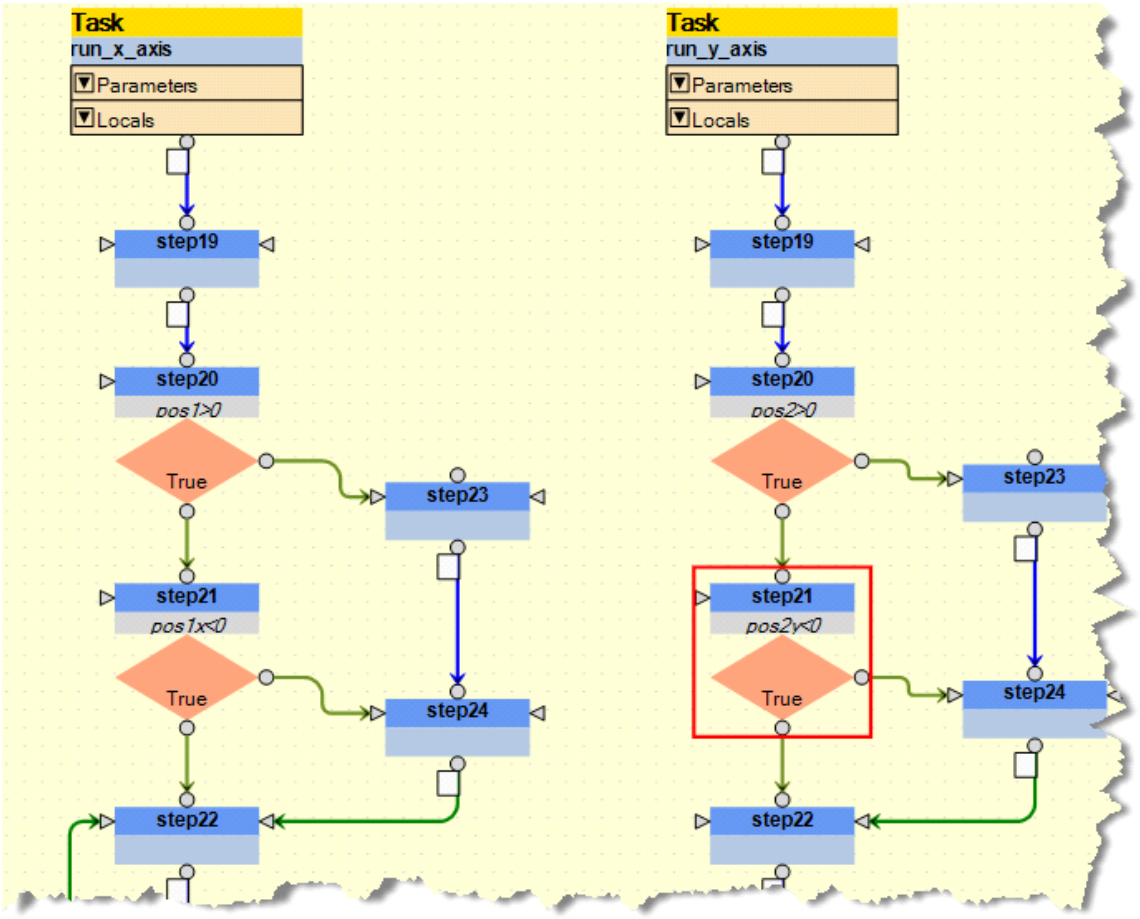
Whereas Quickstep 2 & 3 used a single program that consisted of many steps, QuickStep 4 allows users to break their programs into reusable and flexible tasks, events and functions that encourage good programming practices.

3.1 QS4 SFC Graphical Constructs

QuickBuilder uses a series of graphical constructs to define the overall logic for a QS4-based project within the SFC window. These constructs are as shown below.

3.1.1 SFC Diagram

The SFC Diagram is a graphical representation of overall program flow. The SFC Diagram contains a series of connected graphical constructs which determines and controls how the program is executed on the controller.



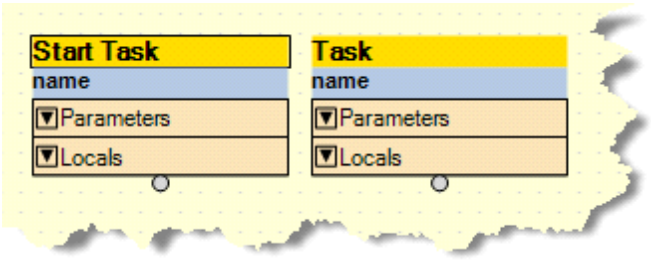
A toolbar in QuickBuilder is used to insert these graphical constructs onto the SFC Diagram. This toolbar is pictured below.



Each of the icons in this toolbar inserts a specific graphical construct into the diagram. In the section that follows, these icons appear in the sub-section header as a guide to their meaning.

3.1.2 QS4 Task Definition

Icon(s) on toolbar: 



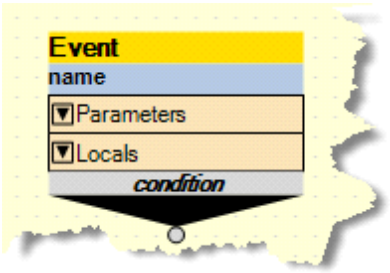
This graphical construct begins a QuickStep4 task definition. The QS4 task name appears within the blue box. When a task is designated as the *start* task, bold lines appear at the top (as shown above on the left). Every QuickBuilder project must have exactly one starting task for each defined controller.

Task constructs are used to begin a series of QS4 steps. Tasks are a section of code that includes multiple steps, decisions, and conditions. Start tasks run only when the program begins. You must have one start task and you can only have one start task. Standard tasks can be started from either the start task or any other task that is currently running. Also, note that you can have multiple tasks running simultaneously. When multiple tasks are running, QuickBuilder uses a time-slice or time-share method to run them. Time-sharing is the sharing of a computing resource (the CPU) among many processes. In the background, the CPU automatically manages which steps to attend to next.

Tasks can have parameters and local variables – although a *start* task cannot take parameters (since no one will ever call a *start* task).

3.1.3 QS4 Event Definition

Icon on toolbar: 



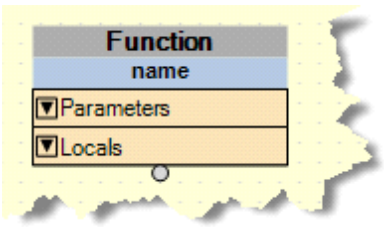
This graphical construct begins a QuickStep4 Event. This construct is a mechanism to define logic for asynchronous events. When the specified condition evaluates to true, the defined steps are executed *in parallel* to other steps that are executing.

Events are implemented like tasks except that they wait on a specific condition before they run. Rather than beginning an event, you enable or arm an event. When you enable an event, it scans and waits for the condition to occur. When the condition occurs, it triggers the event to run. Events are specifically useful for error-handling and looking for error conditions such as E-Stops or light-curtain triggers.

Events are allowed to have local variables, but they do not take parameters.

3.1.4 QS4 Function Definition

Icon on toolbar: F



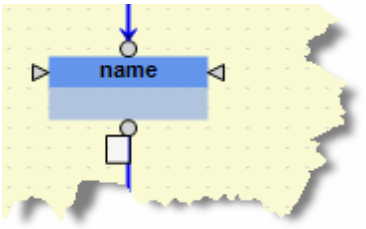
This graphical construct begins the definition of a QuickStep4 (user-defined) function. User-defined functions are a series of steps which perform some operation and possibly return a value. These functions are called by using the QS4 instruction *call*.

Functions are implemented much like a tasks in that they allow both local variables and to pass parameters. Functions are "called" rather than "started" and must be completed before the task that called the function can resume processing. This is commonly known as subroutine programming.

Functions can have parameters and local variables.

3.1.5 QS4 Step

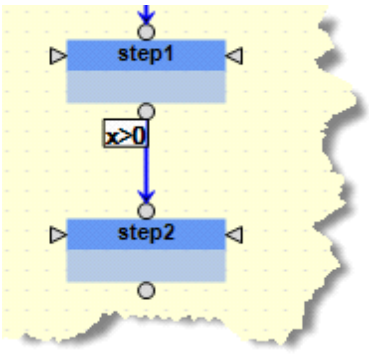
Icon on toolbar: S



The QS4 step contains QuickStep4 code. The name that appears in the top box is the QS4 *step name*.

A QS4 step can be connected to another QS4 step. When a connection *transition* (the white box below a step) is used, it defines when and if the program flow will proceed to the next step. If the connection transition is left blank, program flow continues without waiting for any condition (implicitly *true*).

In the diagram below, two steps are shown with a transition in between. In this case, *step1* will only proceed onto *step2* when $x>0$.

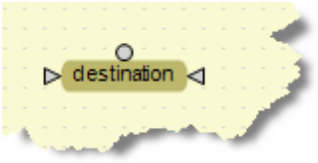


QS4 steps must parent from a QS4 Task, a QS4 Event or a QS4 Function.

QS4 steps are guaranteed atomic during execution with specific caveats on certain instructions.

3.1.6 QS4 Goto

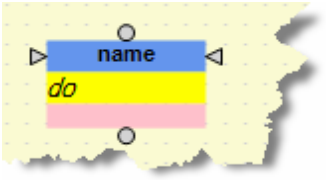
Icon on toolbar: 



The QS4 *Goto* sends program flow to the named destination. It is sometimes more convenient to utilize a graphical "goto" rather than connecting lines in the SFC diagram.

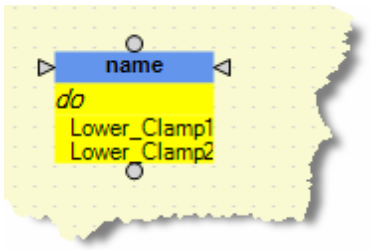
3.1.7 QS4 Do Step

Icon on toolbar: 



The QS4 *Do* Step is a shortcut symbolic construct that represents the textual QS4 do statement. This aids the user in visualizing multi-tasking operations which would have been otherwise obscured.

For example, the construct:

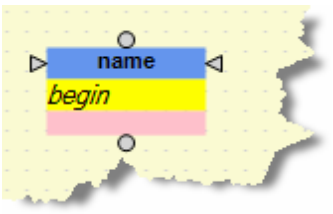


is equivalent to the QS4 statement:

```
do ( Lower_Clamp1 Lower_Clamp2 );
```

3.1.8 QS4 Begin Step

Icon on toolbar: 



The QS4 *Begin* Step is a shortcut symbolic construct that represents the textual QS4 begin statement. As with the QS4 Do Step, it aids the user in visualizing multi-tasking operations which would have been otherwise obscured.

For example, the construct:

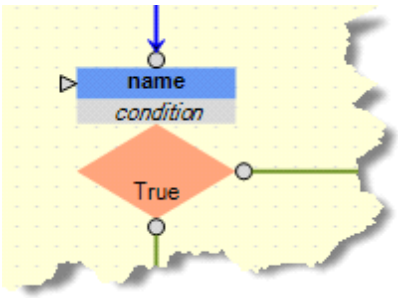


is equivalent to the QS4 statement:

```
begin parallel_task;
```

3.1.9 QS4 Decision Step

Icon on toolbar: 



The QS4 Decision construct can be used in place of the if/then/else statement to provide a more readable SFC diagram.

By default, the *true* path is on the bottom, but the user can select an alternate with the *false* path on the bottom.

3.1.10 QS4 Done Step

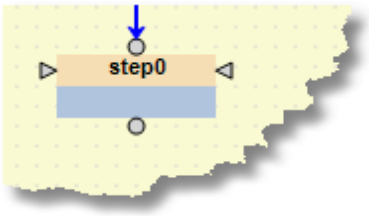
Icon on toolbar: 



The QS4 Done construct visually represents a QS4 done statement (for tasks and events) or a return statement (for functions).

3.1.11 QS4 'C' Step

Icon on toolbar: 



The QS4 'C' step contains ANSI 'C' code provided by the User. The name that appears in the top box is the *step name*.

A QS4 'C' step can be connected to other QS4 steps. Unlike the QS4 step, there is no transition blocks. QS4 'C' steps must parent from a QS4 Task, a QS4 Event or a QS4 Function.

QS4 'C' steps execute just like other steps except that they own the task processing until they return, thus they should not stay in a loop once invoked, returning immediately. Reference Chapter 3 of the 5300 'C' Users Programming Guide for some of the available internal function calls. Chapter 6 of the 5300 Enhancements Overview also references Variant access via 'C'. A sample project is available, called 'QB_C.zip', which provides numerous 'C' function examples.

'C' steps are very powerful constructs that can be used to enhance both the QuickBuilder language and performance. The code within the block is compiled inline with that of QS4 steps, thus no external tools are needed.

⚠ The gcc compiler V3.4 is currently used for the 5300 PLC controller. Microsoft Visual Studio Community Edition 2013 to 2017 is used by PC based Incentive. C++ is not supported.

⚠ 'C' steps can access all QS4 resources, including the file system and communications. When a QS4 program is generated 'C' code is created. Check out the qs4.c file that is available in the project folder after translation. 'C' steps are simply inserted in this file as function calls and compiled with the rest of QS4 in a very efficient manner. You may even place your own external functions and globals in other files and include them.

⚠ 'C' steps can not be easily debugged. It is suggested that you write small amounts of code and test it, using XVAR's as a way to view variables and relay information from you 'C' step during debugging.

XVARNAME.value is what is referenced at the 'C' level, reference the generated qs4.c file and QB_C.zip examples for details.

⚠ 'C' steps are each actually a 'C' function call encapsulated within a step. The function is passed a TASK *task which is typically used to access local variants.

Example:

```
// Convert a 32 bit binary value to an ASCII string
// storing in successive integer registers starting at _CTC_destRegister
// Example:
// _CTC_floatValue = 3.567
// On exit _CTC_destRegister would be 0x33
// _CTC_destRegister+1 would be 0x2e
// _CTC_destRegister+2 would be 0x35
// etc...
// ON RETURN:  _CTC_length is 0 if failed or number of registers used

char buffer[32];  // scratch buffer for convertion
int i, j, val;

      // Convert 32 bit binary value to ascii bytes locally
      sprintf(buffer,"%0.6f",_CTC_floatValue.value);

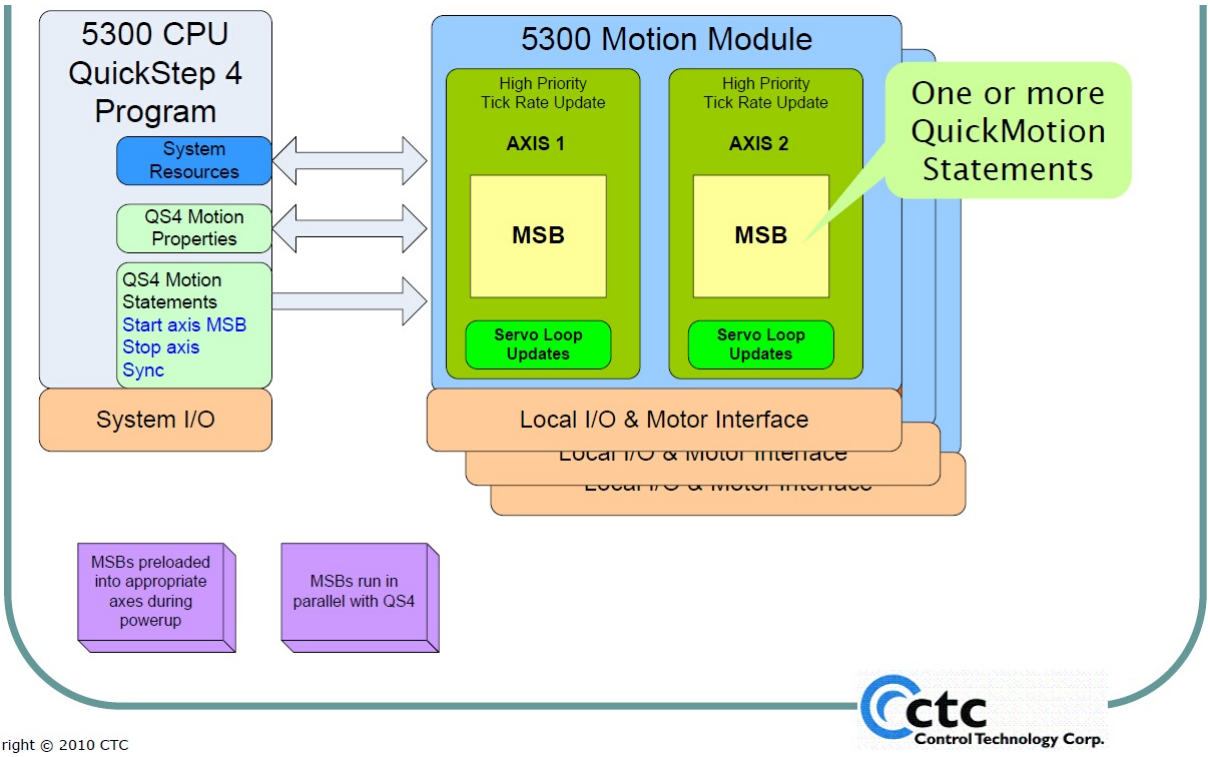
      // Find out length of all characters
      i = strlen(buffer);

      for (j = 0;j!=i;j++,_CTC_destRegister.value++)
      {
            // Get a character
            val = buffer[j];
            // Store it in the next desired sequential register
            regWrite(_CTC_destRegister.value, val);
      }
   // Set number of converted characters
   _CTC_length.value = i;
```

3.1.12 Motion Overview & Sequence Blocks

CTC's Model 5300 and PC based Incentive products use a powerful object-oriented approach to solve motion control applications. This greatly simplifies application creation and maintenance. It also improves performance by off loading the demanding motion control tasks to specialized motion control processors. Motion control commands are created within QuickBuilder and then transferred to one or more physical Motion Modules, for true parallel operation. The logic to control motion is encapsulated in what is called a Motion Sequence Block, or MSB.

Operational Overview:



right © 2010 CTC

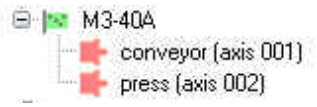
The main components used in Model 5300 and Incentive motion control are:

The Axis Module - The physical Motion Module in the rack (or process running on a Windows PC core when using an embedded PC).

The Axis Object -

- The Axis Object represents a physical servo or stepper axis on a motion module.
- It is created automatically when a motion module is added to a rack in the Resource Manager.
- Axis Objects have many specialized properties that can be configured using the Property Inspector.
- Most of these properties can also be changed dynamically in the QuickBuilder project.
- Axis Objects have various inputs and outputs that control the servo (or stepper) and usually feedback signals that are used to monitor position.

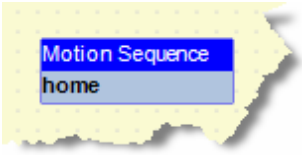
- Using Motion Sequence Blocks (MSBs), you can command each axis to perform a sequence of motion statements.



The MSB - The **M**otion **S**equence **B**lock containing one or more motion statements that execute on the Axis Module's CPU under the supervision of **QuickStep** on the main 5300 CPU (or process running on a Windows PC core when using an embedded PC).

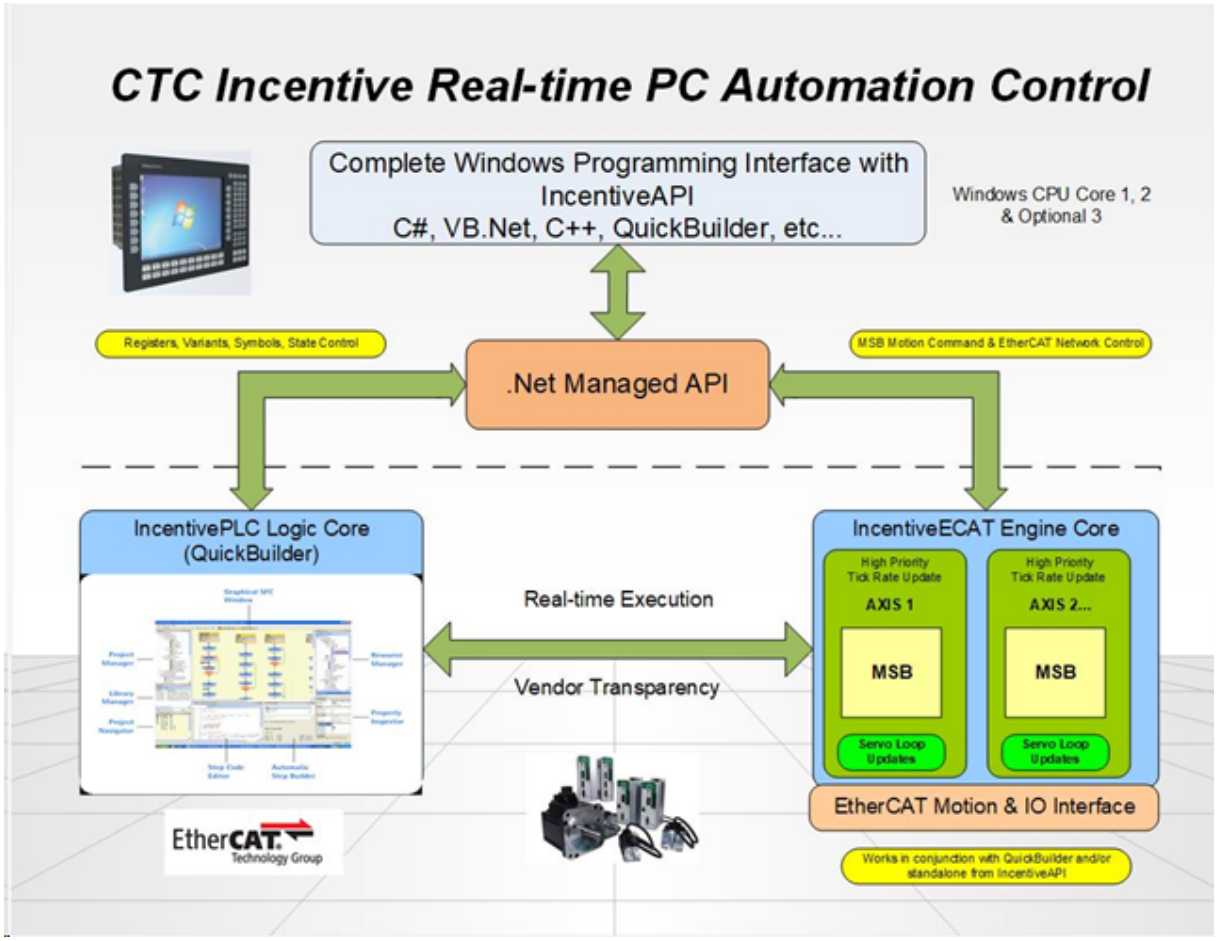
- The Motion Sequence Block (MSB) element holds *QuickMotion* statement sequences.
- MSBs appear in the QuickBuilder project as stand-alone graphical elements.
- MSBs are not associated with any particular axis. This allows the same sequence to be reused many times for different axes, similar to the way a function works.
- An MSB is started on a given axis by using the "Start MSB" statement within QuickBuilder.
- MSBs are programmed in the *QuickMotion* language –a language designed specifically for motion.
- One MSB can start another MSB that can run in parallel on the same axis: Up to 4 foreground (500-800µs) MSBs can be running simultaneously. A foreground MSB runs an instruction per servo tick cycle while a background MSB does not run as periodic, executing using free processor time about every 2 mS.
- Up to 32 total MSBs can be running simultaneously (memory limited).

Icon on toolbar: 



The Motion Sequence Block (MSB) construct holds *QuickMotion* command sequences. These sequences are used with QS4 *Axis* objects.

On the Incentive PC platform MSBs run within the IncentiveECAT process, QuickBuilder tasks run within the IncentivePLC process:



See the *QuickMotion Reference* document for further information on using *QuickMotion*.

3.2 QS4 Resources

3.2.1 Symbolic Names and Resources

Symbolic names are used to reference controller resources within a Quickstep program. Resources are things like memory storage (volatile and non-volatile *variables*), input and outputs, etc.

Basically all instructions either reference variables, QS4 objects or constant values (such as numbers or strings).

Each symbolic name, when referencing a resource, may have properties associated with it that extend its usage and capabilities.

You define symbolic names for the following items:

- Controller memory storage;
- Specialized motion control devices – such as servo and stepping axes;
- Specialized I/O resources such as digital inputs / outputs and analog inputs / outputs;
- *Constants* used in a QS4 program (e.g., multiplier value, maximum speed of a stepping motor or a temperature value).

Each the above resources are identified by a unique *type*, each of which has specific properties particular to it.

In other words a *Variable resource* is nothing more than general memory storage (what was called a "register" in QS2) whereas an *AnalogInput* resource references a hardware-based analog input.

The *AnalogInput* resource is much more complex and contains *properties* that can adjust its operation, while a general memory storage *Variable* is more limited to read/write and the type of data being accessed.

Resource *types* consist of:

- **Variable** – (volatile) *simple* generic memory store for strings, integers (default), and single(float32) or double-precision floating-point (float64);
- **NVariable** – (non-volatile) *simple* generic memory store for strings, integers (default), and single (float32) or double-precision floating-point (float64);
- **AnalogInput** – a reference to an analog input on I/O board;
- **AnalogOutput** – a reference to an analog output on I/O board;
- **DigitalInput** – a reference to an digital input on I/O board;
- **DigitalOutput** – a reference to an digital output on I/O board;
- **PID** - a reference to associated inputs upon which a PID algorithm should be run and what output is to be controlled;
- **XVar** - (volatile) *simple* generic memory store for strings, integers (default), and single or double-precision floating-point. Stored local to generated output code and not public to most communication protocols. Faster operation than a Variable and arrays not supported. May also be designated read-only and initialized with static data when defined as a CONSTANT;

- Motion-related resources such as *Axis*.

One key departure from QS2 is the concept of a register. Users can visualize that *Variable* or *NVariable* resources are simply registers but in reality, QS4 allows these resources to contain an *integer*, a single precision floating point (*float*), a double precision floating point (*double*), or a *string* – or a 1 or 2 dimensional array of such.

3.2.2 Resource Declarations

Resources must be defined prior to translation for use in the controller. The *Resource Manager* handles such in a graphical manner. Users may elect to define resources prior to use in the QS4 Editor, as they are used, or after they are used in the logic.

All resources declarations are public and may be referenced within a *task*, *function* or *event*.

The assignment of specific resources, such as which *DigitalInput* corresponds to what physical connection at the controller, is performed at the project level within QuickBuilder using the *Resource Manager*. This "soft configuration" allows the same program to be reused regardless of the physical I/O assignment. Each controller can use the same program, with a differing I/O assignment, as desired.

3.2.3 Vector and Table Declarations (Arrays)

An indexed *one-dimensional* set of the same type of a *Variable* or *NVariable* (generically a *variable*) is known as a *Vector*. An indexed *two-dimensional* set of the same type of a *variable* is known as a *Table*. Both are considered *Arrays*.

An *Array* is declared in much the same way that a variable is with one exception: the type of storage (either *vector* or *table*) is specified in the Resource Manager. Non-arrays are referred to as *scalars*.

Within a QS4 statement, a reference to a specific element of a *vector* is coded by using square-brackets with the desired *index* within. For example, to reference the *i-th* element of the *vector variable VectorOfData* one would write:

```
VectorOfData[i]
```

The *index* may be a numeric constant (i.e. 5), a variable or a complicated expression such as *(i+1)*2*:

```
VectorOfData[(i+1)*2]
```

To reference the *i-th* row and *j-th* column element (cell) of the *table variable TableOfData,* one would write:

```
TableOfData[i][j]
```

The first element in the array is the 0th element (e.g., i would equal 0 to reference it).

⚠ It is a requirement for NV arrays (tables & vectors) that the array must be pre-allocated. This is done by writing to the "last cell" (last row, last column) you will ever need to access *before* writing to any other cells. Since NV arrays are stored on the file system this allows hashing tables to be built and optimized for the array size you will be using, thus providing my better performance. You can always expand the number of rows in your array by simply writing to the next last expected row cell. Columns can not be expanded once a table is created although the table could be deleted and re-created. To do this entails stopping the application program, using the

telnet 'set close nvariant [variant #]', command and then physically deleting the file from the RAMDISK. Arrays will still work if you expand the row size dynamically, although with much slower access time.

For example, if a 1000 x 6 non-volatile table is required, it would be coded as follows:

```
// First pre-allocate table by selecting and storing a value to

// the last cell to ever be used BEFORE writing to any other cells

NVtable[1000][6]=0;

// then init or fill in the table as your program requires.

NVtable[0][0]=5;

NVtable[10][3]=7; // etc.
```

Deleting a non-volatile table that has been created on the controller using a telnet session:

```
BlueFusion/RAMDISK/_nvar/>dir
drw-rw-rw-  0 owner group 000256 JAN 03 19:38 .
drw-rw-rw-  0 owner group 000000 JAN 03 19:38 ..
-rw-rw-rw-  0 owner group 000528 JAN 03 19:45 _nv36705.var
-rw-rw-rw-  0 owner group 000664 JAN 03 19:11 _nv36706.var
-rw-rw-rw-  0 owner group 002324 JAN 04 05:43 _nv36702.var
Volume:  Capacity - 1012992  Free - 950016  Deleted - 0.

BlueFusion/RAMDISK/_nvar/>set close nvariant 36702
SUCCESS:  Closed non-volatile variant 36702.
BlueFusion/RAMDISK/_nvar/>delete _nv36702.var
SUCCESS: File deleted.

BlueFusion/RAMDISK/_nvar/>dir
drw-rw-rw-  0 owner group 000256 JAN 03 19:38 .
drw-rw-rw-  0 owner group 000000 JAN 03 19:38 ..
-rw-rw-rw-  0 owner group 000528 JAN 03 19:45 _nv36705.var
-rw-rw-rw-  0 owner group 000664 JAN 03 19:11 _nv36706.var
Volume:  Capacity - 1012992  Free - 952576  Deleted - 0.

BlueFusion/RAMDISK/_nvar/>
```

3.2.4 Constants & Literals

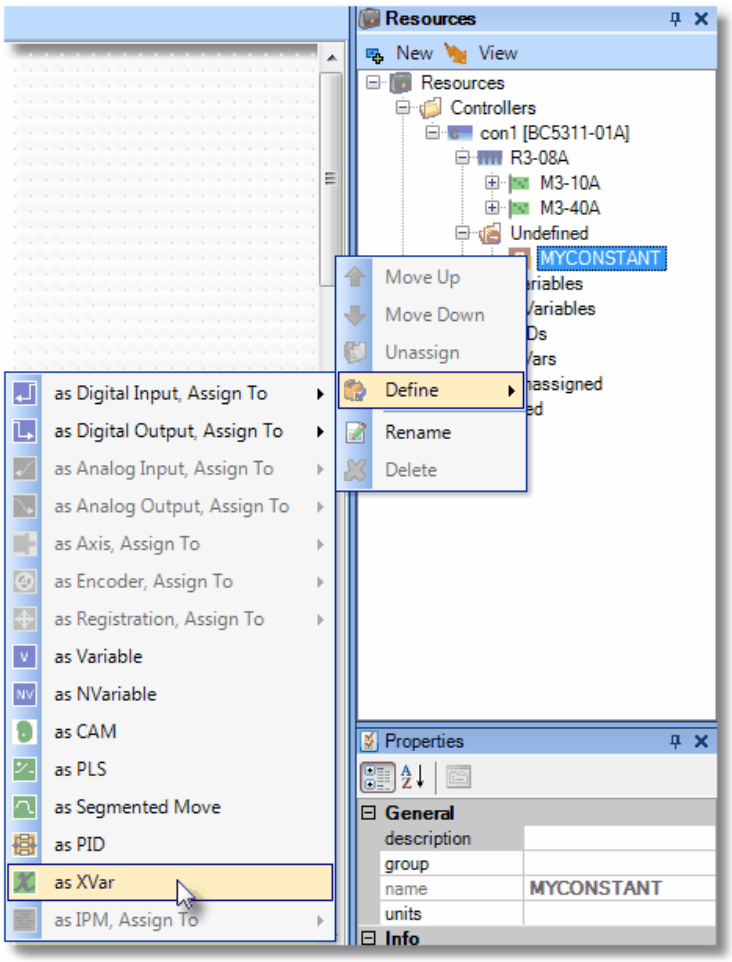
Literals are values that which define themselves and are not reference by a symbol, like constants. The number '5' is a literal. String literals must be used using the 'string' function, such as string("My literal string").

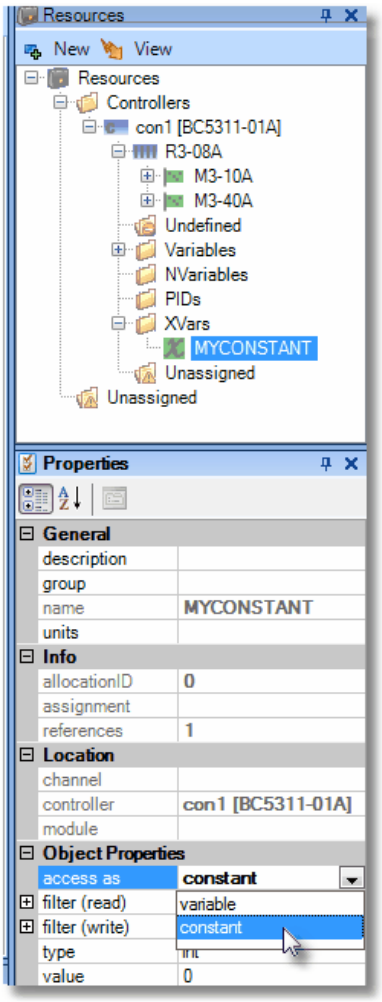
⚠ Note that if a negative number is used in a conditional it should be enclosed in parenthesis, (...). For example x >= (-8). Failure to do so will typically result in a 0 literal.

Constants are used to associate a symbolic name with a constant value. Unlike variables, constants are read-only, and can only be optionally defined using an XVar.

Constants may be defined as integer, double or string constants.

It is suggested, but not required, that constants be in uppercase.


Resources
New
View
Resources
Controllers
con1 [BC5311-01A]
R3-08A
M3-10A
M3-40A
Undefined
MYCONSTANT
Move Up
Move Down
Unassign
Define
Rename
Delete
as Digital Input, Assign To
as Digital Output, Assign To
as Analog Input, Assign To
as Analog Output, Assign To
as Axis, Assign To
as Encoder, Assign To
as Registration, Assign To
as Variable
as NVariable
as CAM
as PLS
as Segmented Move
as PID
as XVar
as IPM, Assign To
Properties
General
description
group
name
MYCONSTANT
units
Info



'value' property field can be modified to set the read-only value.

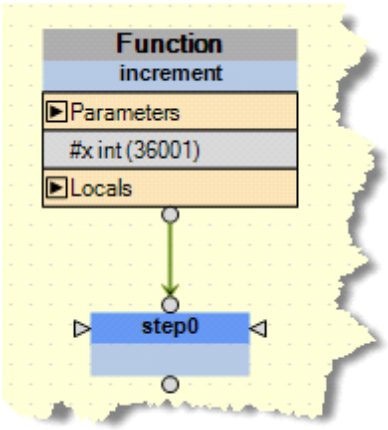
3.2.5 Indirect Variables

Indirect Variables are used to pass a *reference* to a variable when used with a *Task* or *Function*. This allows the *Task* or *Function* to operate on the passed variable – including changing the variable's value.

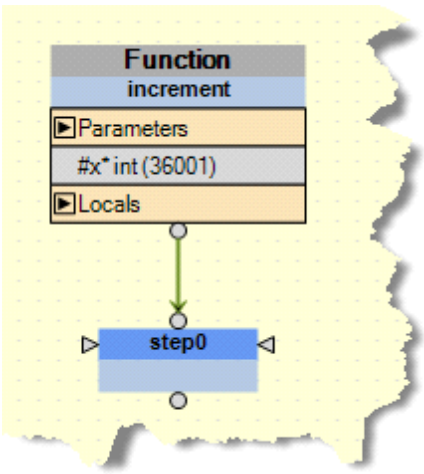
Normally, without *indirection*, parameters are passed by value. *Non-scalar* variables (arrays) can only be passed *indirectly*.

Indirect Variables appear with an asterisk by their name.

In this function definition, the parameter *x* is passed by value:



In this function definition, the parameter *x* is passed by reference (indirect):



Assume that *step0* contains the following code:

```
#x = #x + 1;
```

If we were to call these two different function using:

```
call increment(some_global_variable);
```

there would be two quite different outcomes:

In the first function (that uses pass by value) – the value of *some_global_variable* would be passed to the function *increment* – and the value would get incremented in *step0* – but the global variable *some_global_variable* would not change.

In the second function (that uses pass by reference) – a reference to *some_global_variable* is passed to the function *increment* – therefore all operations on *x* will actually operate on *some_global_variable*. Thus, the code in *step0* will affect *some_global_variable* – in this case, *some_global_variable* will be incremented by 1.

This is a powerful tool as one could create a task or function which operates on a set of digital outputs – and the code which invoked the task or function can pass a different set of outputs to be manipulated.

3.2.6 Tasks and Steps

QS4 allows for multiple *tasks* versus a single large program. There may be as many *tasks* as desired (up to the controller task limit of 96), each containing any number of *steps*. The *tasks* are combined, as desired, during a link process, with outputted code directly executable by the controller CPU (as compared to QS2, which used an interpreter).

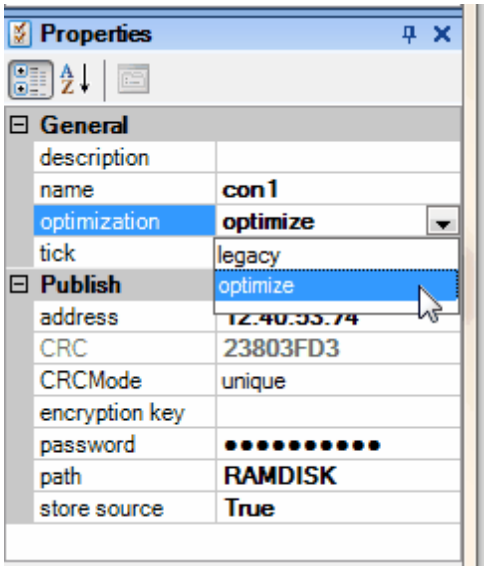
A few rules about *tasks*:

- Each *task/function/event* must have a unique, non-empty name within the project;
- Multiple *tasks* may be included within a single project;
- You may multitask as many instances of a *task* as desired (using ***do*** / ***begin***) up to the controller task limit – presently 96.
- Each *function/event* also counts towards the limit of 96 simultaneously executing tasks.

As will be discussed further in the next section, controller resources are made up of *variables* and *QS4 Objects*. *Variables* and *QS4 Objects* are created and defined using QuickBuilder's Resource Manager (RM).

QuickBuilder has two modes of translation which directly effect task execution, legacy and optimized. Legacy mode was a transition from Quickstep to QuickBuilder. Some of the QuickBuilder application program remained in Quickstep, some was in compiled 'C'. This was needed for product stability given the scope of QuickBuilder. Legacy mode was all that was available prior to November 2010. It is still supported for existing installations but a new, higher performance mode is now available, Optimized. With the availability of Optimized task execution and code has totally changed. Tasks are now independent threads, executing in parallel, held off only for atomicity. Most importantly is Quickstep has been removed and QuickBuilder code is now 100% compiled 'C' code yielding a significant performance increase. In some cases as much as 22 times faster.

The selection of Legacy and Optimize modes is from the property panel of the controller:



The controller is designed to handle Quickstep, QuickBuilder Legacy, as well as QuickBuilder Optmized programs and can detect which has been downloaded for automatic configuration. and execution.

3.3 QS4 Functions and Expressions

3.3.1 Expressions

Expressions perform mathematical and string operations on variables and constants. Expressions consist of unary (one operand) and binary (two operand) operators as well as functions.

Expressions can contain other expressions enclosed in parentheses.

+ operator

The unary "+" operator is a meaningless operator – no operator occurs. The binary "+" operator adds two numeric values together. It also serves as a concatenation operator for strings. For example, 3+4 evaluates to 7 where as "under" + "way" evaluates to the string "underway". It is the only operator that is valid for strings.

- operator

The unary "-" operator changes the sign of the expression. For example –(3+4) evaluates to -7. The binary "-" operator subtracts two numeric values. For example, 10-7 evaluates to 3.

* operator

The binary "*" operator multiplies two numeric values. For example 11*6 evaluates to 66.

/ operator

The binary "/" operator divides two numeric values. For example 21/2 evaluates to 10.5.

% operator

The binary "%" operator computes the modulus (remainder) of two values. For example, 17 % 4 evaluates to 1 (17/4 is 4 with a remainder of 1).

| operator

The binary "|" operator computes a bit-wise logical-or of two numeric values. For example, 8 | 2 evaluates to 10.

& operator

The binary "&" operator computes a bit-wise logical-and of two numeric values. For example, 9 & 5 evaluates to 1.

^ operator

The binary "^" operator computes a bit-wise exclusive-or of two numeric values.

>> operator

The binary ">>" operator bit shifts *left* a numeric value a specific number of times. A left bit shift is effectively a divide by 2 for each time the bit is shifted. For example 20 >> 2 evaluates to 5.

<< operator

The binary "<<" operator bit shifts *right* a numeric value a specific number of times. A right bit shift is effectively a multiply by 2 for each time the bit is shifted. For example 20 << 2 evaluates to 80.

> operator

The binary ">" operator returns a non-zero value when the left-side expression is greater than the right-side expression.

>= operator

The binary ">=" operator returns a non-zero value when the left-side expression is greater than or equal to the right-side expression.

< operator

The binary "<" operator returns a non-zero value when the left-side expression is less than the right-side expression.

<= operator

The binary "<=" operator returns a non-zero value when the left-side expression is less than or equal to the right-side expression.

== operator

The binary "==" operator returns a non-zero value when the left-side expression is equal to the right-side expression.

!= operator

The binary "!=" operator returns a non-zero value when the left-side expression is not equal to the right-side expression.

&& operator

The binary "&&" operator returns a non-zero value when the left-side expression and right-side expressions are *both* non-zero.

|| operator

The binary "||" operator returns a non-zero value when *either* the left-side expression or right-side expressions are non-zero.

! operator

The unary "!" operator inverts the truth of a Boolean expression – when applied to a non-zero expression, the result is 0.

~ operator

The unary "~" operator inverts all the bits (logical not – one's complement) of specified numeric value.

pi constant

The constant "pi" or "PI" evaluates to the trigonometric constant 3.14159….

Boolean constants

The pre-defined constants "true", "TRUE", "on" and "ON" represent a true Boolean condition and are integer valued to 1. The pre-defined constants "false", "FALSE", "off" and "OFF" represent a false Boolean condition and are integer valued to 0.

⚠ Note that if a negative number is used in an *expression* it must be surrounded by parenthesis, (...). For example x >= (-8). Failure to do so will typically result in a 0 literal.

3.3.2 Numerical Functions

To call a numerical function, use the function name, with appropriately typed parameters (comma-separated) inside parentheses.

Name (Parameters)	Return type	Description
max(a, b)	type independent	Returns the maximum of two values.
min(a, b)	type independent	Returns the minimum of two values.
sin(double x)	double	Returns the sine of x where x is specified in radians.
cos(double x)	double	Returns the cosine of x where x is specified in radians.
tan(double x)	double	Returns the tangent of x where x is specified in radians.
asin(double x)	double	This function computes the arc sine of x – that is, the value whose sine is x. The returned value is scaled in radians and lies between $\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$. The arc sine function is defined mathematically only over the domain -1 to 1.
acos(double x)	double	This function computes the arc cosine of x – that is, the value whose cosine is x. The returned value is scaled in radians and lies between $(0,\pi)$. The arc cosine function is defined mathematically only over the domain -1 to 1.
atan(double x)	double	This function computes the arc tangent of x – that is, the value whose tangent is x. The returned value is scaled in radians and lies between $\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$.
atan2(double y, double x)	double	This function computes the arc tangent of y/x – that is, the value whose tangent is y/x. The signs of both arguments are used to determine which quadrant the result lie within. The returned value is scaled in radians and lies between $(-\pi,\pi)$.

exp(double x)	double	Returns the natural number raised to the power of x. In other words, e^x.
log(double x)	double	Returns the natural logarithm (base *e*) of x.
log10(double x)	double	Returns the logarithm (base 10) of x.
pow(double x, double y)	double	Returns x^y.
sqrt(double x)	double	Returns $\sqrt{x}$.
hypot(double x, double y)	double	Returns $\sqrt{x^2 + y^2}$.
sinh(double x)	double	Returns the hyperbolic sine that is defined by $\frac{e^x - e^{-x}}{2}$.
cosh(double x)	double	Returns the hyperbolic cosine that is defined by $\frac{e^x + e^{-x}}{2}$.
tanh(double x)	double	Returns the hyperbolic tangent that is defined by $\frac{e^{2x} - 1}{e^{2x} + 1}$.
abs(x)	type independent	Returns the absolute value of x.
ceil(double x)	double	Returns the x rounded upwards to the nearest integer.

floor(double x)	double	Returns the x rounded downwards to the nearest integer.
frac(double x)	double	Returns the fractional part of x.
sign(x)	int	Returns 1 if x>0, 0 if x equals 0 and -1 if x < 0.
sign2(x, y)	type independent	Returns the sign of x applied to y. If x < 0, then –y is returned. If x > 0 then y is returned. If x = 0, then 0 is returned.
fmod(double x, double y)	double	Returns the floating point remainder of x divided by y.

3.3.3 String Functions

To call a string function, use the function name, with appropriately typed parameters (comma-separated) inside parentheses.

Name (Parameters)	Return type	Description
trim(string x, string y)	string	Returns the string *x* with all leading and trailing characters specified by the second string *y* removed from it.
left(string x, int s)	string	Returns the leftmost *s* characters from the string *x*.
right(string x, int s)	string	Returns the rightmost *s* characters from the string *x*.
mid(string x, int start, int length)	string	Returns *length* characters starting at position *start* from the string x.
padl(string x, int length, string p)	string	Pads and returns the string *x* with the string *p* such that the length of the string is greater than or equal to *length*. Padding occurs on the left. The length of the padding string must be 1.
padr(string x, int length, string p)	string	Pads and returns the string *x* with the string *p* such that the length of the string is greater than or equal to *length*. Padding occurs on the right. The length of the padding string must be 1.
len(string x)	int	Returns the length of the string *x*. An empty string has a length of 0.
compare(string x, string y)	int	Compares two string – returns -1 if *x* is lexically less than *y*, 0 if the strings are the same, and 1 if *x* is lexically greater than *y*.
find(string x, string y)	int	Searches for string *y* in string *x* – if found, returns the position in *x* (starting from 0) that the string was found. If not found, returns -1.

3.3.4 Bit Functions

To call a bit function, use the function name, with appropriately typed parameters (comma-separated) inside parentheses.

Name (Parameters)	Return type	Description
bit(int x, int n)	int	Returns 1 if bit #*n* is ON in *x*, or 0 if the bit is OFF. *n* can range from 0 to 31.
_rol(<variable>,times)	int	Rotate bits left and returns result, variable not modified, 0 shifted into LSB location.
_ror(<variable>,times)	int	Rotate bits right and returns result, variable not modified, 0 shifted into MSB location.

3.3.5 Special Functions

To call a special function, use the function name, with appropriately typed parameters (comma-separated) inside parentheses.

Name (Parameters)	Return type	Description
addr(*variable name*)	int	Returns the address of the named variable (used for indirection assignment).
isdone(*variable name*)	int	Returns a 1 if the task handles contained in the specified variable are all done, otherwise a 0. If the passed variable is a *scalar*, then it is assumed that the variable holds a task handle to a single task – and therefore only that task is checked if "done". If the passed variable is a *vector*, then it is assumed that the variable holds a task handle to multiple tasks – and therefore all tasks are checked for "done." Thus, this function returns a value of 1 if and only if all of the tasks are "done."
_servoInfo(<Axis>.Axis, request)	int	returns a 1 or 0 based on true or false where 'request' is one of following: XVars defines: __CTC_SERVO_ERROR with constant value of 8 __CTC_SERVO_POSITION with constant value of 7 __CTC_SERVO_RUNNING with constant value of 5 __CTC_SERVO_STOPPED with constant value of 6

		Note: __CTC_SERVO_POSITION returns a 32 bit signed count value representing the current position. *This function is only usable when the Quickstep 2/3 motion simulation MSB's are being used.

3.4 QS4 System Variables

There are a number of pre-defined system variables useful in QS4 programming. These variables are:

- $TASKTIMER
- $DINPUTS
- $DOUTPUTS
- $REGISTERS
- $TRIGGER
- $CBITS
- $CVARS
- $TASKPRIORITY
- $CURRENT_TASKPRIORITYLEVEL
- $TASKHANDLE

3.4.1 $TASKTIMER

The $TASKTIMER variable is similar to the register-based 13002 millisecond timer register in the QS2 world with one difference: the timer is unique and independent for each Task, Function and Event.

$TASKTIMER increments by 1 every millisecond and is useful for timing things as well as handling timeouts waiting for something to occur.

$TASKTIMER is cleared when a task is started, and is therefore useful as well to determine how long a task has been running.

Examples:

```
// wait for up to 1000ms for input1 to turn on

$TASKTIMER = 0;

while !input1 && TASKTIMER<1000 repeat { }



// measure how long the tablefill function takes

$TASKTIMER = 0;

call tablefill;

tablefill_duration = $TASKTIMER;
```

3.4.2 $DINPUTS[ ]

The $DINPUTS vector system variable provides direct access to the digital inputs. The first input is in the [1] element of the vector.

Example:

```
// count how many of the first 16 inputs are on

count = 0;

for i = 1 to 16 repeat {

      if $DINPUTS[i] then count += 1;

}
```

3.4.3 $DOUTPUTS[ ]

The $DOUTPUTS vector system variable provides direct access to the digital outputs. The first output is in the [1] element of the vector.

Examples:

```
// count how many of the first 16 outputs are on

count = 0;

for i = 1 to 16 repeat {

      if $DOUTPUTS[i] then count += 1;

}

// clear the first 8 outputs

for i = 1 to 8 repeat {

      $DOUTPUTS[i] = 0;

}
```

3.4.4 $REGISTERS[ ]

The $REGISTERS vector system variable provides direct access to the controller registers. This is a better alternative to using an override on a user-defined variable. The return value is always an integer. If a variant is referenced with a float type, it will be rounded.

Examples:

```
// clear the millisecond timer

$REGISTERS[13002] = 0;



// read the real time clock

year = $REGISTERS[13019];

month = $REGISTERS[13018];

day = $REGISTERS[13017];

hour = $REGISTERS[13016];

minute = $REGISTERS[13015];

second = $REGISTERS[13014];
```

3.4.5 $TRIGGER

The $TRIGGER system variable triggers QuickScope for data capture when the scope is set for Triggered mode.

It should only be written to – the value written is ignored and can be any value.

Examples:

```
// trigger QuickScope

$TRIGGER = 1;

// also triggers QuickScope since the value is ignored

$TRIGGER = 0;
```

3.4.6 $CBITS[ ]

The $CBITS vector system variable provides direct access to the global common bits that are used on such modules as the M3-40A. The first common bit is in the [0] element of the vector.

The $CBITS vector is a Boolean read/write system variable with 256 elements ([0] to [255])

Common bits are further documented in the QuickMotion manual.

Examples:

```
// wait for CBITS[4] to be on

while !$CBITS[4] repeat { }



// set CBITS[2] on

$CBITS[2] = 1;
```

3.4.7 $CVARS[ ]

The $CVARS vector system variable provides direct access to the global common “vars” (variables) that are used on such modules as the M3-40A. The first common “var” is in the [0] element of the vector.

The $ CVARS vector is a read/write system variable taking values of 0 through 255 with 32 elements ([0] to [31]).

Common “vars” are further documented in the QuickMotion manual.

Examples:

```
// wait for CVARS[22] to be > 20

while $CVARS[22] <= 20 repeat { }



// set CVARS[2] to 11

$CVARS[2] = 11;
```

3.4.8 $TASKPRIORITY

The $TASKPRIORITY variable is to set the run time priority of a task to something different from the default, 0. By default all tasks have the same priority and between each step, or during for/while loops, they yield to other tasks, sharing processor execution. At times greater performance may be needed. The $TASKPRIORITY allows

you to set a number from 0 to 100. The greater the number the more processor time the task will get based upon the $CURRENT_TASKPRIORITYLEVEL.

The setting of the $TASKPRIORITY has no immediate effect unless it is -1 or 100. A -1 will force a task yield, so others can run. A priority of 100 can only be used by a single task an makes that task the only one that will run. It will also starve some communication protocols (CTC BINARY and Modbus). A priority of 100 may be useful if a 'for' loop must be executed that has a critical timing requirement, a programmer might do the following:

$TASKPRIORITY = 100; // restrict execution to just this task.

for loop ...

$TASKPRIORITY = 0; // allow others to run now.

Reference the $CURRENT_TASKPRIORITYLEVEL variable for additional features.

Examples:

```
// Set task priority to default, 0

$TASKPRIORITY = 0;

// Yield to allow communications to run (useful if TASKPRIORITY is
100).  This value is not stored.

$TASKPRIORITY = -1;

// Set task priority to 5, a read will return a 5 as well.

$TASKPRIORITY = 5;
```

3.4.9 $CURRENT_TASKPRIORITYLEVEL

The $CURRENT_TASKPRIORITYLEVEL variable is used to activate the $TASKPRIORITY. Basically any task equal or greater than the current $CURRENT_TASKPRIORITYLEVEL setting will be allowed to run. This variable is global and common to all tasks whereas the $TASKPRIORITY is unique to each task.

For example if there are three tasks:

TASK 1 - $TASKPRIORITY = 5

TASK 2 - $TASKPRIORITY = 0; // default

TASK 3 - $TASKPRIORITY = 8;

During normal execution all tasks will run. If one of the tasks sets the $CURRENT_TASKPRIORITYLEVEL to 5 then only TASK 1 and 3 will run. TASK 2 will complete its current step or conditional loop and stop execution until its priority level is >= the $CURRENT_TASKPRIORITY_LEVEL.

.

3.4.10 $TASKHANDLE

Returns the current task handle.

3.5 MSB System Variables

'cmode' variable settings:

$CYCLIC_SYNC_POSITION_MODE	0
$PROFILE_VELOCITY_MODE	1
$STEPPER_MODE	2
$INTERPOLATED_POSITION_MODE	3
$PROFILE_POSITION_MODE	4
$PROFILE_TORQUE_MODE	5
$HOMING_MODE	8
$VELOCITY_MODE	9

Example:

```
[beginTest]
cmode = $CYCLIC_SYNC_POSITION_MODE;
```

'eCAT_driveType' variable settings:

$DRIVE_COPLEY	2
$DRIVE_YASKAWA"	3
$DRIVE_ELMO"	4
$DRIVE_KOLLMORGEN"	5
$DRIVE_SANYO_DENKI"	6
$DRIVE_EMERSON"	7
$DRIVE_AMC"	8
$DRIVE_VIRTUAL"	9
$DRIVE_IAI_ACON_MODE0"	10
$DRIVE_IAI_ACON_MODE3"	11
$DRIVE_ABB_MICROFLEX"	12
$DRIVE_MITSUBISHI"	13
$DRIVE_PANASONIC"	14
$DRIVE_LINMOT"	15
$DRIVE_YASKAWA_V1000"	16
$DRIVE_MOTIONLINX"	17
$DRIVE_WAGO_ENCODER_631"	18
$DRIVE_WAGO_ENCODER_637"	19

Example:

```
if eCAT_driveType == $DRIVE_WAGO_ENCODER_631 goto Virtual;
if eCAT_driveType == $DRIVE_WAGO_ENCODER_637 goto Virtual;
```

3.6 QS4 Statements

QS4 has a wide selection of statements that give you tremendous flexibility in building a powerful control strategy. QS4 is the latest version of CTC's proven QuickStep state language. QS4 statements and their use are reviewed in the following sections.

3.6.1 QS4 Statement syntax

QS4 steps are made up of one or more statements and are entered in QS4 graphical steps.

The syntax and definition of these statements (*instructions*) are defined in subsequent sections.

The general syntax for a statement is free-form.

Comments may be placed on a line following double-slashes "//". All text following the double-slashes (on that line) is ignored.

Multi-line comments begin with "/*" and end with "*/". For example:

```
/*
      Initialize x and y
*/
x = 10;
y = 1;
```

Statements may be placed on more than one line and indented as desired, which enhances readability.

3.6.2 QS4 Editor Color Codes

The QS4 Editor automatically applies color coding for improved readability according to the following rules:

GREEN for comments: `// Comments`
BLUE for commands: `goto`
BLACK for values and operators: `3.14 + 77`
PURPLE for keywords: `off`
DARK RED for variables: `input_conveyor`
DARK BLUE for functions: `y=sqrt(144);`

3.6.3 Assignment (numeric)

There are four assignment statements in QS4. The first form simply stores the value computed from the right side of the "=" into the location specified to the left of the "=".

For example, to store 12 into the variable named x, one would write:

```
x = 12;
```

To add two variables, a and b together and store the result in c, one would write:

```
c = a+b;
```

The second and third forms of the assignment statement increment or decrement a variable by the value computed from the right side. These two forms use to different symbols – "+=" for increment and "-=" for decrement.

For example, to decrement the variable *x* by 5, one would write:

```
x -= 5;
```

This last expression is equal to the following:

```
x = x-5;
```

The right side of the increment/decrement assignment is not limited to constant values. For example, if one wanted to increment *x* by the value of *(y-5)* one would write:

```
x += y-5;
```

The last form of the assignment statement is to change the *address* of an indirect variable:

```
x <- addr(y);
```

This would set the indirect variable *x* to "point to" the variable *y*.

3.6.4 Assignment (string)

When assigning strings to a variable, a special notation must be used:

```
x = string("Hello There");
```

This would assign the variable *x* the value "Hello There". This notation is only required when dealing with *expressions* that evaluate to strings.

For example, to concatenate two strings (contained in variables *string1* and *string2*) and store the result in *string3*, one would write:

```
string3 = string(string1 + string2);
```

When a function that returns a string is utilized, the **string()** notation must also be used:

```
string2 = string(left(string1, 5));
```

The above example would take the leftmost 5 characters of *string1* and store them into *string2*.

Data types can be mixed where integers, floats and doubles will automatically be converted to a string:

```
x = string("The total count is:  " + sum);  // where sum is an integer variable, float, and double can also be referenced
x = string("" + sum);  // where just the integer value is desired, thus add an empty string.
```

Array Indexing is supported for strings where x is a string vector table:

x[0] = **string**("Hello There");
x[1] = **string**("Goodbye");

Concatenation Operator '+':

y = **string**("Joe");

x = **string**("Hello There" + y);

Where the resulting string would be "Hello There Joe". Note that the quotes are not part of the string stored.

Functions that *do not return* strings *must not* be enclosed in **string()**. For example, the function that returns the length of a string (**len**) cannot and should not be enclosed in **string()**:

```
x = len(left(string1, 5));
```

The variable *x* will be set to the length of the result of the **left()** operation – one would assume that *x* will be 5, but it can be any value from 0 to 5 depending on the value of *string1*.

3.6.5 Store

The ***store*** statement is fundamentally the same as the first form of the assignment statement. It remains as part of the QuickStep syntax for compatibility with QS2.

The syntax of this statement is:

```
store expression to location;
```

The ***store*** statement can also be expressed as an assignment:

```
location = expression;
```

Expressions must follow the same rules for strings as mentioned in the previous section on *string assignment*.

3.6.6 Set

The ***set*** statement alters a list of digital outputs or booleans.

The syntax of this statement is:

```
set list on | off | ON | OFF;
```

For example, to turn two outputs named *dig1* and *dig2* on, one would write:

```
set dig1,dig2 off;
```

The *list* is comma-delimited.

3.6.7 Setbit, Clrbit

These two instructions manipulate bits (formerly referred to as *flags*) within a controller. Bits have two logical values, *true* and *false*.

Bits are numbered from 0 upwards to 31.

You can also use them to store Boolean information just as flags were used in the QS2 world. Since one register can hold 32 bit values, then the number of Boolean bits available to the user is quite high in comparison to QS2.

Multiple tasks can set/clear bits in the same variable, at the same time and receive the correct results.

The syntax of these statements is:

```
setbit BITn variable;
setbit n variable;
clrbit BITn variable;
clrbit n variable;
```

For example, to set bit 5 in a variable named *x*, one would write:

```
setbit BIT5 x;
```

The prefix *BIT* is optional – therefore the last example could also be written:

```
setbit 5 x;
```

To clear bit 10 in a variable named *x*, one would write:

```
clrbit 10 x;
```

One can test the value of a bit using the *bit()* function.
For example, the follow sets the variable *y* to 1 (*true*) when bit 5 in the variable x is on (*true)*:

```
y = bit(x, 5);
```

3.6.8 Goto

The ***goto*** instruction tells the controller which step to execute next. The destination step specified in the instruction must be within the same task.

The syntax of this statement is:

```
goto destination;
```

Once executed, the existing step is exited, other steps allowed to execute if pending, and then the branch finally occurs.

The following example instructs the controller to jump to the step named *SHUT_DOWN*:

```
goto SHUT_DOWN;
```

A special form of this instruction simply tells the controller to proceed to the next step in sequence:

```
goto next;
```

3.6.9 Call, Return

The ***call*** instruction tells the controller to invoke a user-defined function and return control to the next instruction upon completion.

Calling a function repeatedly, without executing a ***return*** instruction will result in a controller fault, namely a *stack overflow*. This will occur if you use a ***call*** instruction repeatedly and never ***return*** to the calling step.

The ***call*** instruction can pass up to 9 parameters and also return a single *integer* variable.

The various forms of the ***call*** statement are:

```
call function;
call function store to result;
call function (param1, param2, ...);
call function (param1, param2, ...) store to result;
```

The comma-separated parameter list (*param1, param2, ...*) can be a mix of variable names and/or constants (numbers and strings). The returned *integer* result can be stored in the optionally specified variable *result*.

In order to return control (from within the called routine), a ***return*** instruction must be executed.

The two forms of the ***return*** statement are:

```
return;
return result;
```

The first form returns a zero value, and the second form returns a specific result.

⚠ A 'call' function counts as an active task. Thus a single task which invokes a function is now 2 tasks until you return from the function call. You are limited to 96 total active tasks, functions & events.

3.6.10 If/Then/Else

The ***if*** instruction tests a conditional expression for a true (non-zero) value.

In QS2, the ***if*** instruction only included a single operation, namely a ***goto***. In QS4, ***if*** statements can be made up of multiple lines. QS4 also supports ***else*** logic.

The *if* statement has two forms:

```
if expression goto step;
if expression then statement;
```

In the first form (the *goto* form), the controller will jump to the step named *step* if the expression evaluates to a non-zero (true) value. This form is *not* allowed to have an *else* clause.
In the second form (the *then* form), the controller will execute the listed statement if the expression evaluates to a non-zero (true) value.

The special target step *next* can be used as the destination for the *goto*.

As mentioned earlier, QS4 also allows a multi-line ***if***. This form is written:

```
if expression then {
      statement;
      statement;
      ...
```

```
}
```

In this form, when the *expression* evaluates to a non-zero value, each of the statements are executed. The ***if*** statement (in either the *goto* or *then* forms) can optionally be followed by an ***else*** statement which is written:

```
else statement;
```

If the associated *if* evaluates to false, then the *else* statement is executed. Just like the *if* statement, the *else* statement supports a multi-line format:

```
else {
      statement;
      statement;
      ...
}
```

Multiple *if* statements can be logically nested to form *if/else-if/else-if/…else* constructs. For example:

```
if expression1 then {
      statement1;
      statement2;
      ...
}
else if expression2 then {
      statement3;
      statement4;
}
else if expression3 then {
      statement5;
      statement6;
}
else {
      statement7;
      statement8;
}
statement9;
```

In this case, *expression1* is evaluated. If *expression1* is true, then statements 1 & 2 are executed followed by statement 9. If it is false, then *expression2* is evaluated – if true then statements 3 & 4 and executed followed by statement 9. If *expression2* is false as well, then *expression3* is evaluated – if true, then statements 5 & 6 are executed followed by statement 9. If *expression3* also evaluates to false, then statements 7 & 8 are executed followed by statement 9.

Examples:

```
if x >= 8 then goto ProcessComplete; else goto next;

if dig1 then {
       set digout1 off;
       x = 0;
}
```

```
else x = 1;
```

If instructions are not atomic when contained in a step, but the *body* of the *then* or *else* instruction *is* atomic.

⚠ Note that if a negative number is used in a conditional statement it must be surrounded by parenthesis, (...). For example x >= (-8). Failure to do so will typically result in a 0 literal.

3.6.11 While

The ***while*** instruction repeatedly tests a conditional expression and executes one or more statements while true.

The *while* instruction operates as follows:

If the condition initially evaluates to true (non-zero) then the statement or statement block is executed.

Once the statement or statement block has been executed, the conditional expression is evaluated again and if still true, then the statement or statement block is again executed.

The process repeats over and over again until the conditional expression evaluates to false (a zero value).

The two forms of the *while* statement are:

```
while expression repeat statement;

while expression repeat {
      statement;
      statement;
      ...
}
```

While instructions are not atomic when contained in a step, but the *body* of the *while* instruction *is* atomic.

⚠ Note that if a negative number is used in an *expression* it must be surrounded by parenthesis, (...). For example x >= (-8). Failure to do so will typically result in a 0 literal.

3.6.12 Repeat/Until

The ***repeat…until*** instruction executes a statement or statement block and then tests whether or not a given condition evaluates to true (non-zero). If the condition is true, then control exits the *repeat*. If the condition is false (zero), then the statement or statement block is executed again.

Once the statement or statement block has been executed again, the conditional expression is evaluated again and if still false, then the statement or statement block is again executed.

The process repeats over and over again until the conditional expression evaluates to true (a non-zero value).

The two forms of the *repeat* statement are:

```
repeat statement; until expression;

repeat {
```

```
        statement;
        statement;
        ...
} until expression;
```

Repeat instructions are not atomic when contained in a step, but the *body* of the *repeat* instruction *is* atomic.

⚠ Note that if a negative number is used in an *expression* it must be surrounded by parenthesis, (...). For example x >= (-8). Failure to do so will typically result in a 0 literal.

3.6.13 For

The ***for*** instruction executes a statement or statement block while at the same time iterating a variable over a range of values.

For instructions are not atomic when contained in a step, but the *body* of the *for* instruction *is* atomic.

The four forms of the *for* statement are:

```
for variable = start to end repeat statement;
for variable = start to end by increment repeat statement;

for variable = start to end repeat
{
        statement;
        statement;
        ...
}

for variable = start to end by increment repeat
{
        statement;
        statement;
        ...
}
```

For example, to iterate the variable "i" from 1 to 10 (inclusive) by an increment of 1 (one):

```
for i = 1 to 10 repeat
{
        statement;
        statement;
        ...
}
```

To iterate the variable "i" from 0 to 6 (inclusive) by steps of 2 (0, 2, 4, 6):

```
for i = 0 to 6 by 2 repeat
{
        statement;
        statement;
```

```
        ...
}
```

To iterate the variable "i" from 10 to 1 (inclusive) decrementing each time:

```
for i = 10 to 1 by -1 repeat
{
        statement;
        statement;
        ...
}
```

3.6.14 Break

The ***break*** instruction aborts a *while*, *for* or *repeat* instruction. When this instruction is encountered, no more looping or iterating occurs and the "exit" condition (such as the *while's until* condition) is not tested.

The syntax of the *break* statement is:

```
break;
```

In the following example, the variable "i" is iterated from 1 to 100. If however, the variable "x" exceeds 100, then no more iteration occurs:

```
for i = 1 to 100 repeat
{
        ...
        if (x > 100) then break;
        ...
}
```

In the following example, statements 1 and 2 are executed while the variable "digin1" is true. If, however, the variable "y" equals -1, then the while loop is exited and statement 3 is executed.

```
while digin1 repeat {
        statement1;
        statement2;
        if y == -1 then break;
        ...
}
statement3;
```

3.6.15 Continue

The ***continue*** instruction immediately begins the next iteration ("loop") of a *while*, *for* or *repeat* instruction. When this instruction is encountered, no more statements are executed until the iteration condition is tested (*while* and *repeat*) or the next iteration begins (*for*).

The syntax of the *continue* statement is:

```
continue;
```

In the following example, the variable "i" is iterated from 1 to 100. If however, the variable "x" exceeds 100 (after statement 9 is executed) then statement 10 is not executed but the next iteration of "i" occurs instead:

```
for i = 1 to 100 repeat
{
        ...
        statement9;
        if (x > 100) then continue;
        statement10;
}
```

3.6.16 Delay

The ***delay*** instruction is similar to the *delay* instruction in QS2, except there is no branch operation (reference 'timeout' for compatibility). This instruction causes the controller to proceed after a specified amount of time has passed.

This time delay is specified as:

```
delay expression ms;
```

Note: the internal resolution of the controller is 1 millisecond and all times will be rounded appropriately.

3.6.17 Timeout

The *timeout* instruction is similar to the *delay* instruction available in QS2. This instruction causes the controller to proceed after a specified amount of time has passed.

This time delay is specified as:

```
timeout expression ms goto label;
```

The big difference from the *delay* instruction is *timeout* runs in the background. The first time it is executed, within a step, it is initialized, every execution after that is a test to see if the *timeout* occurred, resulting in the branch if it did. You must re-execute in order to test the state of the timer. Also if multiple *timeout* instructions are executed within a step, the last one if the one active but all will be tested. Exiting a step will clear the timer.

Note: the internal resolution of the controller is 1 millisecond and all times will be rounded appropriately.

3.6.18 When

The ***when*** instruction is simpler in form than the ***if*** instruction but more efficient in operation.

The syntax for the *when* instruction is:

```
when expression goto step;
```

If the *expression* is initially *true* (non-zero) a branch will occur to the specified step *step* immediately – if it is *false* (zero) the *expression* will be tested the next time the task is allowed to run.

Once the *expression* finally evaluates to *true* a branch will occur to the specified step.

The *when* instruction is *not* equivalent to the *monitor* instruction in QS2 – the *when* instruction *is* equivalent to the *transition* condition used in QS4 steps.

⚠ Note that if a negative number is used in an *expression* it must be surrounded by parenthesis, (...). For example x>=(-8). Failure to do so will typically result in a 0 literal.

3.6.19 Enable, Disable (Event)

Events and their associated functionality often require program-level enabling and disabling.

For example, there may be a specific point in a QuickStep program where an *event* should not be allowed to occur (initialization, during fault-handling, etc.). For this reason, a couple of QS4 instructions were creating to facilitate enabling and disabling of these events.

The two forms of this instruction are:

```
enable event event;
disable event event;
```

The *enable* and *disable* instructions are also used with other resource variables and are covered later in this document.

All events are initially *disabled* when program execution begins – it is required that a task programmatically enable some or all events as required.

Example:

```
enable event MY_event;
disable event MY_event;
```

⚠ Note that an 'event' counts as an active task. You are limited to 96 total active tasks, functions & events.

3.6.20 Do

The ***do*** instruction starts one or more *child* tasks.

The *parent* task (that executed the ***do*** instruction) will not continue execution until a ***done*** instruction is issued by each child task.

The two forms of this instruction are:

```
do ( task task ... )
do task with expression, expression, ...
```

The first form starts the listed *child* tasks but waits until each *child* task has finished executing before continuing execution.

The second form starts a single *child* task passing a series of parameters to the task. The *parent* task will wait until the single *child* task has finished before continuing execution.

The parameter types and storage must match the *signature* of the task being started – that is if the task being started takes two parameters (for example, a scalar integer and a scalar double), then the calling parameter types must match.

3.6.21 Begin

The ***begin*** instruction starts one or more *child* tasks.

Unlike the ***do*** instruction, the ***begin*** instruction does not wait until the child task(s) complete(s) before execution continues – the task(s) is/are started and control is returned immediately to the *parent* task for continued execution.

Since more than one instance of a task can be started simultaneously, when tasks are *begin-ed* the QS4 runtime returns a *task handle* for each started task.

When a single task is started with *begin*, the user can store a *task handle* in a *scalar* variable – if multiple tasks are *begin-ed*, the user can store these series of *task handles* in a ***vector*** variable.

If the *task handles* are not needed later in program execution, they do not need to be stored.

The first two forms of this instruction are:

```
begin ( task task ... ) ;
begin ( task task ... ) tasks to variable ;
```

These two forms of the *begin* instruction start a series of tasks. The second form allows the user to store the series of returned *task handles* into a ***vector*** variable.

The last four forms of this instruction are:

```
begin task ;
begin task task to variable ;
begin task with expression, expression, ... ;
begin task task to variable with expression, expression, ... ;
```

The first form simply begins a task.

The second form begins a task and stores the *task handle* for the started task into the specified (integer) *scalar* variable.

The third form begins a task with a list of parameters. Just like its counterpart *do*, the parameter types and storage must match the *signature* of the task being started – that is if the task being started takes two parameters (for example, a scalar integer and a scalar double), then the calling parameter types must match.

The fourth and final form begins a task with a list of parameters – in addition, a variable is specified for holding the resulting *task handle*. The same rules apply for this form – the specified variable must be (an integer) *scalar*, and the parameters must match the *signature* of the task being started.

3.6.22 Cancel

The ***cancel*** instruction stops one or more *child* task specified in the instruction itself or all running tasks (2nd form) or all *other* tasks (3rd form).

The two forms for this instruction are:

```
cancel variable;
cancel all tasks;
cancel other tasks;
```

The first form of this instruction cancels one or more tasks:

- If the named *variable* is a *vector* variable, then it is assumed to contain a series of *task handles* – each of these tasks will be stopped.
- If the named variable is a *scalar* variable, then it is assume to contain a single *task handle* – the single task will be stopped.

The second form of this instruction cancels all the running tasks *including* the task that is executing the step containing this instruction.

The third form of this instruction cancels all the running tasks *except* the task that is executing the step containing this instruction.

3.6.23 Done

The ***done*** instruction stops *this* task – the task executing the ***done*** instruction. The ***done*** instruction also informs a possibly waiting *parent* task that this *child* task has been completed.

The only form for this instruction is:

```
done;
```

3.6.24 Start

The *start* statement begins execution of the named *motion sequence block* (MSB) on the specified *axis*. The MSB is started as a *background* (**BG**) MSB. If the named *MSB* is already active on the axis, then the statement is effectively ignored.

```
start axis MSB;
```

See the *QuickMotion Reference Guide* for additional information on motion control programming.

3.6.25 Stop

The *stop* statement ends execution of all foreground (**FG**) and background (**BG**) MSBs that are active on the specified *axis*.

```
stop axis;
```

See the *QuickMotion Reference Guide* for additional information on motion control programming.

3.6.26 Soft Counters

Reference Chapter 3. This is an enhancement added to support Quickstep 2/3, Soft Counters.

3.6.27 Rotate, Shift Flags

Quickstep 2/3 compatible flag rotate instructions:

rotate <QS2flag> << or >> <QS2flag> [number times] range must be on 32 bit boundry 1 to 31, 32 to 63, etc.

The *ROTATE* instruction replaces the status of a flag (either set or clear) with the status of the flag preceding or following it. The first flag in the series inherits the status of the last flag in the series.

shift <QS2flag> << or >> <QS2flag> [number times] range must be on 32 bit boundry 1 to 31, 32 to 63, etc.

The *SHIFT* instruction replaces the status of a flag (either set or clear) with the status of the flag preceding or following it. The first flag in the series is automatically cleared.

Note: <QS2flag> - Overridden register variables from 13201 to 13228 representing individual flag registers within the controller.

4 Chapter 4: Importing QuickStep 2/3 Projects

Quickstep 2/3 programs can be imported into the QuickBuilder programming environment. Quickstep programs were those programs used on controllers such as the 5100/5200/2600/2700 series as well as the 5300. The 5300 also offers the more advanced QuickBuilder programming environment. Importing a Quickstep program will allow it to run on the 5300. When moving from the Quickstep to the QuickBuilder environment a few things need to be noted:

Quickstep:

- Each step executes an 'assignment instruction', such as 'store', only once during step execution. A function 'call' will always executed and yield to other steps.
- If a 'goto' is not executed the step will loop upon itself executing only the 'if', 'while', 'when', 'repeat', 'for', 'call', and other conditional type statements. The conditional is tested during each loop but the assignment operations are only done the first loop, even if the conditional was not satisfied. As long as the conditional is satisfied on the first loop the assignment instructions within that conditional will be executed.
- You must execute a branch instruction to exit a step block, the visual links connecting the blocks serve no function other than specifying which task the step belongs to.
- Execution will be atomic within a step until the end of the step is reached, a branch occurs, or a 'while/when' loop begins to execute again (after does the first loop within itself).
- Monitor instructions are now If conditional instructions.
- Quickstep referenced a single data table of unsigned 16 bit integers, QuickBuilder allows for numerous data tables, both volatile and non-volatiles and with differing types, 32 bit integers, float64, and strings.
- QuickBuilder supports all previous Quickstep instructions natively or modifies syntax to a new format where required. This includes native support for TURN, ZERO, SEARCH AND ZERO, PROFILE, SCOUNT (previously COUNT), ROTATE, and SHIFT. MONITOR is now an IF statement. A new instruction of TIMEOUT replaces the prior Quickstep DELAY and works the same.
- Quickstep allowed tasks to execute code in other tasks while QuickBuilder did not. In other words a task can jump into the code of another task and execute it as though it is its own. This tends to make the code difficult to follow and debug. Since during the import unique step names exist this rule has been made flexible and is now allowed, but not recommended. A warning will be generated during translation and can be ignored.
- Quickstep allowed 40 characters for step and task names, QuickBuilder allowed 16. QuickBuilder was expanded to 40.
- Quickstep allows to multiple symbols to reference the same digital input, active high and active low. QuickBuilder only allows a single symbol definition. To enable the import to function properly this restriction was relaxed to allow the multiple references to be listed in the digital input resource tree. Once deleted it cannot be re-created and QuickBuilder will not allow you to do this within a normal project. It is meant to support the initial Quickstep import but going forward should not be used.

QuickBuilder:

- Each step executes an ‘assignment instruction’, such as a ‘store’, every time executed, similar to procedural programming logic.
- If a ‘goto’ is not executed the step will branch to the next step, connected to by the visual link, if none then the task will execute a ‘done’ command.
- Execution will be atomic within a step until the end of the step is reached, a branch occurs, or a ‘while’, ‘when’, ‘repeat’, ‘for’ loop begins to execute again (after does the first loop within itself). A ‘delay’ and’call’ instruction always yields to other steps.
- The visual links between step blocks are functional when using QuickBuilder mode.
- Quickstep allowed 40 characters for step and task names, QuickBuilder allowed 16. QuickBuilder was expanded to 40. This will cause an old program that is loaded into the new version of QuickBuilder to have its steps visually overlay each other, due to a change in spacing. Select the ‘View-> Autoformat Grouped Area’ menu item to reformat the screen. This only needs to be done once. Each Page will have to be reformatted.

Prior to attempting the import of a Quickstep program the program must be fully compiled using Quickstep and contain no errors. All generated files must be present within the same directory. During import only the Quickstep Booleans of AND, OR, ANDNOT, and XOR are supported. Every attempt is made to execute the imported Quickstep programs in a similar manner as before but all possible deviations can not be 100% tested. It is important to fully test your program prior to deployment into a production environment. For example something as simple as speed of execution may cause problems with some programs, depending upon how they were written.

4.1 Datatables

If a data table is present within the imported Quickstep program a non-volatile variant table, register 36800, is referenced within the QuickBuilder generated code called ‘_datatable’, with a data type of 32 bit signed integer. The table will not be created, only referenced, and it either needs to be created by your program or by using external means, such as using the Quickstep generated .tab file. Not keeping the data table as part of the program is on purpose, since the table is non-volatile, you do not want to initialize it every time a new program is loaded or modified. Quickstep use to keep it as part of the program and whenever a new Quickstep program was loaded into the controller the data table had to be backed up and restored, manually, given the new Quickstep program would overwrite the values in the table.

If a default value of ‘0’ is valid for the table then it can be created programmatically by writing to the last cell of the desired table size:

```
_datatable[lastrow-1][lastcolumn-1] = 0;
```

Assuming the original Quickstep .tab file will be referenced, then that file must be placed on the controller at the ‘datatablespath’ location. By default this is “/_system/Datatables” directory, and resides in flash. If you wish this to be changed to RAMDISK, or some other location you can change the default by using:

```
‘set datatablespath myDatatablesDir
```

Where myDatatablesDir is a user defined path.

Once the Quickstep .tab file has been placed in the proper directory, typically by FTP, the file can be loaded directly into the variant register using the 'load datatable 36800 qs2.tab' telnet command, any existing data table will be erased and the new table created with the values residing within the referenced file.

load datatable 36800 qs2.tab (note that qs2.tab is whatever the file name was generated as Quickstep compilation output).

4.2 Motion Control

QuickBuilder uses MSB's for motion control. Quickstep had a number of commands which were specific to motion control. When a Quickstep program is imported into QuickBuilder two MSB's are automatically included from a library (<InstallationDir>\Resources\QS2_Motion.qbl) which provides an interface for these instructions as well as the Quickstep Motion Registers, such as 14/15/17000 register blocks. These MSB blocks can be customized by the user, as needed. These two automatically generated MSB's are known as the Quickstep Motion Simulation Environment. Note that the user units of Quickstep were counts whereas that of QuickBuilder can be user defined, typically revolutions. QuickBuilder also supports 64 bit data whereas Quickstep was only 32 bit integers. This restriction still applies to an imported program, although as modifications are made full access to the 64 bit data is available as with any QuickBuilder program. The restriction is only in the use of the legacy motion commands, such as TURN, ZERO, PROFILE, and SEARCH AND ZERO.

In order to support the Quickstep MONITOR and IF instructions a function called _servoInfo( ) has been added to QuickBuilder in order to interface with the motion control Quickstep simulation environment. Its accepts two parameters, the first the motion axis name with a property of .Axis following it, which generates an axis number, 1 to N. The second parameter what information is desired. The second parameter can be hard coded or what the importer does, defined as an XVar constant:

_servoInfo(<Axis>.Axis, request) returns a 1 or 0 based on true or false where 'request' is one of following:

XVars defines:
__CTC_SERVO_ERROR with constant value of 8
__CTC_SERVO_POSITION with constant value of 7
__CTC_SERVO_RUNNING with constant value of 5
__CTC_SERVO_STOPPED with constant value of 6

Note: __CTC_SERVO_POSITION returns a 32 bit signed count value representing the current position.

Quickstep Motion instructions now supported in the QuickBuilder environment:

<Axis> - M340 axis name defined in the resource tree.
<cw/ccw> - Either select cw for clockwise, or ccw for counter clockwise rotation.

motion <Axis> hardstop/softstop

The STOP SERVO instruction brings the servo to a halt. You can choose one of the following methods to stop the servo:
• softstop - Causes the servo to stop at the deceleration rate specified in the last profile instruction.

Example:
motion servo_2 softstop

• Hard Stop - Causes the controller to attempt to stop the servo instantly. However, because of momentum (caused by the inertial load), the servo does not stop instantly and consequently the absolute position may be lost and instability may result.

Example:

motion servo_1 hardstop

In either case, you should use a monitor instruction before issuing another turn instruction.

turn <Axis> to <Expression>
turn <Axis> <cw/ccw> [<Expression> steps]
turn <Axis> <cw/ccw>

The TURN SERVO instructions initiate a new servo motion. The controller must have executed a PROFILE SERVO instruction to define the motion parameters. TURN SERVO defines the distance the servo travels using one of the following methods:

• Absolute — Turns the servo a calculated number of steps based on the distance from a predetermined zero position.

Example:

turn servo_1 to 1500
turn servo_2 to 1500 on_start_switch

The second absolute distance instruction also requires a contact closure on the servo module's dedicated start input before the servo motion begins.

• Relative — Turns the servo clockwise or counter clockwise a specified number of steps from the current motor position.

Example:

turn servo_5 cw 70000 steps

NOTE: step = encoder signal transition

• Velocity — Begins continuous clockwise or counter clockwise motion. The servo remains in motion until the controller executes a 'motion <axis> hardstop/softstop' instruction or the servo control module senses a stop input signal.

Example:

turn servo_5 ccw

NOTE: Do not issue another TURN SERVO instruction for a servo while the servo is still in motion. If the servo is still turning, the controller reports a software fault (servo not ready) and halts execution of the program. Before issuing another turn instruction, you should program a monitor servo:stopped (_servoInfo(<axis.axis>,__CTC_SERVO_STOPPED) instruction prior to any subsequent turn instruction.

profile <Axis> servo at position [maxspeed = <Expression>] [accel = <Expression>]
[P = <Expression>] [I = <Expression>] [D = <Expression>]

profile <Axis> motor off at position [maxspeed = <Expression>] [accel = <Expression>]
[P = <Expression>] [I = <Expression>] [D = <Expression>]

profile <Axis> deadband of <Expression> at position [maxspeed = <Expression>] [accel = <Expression>]

[P = <Expression>] [I = <Expression>] [D = <Expression>]

profile *<Axis> [maxspeed = <Expression>] [accel = <Expression>][P = <Expression>] [I = <Expression>] [D = <Expression>]*

Note: P/I/D information is not presently used but is passed to the Quickstep Motion Simulation MSB.

The Profile Servo instruction sets the motion parameters for a servo as follows:

- Maximum Speed (max) — Establishes the maximum speed of the servo. It is defined in encoder pulse edges (steps) per second (fully decoded).
- Acceleration Rate (accel)— Specifies the acceleration rate of the servo. Defined in encoder pulse edges (steps) per second per second (steps/sec2). This parameter also sets the deceleration rate. If you want the acceleration and deceleration values to be different, store the deceleration value to a special purpose register. The following example sets the deceleration rate:

Example:

profile servo_1 max=50000 accel=100000
store 20000 to reg_15006 (axis No. 1 deceleration register)

- P (Proportional) Filter - Specifies the factor applied to the sensed position error to create a correction signal. It is expressed as a multiplication factor from 0 to 255. (Not currently used).
- I (Integral) and D (Derivative) Filters - Determine the characteristics of the built-in digital compensation filter. (Not currently used).
- Holding Mode - Specifies the status of the servo when stopped, using one of the following parameters:
  - Servo at position - Once the servo reaches the desired position, the actuator will continuously seek this position. If the actuator is forced from its position, the servo control module sends a correction signal to attempt to correct the perceived error.
  - Deadband of __ at position - The servo control module senses position errors but does not correct them unless the error is out of the range of the Deadband.
  - Off at position - Once the servo reaches position no further corrective action occurs. This allows manual adjustment or another external force to change the position of the servo.
  - None - Indicates that the controller should use the holding mode specified in a previous PROFILE SERVO instruction.

The PROFILE SERVO instruction does not start the servo motion. To initiate motion use the TURN SERVO instruction. You may respecify the servo profile parameters any number of times in the same program. Any of the numeric parameters for a servo motor can be drawn from any of the controller's numeric resources, instead of being expressed as a fixed number. For additional information, refer to the section on the STORE instruction.

Unlike a stepping motor, you can execute a new PROFILE SERVO instruction while the servo is still in motion. You can change any parameter except the acceleration rate.

Example:

profile servo_3 servo at position maxspeed=15000 accel=35000 P=10 I=95 D=50

search and zero *<Axis>*

The SEARCH AND ZERO SERVO instruction sets a zero or home reference position for a servo. The SEARCH AND ZERO SERVO instruction starts the servo turning at the rate specified in the PROFILE SERVO instruction until the servo control module senses a contact transition on its home limit switch

input (dedicated input). Depending on the model of the servo control module you have, the instruction functions differently.

Example:
search and zero servo_1

zero <*Axis*>

The ZERO SERVO instruction sets the current position of the servo as its zero or home reference position.

Example:
zero servo_1

⚠ Important: When doing motion control you must tune your motor and set the properties for each axis prior to operation (M3-40 axis). Also make sure you set the driveenable property to the proper output to enable the drive, it is disabled, 0, by default. CTC demo boxes use output #1, thus a change is needed.

4.3 Variables

When importing a Quickstep program some variables are automatically created, other than the XVar constants referenced in the Motion Control section, another is __qb_local01. This is a scratch variable for internal use only and is assigned to use the task local register 36089.

4.4 Soft Counters

The counter instructions control the eight internal counters in a controller. These instructions can start a counter and increase or decrease the value in one of the controller's counters. They also reset, enable, and disable any of the controller's counters.

<counter> - Where counter a register overridden as 1 to 8 or variable that contains the counter number.

scount up <*counter*>

COUNT UP adds one to the current value in the counter.

Example:
scount up ctr_3

scount down <*counter*>

COUNT DOWN subtracts one from the current value in the counter.

Example:
scount down ctr_3

scount enable <*counter*>

Count Enable reactivates a counter that was temporarily disabled.

Example:

scount enable ctr_3

scount disable <counter>

Count DISABLE temporarily disables a counter so that it does not accept any count up, count down, or reset pulses until it is enabled.

Example:

scount disable ctr_3

scount reset <counter>

Count Reset returns the value in the counter to zero.

Example:

scount reset ctr_3

scount start <counter> [up <input>] [down <input>] [reset <input>]

Count Start initializes a counter. The counters overlay the first eight registers (i.e., counter No. 1 = register No. 1). When starting the counter,
you can assign three of the controller's inputs to perform the count-up, count-down, and reset functions:

scount start ctr_1 up in_5A down in_6B reset in_7A

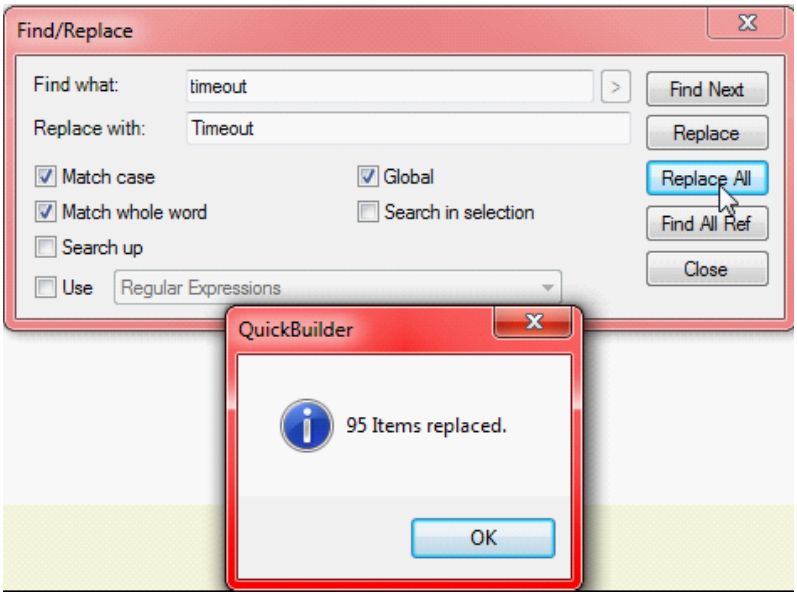
This initializes the counter and assigns functions to three of the controller's inputs. These inputs continue sending input signals to the counter until the counter is re-initialized or disabled.

— Up in_5A, specifies that input No. 5 is being used for the count up function and increments the counter for each switch closure. The A specifies that the input is a normally-open input. A normally-open input means that the count occurs when the switch closes.
— Down in_6B, specifies that input No. 6 is being used for a countdown function and decreases the value in the counter by one for each time the switch opens. The B specifies that the input is a normally closed input. A normally-closed input means that the count occurs when the switch opens.
— Reset in_7A, resets the value in the counter to zero when the switch closes.

Note: The De-bounce available in Quickstep Start Count instruction is not supported and is removed upon importing. Also the DigitalInput 'activeState' property is not referenced, using the default 'true' value. 'activeState' is only applied for normal register read operations.

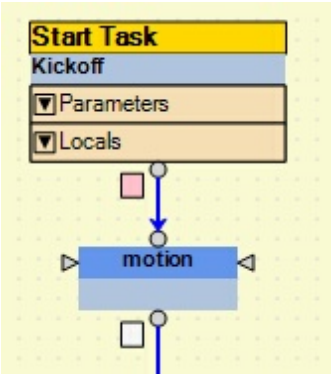
4.5 Reserved Words Error Search

A Quickstep program may use variable and/or step names which are reserved by QuickBuilder. Sometimes these can be difficult to find. The correction can be made by simply searching for all instances of the reserved word and replacing it with a new one. If the reserved word was used as a variable the variable name should be changed first, for example 'timeout' changed to 'Timeout' or 'position' changed to 'Position'. Then attempt a translation and it will fail due to a syntax error. A dialog box will display what task and step the error is in. Scroll through that task until you see the red underlines under the instructions, indicating a syntax error. Assuming you have more than one place you use this variable, select the 'Editor' tab and you will now have a full text editor display of multiple steps. Hold the Control key down and hit 'F', CNTRL F, and a find/replace screen will display. Fill out the dialog as desired and select 'Replace All':



Your entire program, all pages, will be searched for what desired and replaced accordingly. Be sure to select 'Match case', 'Match whole word', and 'Global' prior to selecting the 'Replace All' button. Close the Find/Replace screen and select 'Translate' again, repeating above for each error instance until your program fully translates.

In looking for step names that may be using a reserved word as its name typically the link prior to that having a reserved word will be pink versus white. For example 'motion' is a reserved word, below shows the error:

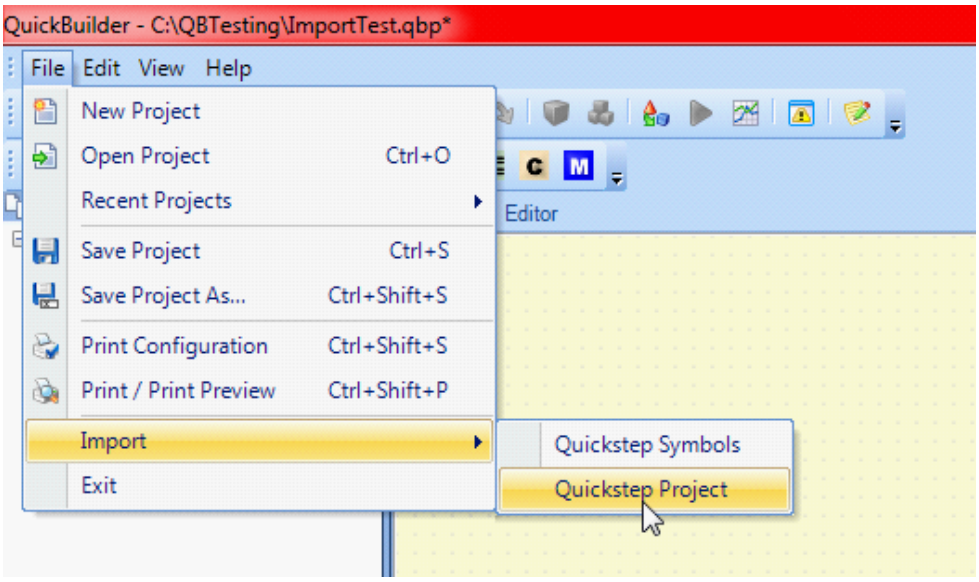


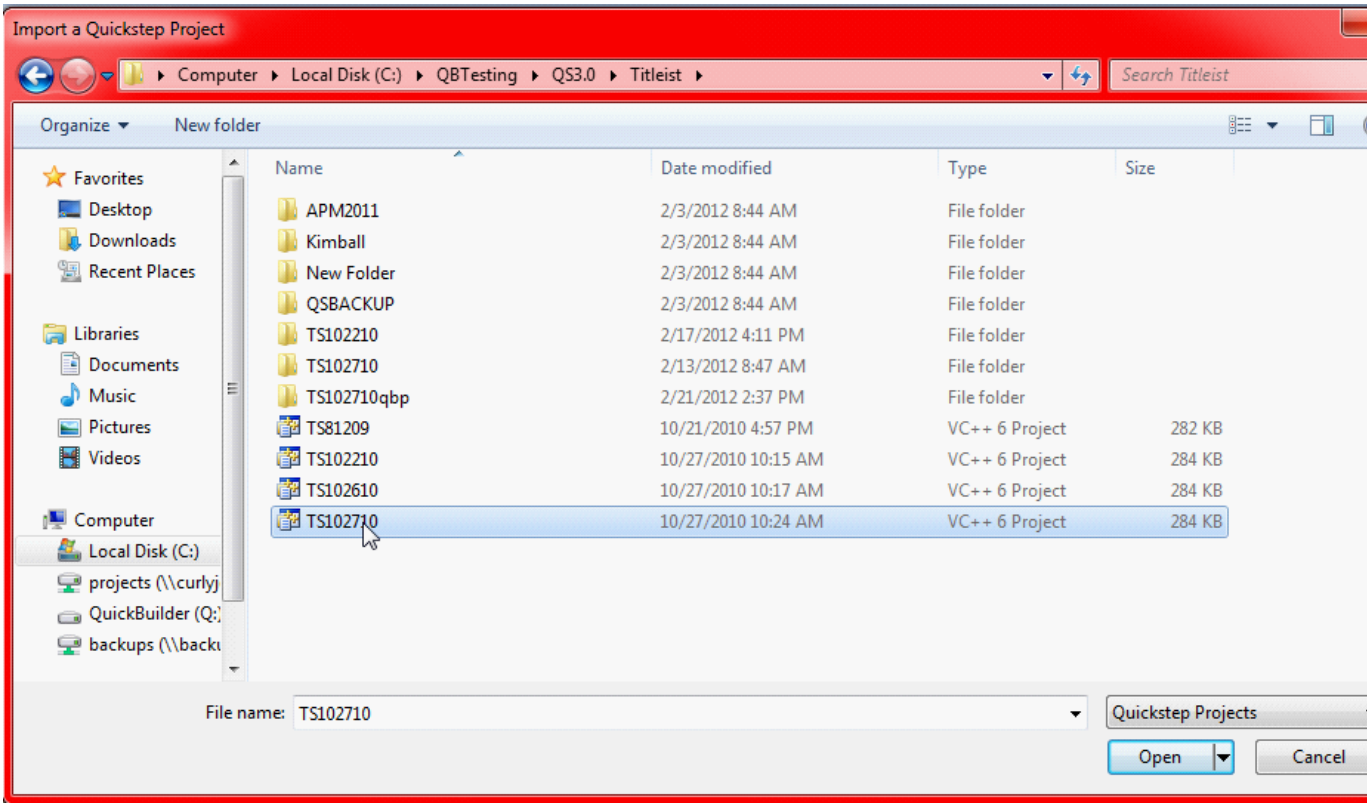
To correct this problem the step named 'motion' would have to be edited, for example changed to 'motionx', as well as any references. To resolve the pink box and clear that error simple double click on the box and the parser will then see the step name has been changed and clear the error. Sometimes you need to click the clear box below as well if an error still exists.

4.6 Importing

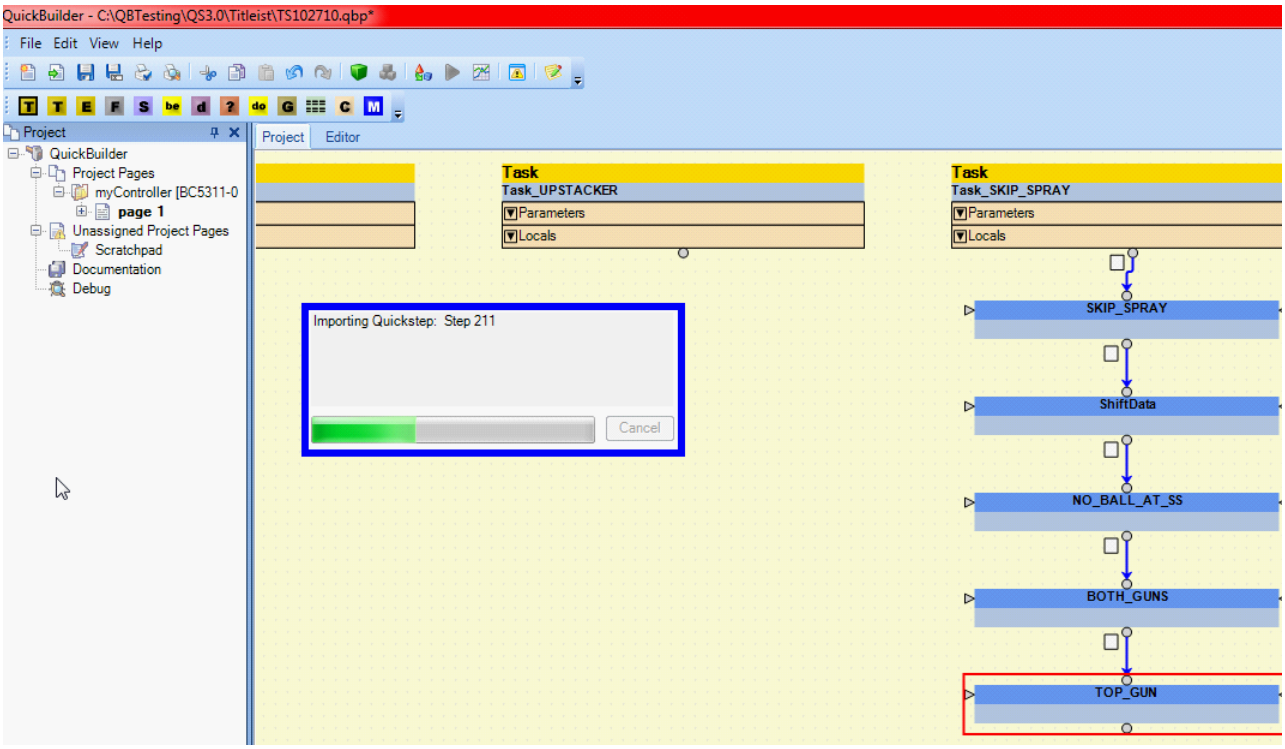
To import a Quickstep project first create an empty project with the proper controller modules in place. **Save that project with no steps or tasks present, now re-open the empty project**. If the project is imported and there are not

enough resources you will be warned to reference the unassigned folder for resources not defined. Re-open the saved project, added the needed resources, and attempt the Import again until adequate resources exist.

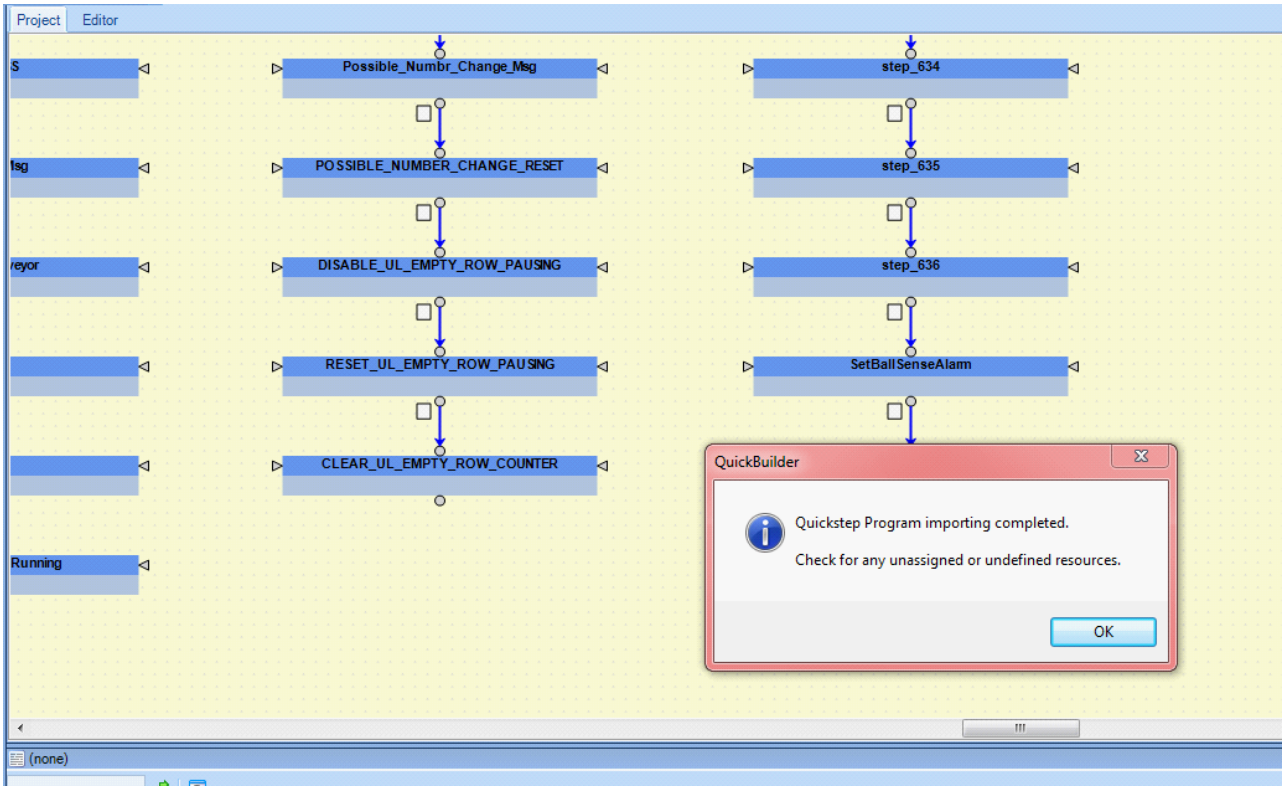




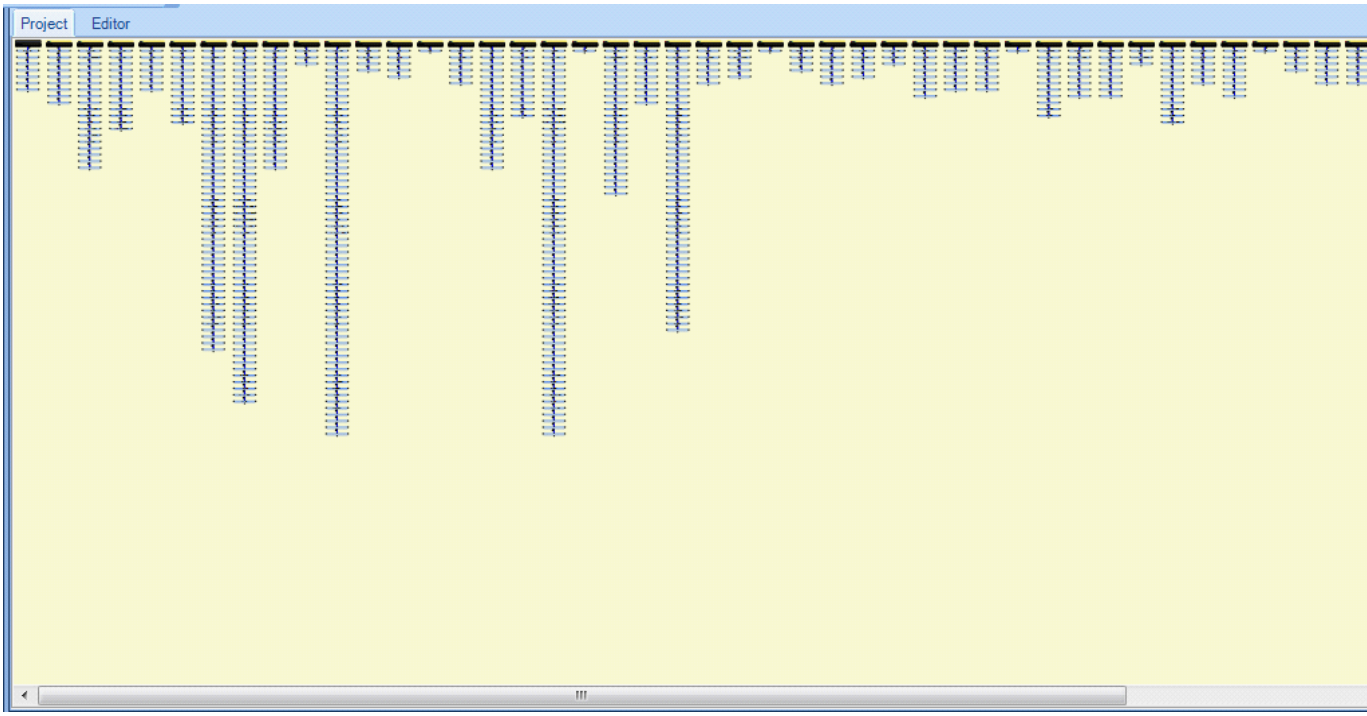
When adequate resources exist the import process will begin, first by assigning symbols followed by scanning for all tasks and then building the task tree's.

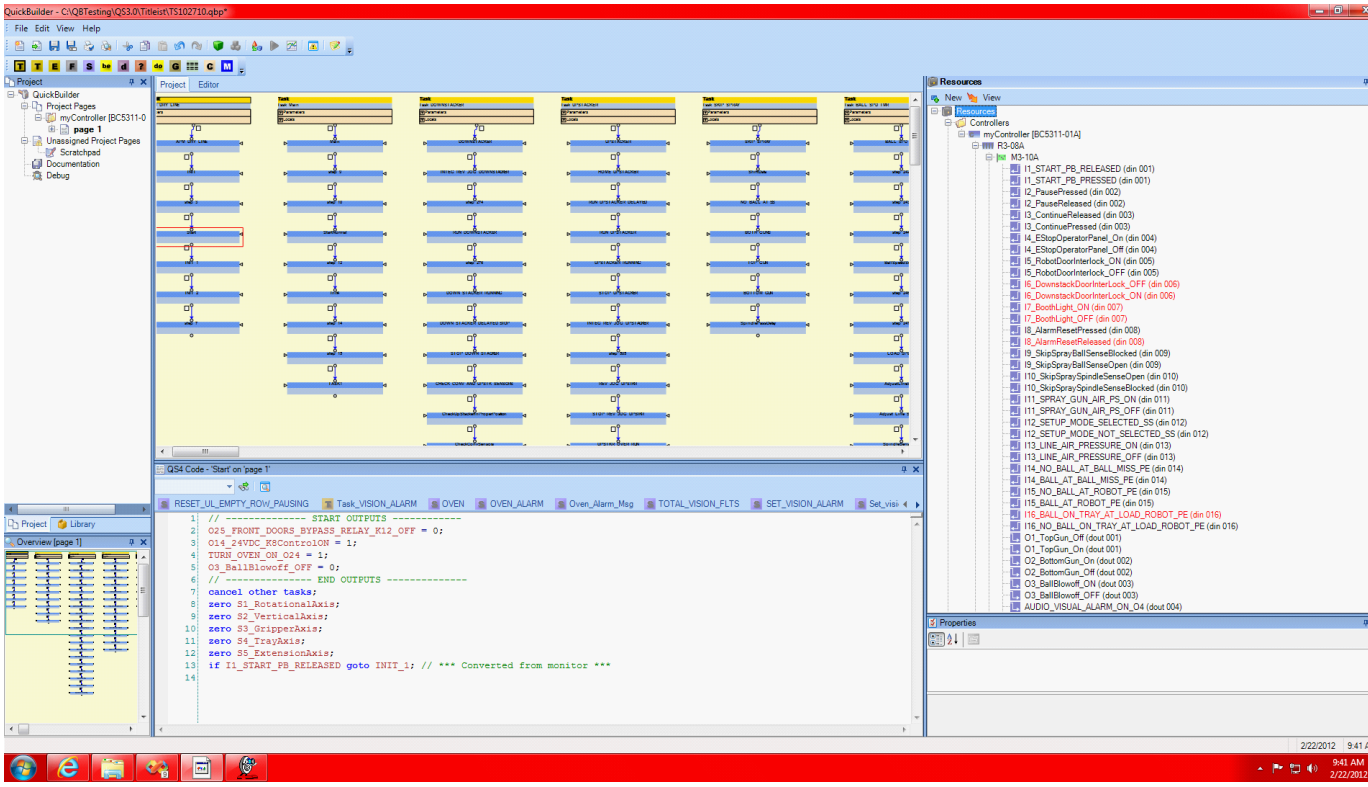


Once complete a dialog box will appear.

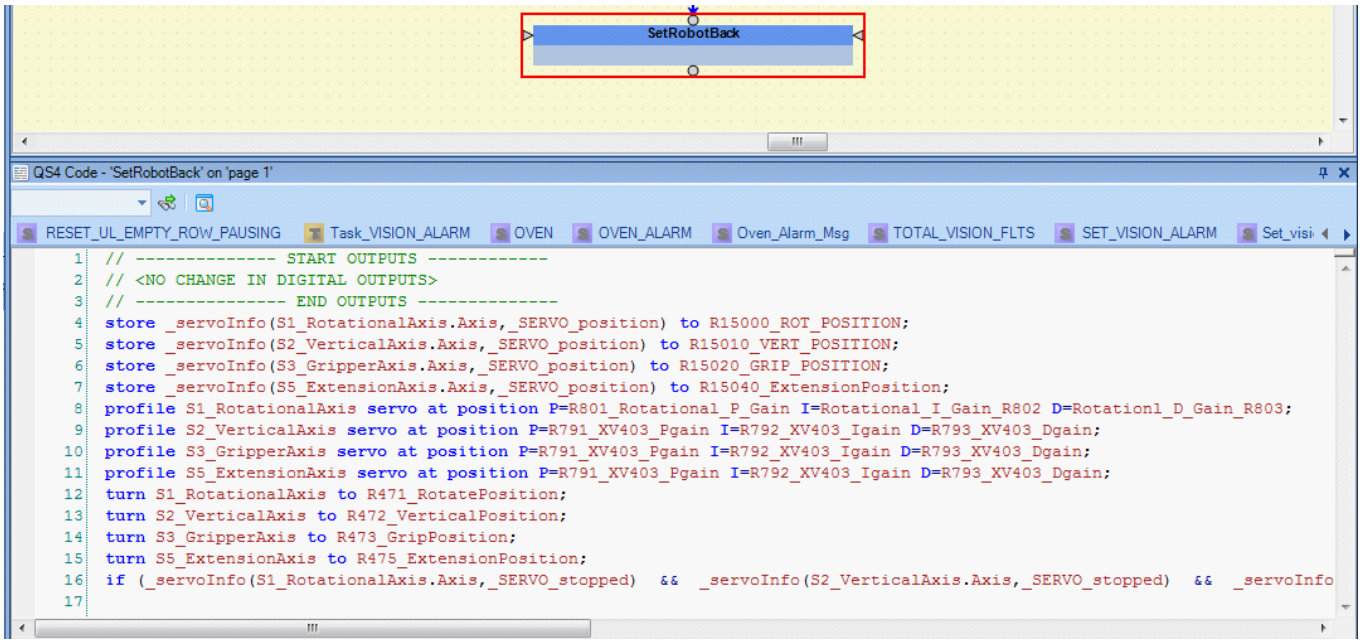


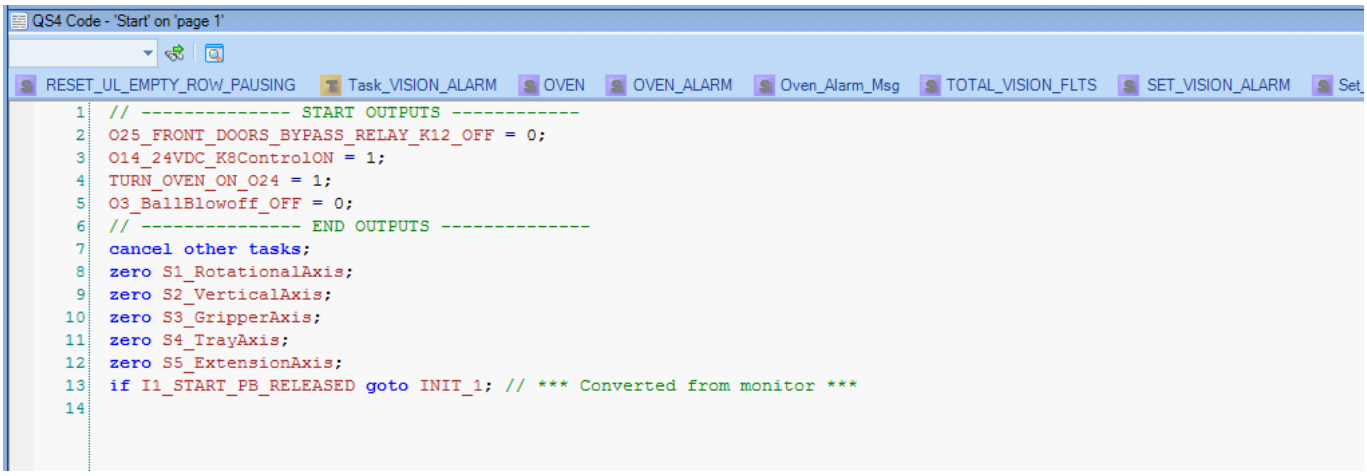
Below is an example of an imported program, each task is a separate tree.



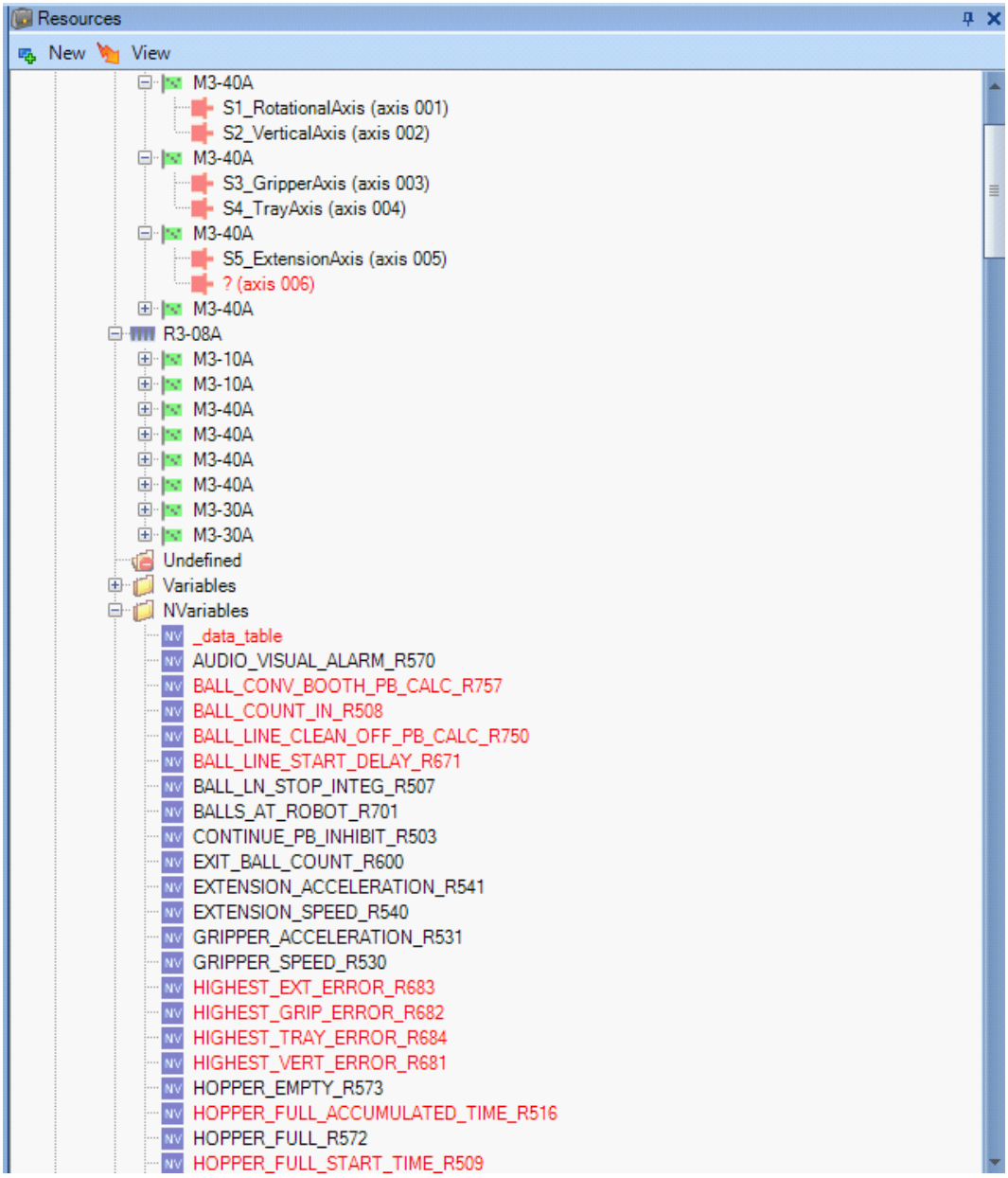

QuickBuilder - C:\QBTesting\QS3.0\Titleist\TS102710.qbp*
File Edit View Help
Project
QuickBuilder
Project Pages
myController [BC5311-0
page 1
Unassigned Project Pages
Scratchpad
Documentation
Debug
Project Editor
QS4 Code - 'Start' on 'page 1'
RESET_UI_EMPTY_ROW_PAUSING
Task_VISION_ALARM
OVEN
OVEN_ALARM
Oven_Alarm_Msg
TOTAL_VISION_FLTS
SET_VISION_ALARM
// -------------- START OUTPUTS ------------
O25_FRONT_DOORS_BYPASS_RELAY_K12_OFF = 0;
O14_24VDC_KSControlON = 1;
TURN_OVEN_ON_O24 = 1;
O3_BallBlowoff_OFF = 0;
// -------------- END OUTPUTS --------------
cancel other tasks;
zero S1_RotationalAxis;
zero S2_VerticalAxis;
zero S3_GripperAxis;
zero S4_TrayAxis;
zero S5_ExtensionAxis;
if I1_START_PB_RELEASED goto INIT_1; // *** Converted from monitor ***
Resources
New View
Resources
Controllers
myController [BC5311-01A]
R3-08A
M3-10A
I1_START_PB_RELEASED (din 001)
I1_START_PB_PRESSED (din 001)
I2_PausePressed (din 002)
I2_PauseReleased (din 002)
I3_ContinueReleased (din 003)
I3_ContinuePressed (din 003)
I4_EStopOperatorPanel_On (din 004)
I4_EStopOperatorPanel_Off (din 004)
I5_RobotDoorInterlock_ON (din 005)
I5_RobotDoorInterlock_OFF (din 005)
I6_DownstackDoorInterLock_OFF (din 006)
I6_DownstackDoorInterLock_ON (din 006)
I7_BoothLight_ON (din 007)
I7_BoothLight_OFF (din 007)
I8_AlarmResetPressed (din 008)
I8_AlarmResetReleased (din 008)
I9_SkipSprayBallSenseBlocked (din 009)
I9_SkipSprayBallSenseOpen (din 009)
I10_SkipSpraySpindleSenseOpen (din 010)
I10_SkipSpraySpindleSenseBlocked (din 010)
I11_SPRAY_GUN_AIR_PS_ON (din 011)
I11_SPRAY_GUN_AIR_PS_OFF (din 011)
I12_SETUP_MODE_SELECTED_SS (din 012)
I12_SETUP_MODE_NOT_SELECTED_SS (din 012)
I13_LINE_AIR_PRESSURE_ON (din 013)
I13_LINE_AIR_PRESSURE_OFF (din 013)
I14_NO_BALL_AT_BALL_MISS_PE (din 014)
I14_BALL_AT_BALL_MISS_PE (din 014)
I15_NO_BALL_AT_ROBOT_PE (din 015)
I15_BALL_AT_ROBOT_PE (din 015)
I16_BALL_ON_TRAY_AT_LOAD_ROBOT_PE (din 016)
I16_NO_BALL_ON_TRAY_AT_LOAD_ROBOT_PE (din 016)
O1_TopGun_Off (dout 001)
O1_TopGun_On (dout 001)
O2_BottomGun_On (dout 002)
O2_BottomGun_Off (dout 002)
O3_BallBlowoff_ON (dout 003)
O3_BallBlowoff_OFF (dout 003)
AUDIO_VISUAL_ALARM_ON_O4 (dout 004)
Properties
2/22/2012
9:41 AM

Sample Quickstep 2/3 motion instructions imported and converted to the QuickBuilder environment:





Sample resources generated after input:



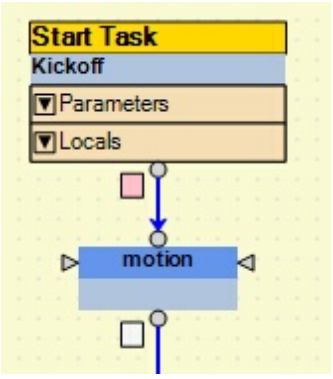
Upon translation warnings may occur when a task branches into the code of another task. This is typical for a Quickstep program, although not good programming practice, hence the warning. It can be ignored.



Severity	Err#	Message	Component
warning	100	Task_APM_DRY_LINE.INIT_2 branching to another task step: Task_Main.Init4	[page 1].Task_APM_DRY_LINE.INIT_2
warning	100	Task_APM_DRY_LINE.step_7 branching to another task step: Task_Main.Main	[page 1].Task_APM_DRY_LINE.step_7
warning	100	Task_Main.step_14 branching to another task step: Task_APM_DRY_LINE.INIT_1	[page 1].Task_Main.step_14
warning	100	Task_ROBOT.step_67 branching to another task step: Task_APM_DRY_LINE.INIT	[page 1].Task_ROBOT.step_67
warning	100	Task_PANEL.AUTO_RESTART branching to another task step: Task_APM_DRY_LINE.INI...	[page 1].Task_PANEL.AUTO_RESTART
warning	100	Task_PANEL.ClearServos branching to another task step: Task_APM_DRY_LINE.INIT	[page 1].Task_PANEL.ClearServos
warning	100	Task_PANEL.RESTART branching to another task step: Task_APM_DRY_LINE.Start	[page 1].Task_PANEL.RESTART
warning	100	Task_TRAJ_ENABLE_DLY.SINGLE_AXIS_PAUSE branching to another task step: Task_PA...	[page 1].Task_TRAJ_ENABLE_DLY.SINGLE_AXIS_PAUSE
warning	100	Task_TRAJ_ENABLE_DLY.SINGLE_AXIS_PAUSE branching to another task step: Task_PA...	[page 1].Task_TRAJ_ENABLE_DLY.SINGLE_AXIS_PAUSE
warning	100	Task_TRAJ_ENABLE_DLY.TrayAdvanceShutDown branching to another task step: Task_...	[page 1].Task_TRAJ_ENABLE_DLY.TrayAdvanceShutDown
warning	100	Task_TRAY_CONV.TRAY_HOMING branching to another task step: Task_ROBOT.DriveEr...	[page 1].Task_TRAY_CONV.TRAY_HOMING
warning	100	Task_TRAY_CONV.TRAY_HOMING branching to another task step: Task_ROBOT.DriveEr...	[page 1].Task_TRAY_CONV.TRAY_HOMING
warning	100	Task_TRAY_CONV.RowNotClear branching to another task step: Task_PANEL.EM_STOP	[page 1].Task_TRAY_CONV.RowNotClear
warning	100	Task_Events.DELAY_ELEVATOR branching to another task step: Task_Elevator1.Elevator1	[page 1].Task_Events.DELAY_ELEVATOR
warning	100	Task_BALL_MISS_CHK.CheckifTrackStillRunning branching to another task step: Task_R...	[page 1].Task_BALL_MISS_CHK.CheckifTrackStillRunning
warning	100	Task_FOLLOWING_ERROR.ServoError branching to another task step: Task_PANEL.EM...	[page 1].Task_FOLLOWING_ERROR.ServoError
info	100	S1_RotationalAxis (axis 001): 0 out of 48 max variables used	
info	100	S2_VerticalAxis (axis 002): 0 out of 48 max variables used	
info	100	S3_GripperAxis (axis 003): 0 out of 48 max variables used	
info	100	S4_TrayAxis (axis 004): 0 out of 48 max variables used	
info	100	S5_ExtensionAxis (axis 005): 0 out of 48 max variables used	
info	0	Image size: 1975060 bytes. 15.37% Program Area Used.	Project
info	0	Translation Time (NO ERRORS): 40.36 sec(s)	Project
info	0	myController archive size: 564,280 byte(s)	Project

⚠ Important: When doing motion control you must tune your motor and set the properties for each axis prior to operation (M3-40 axis). Also make sure you set the driveenable property to the proper output to enable the drive, it is disabled, 0, by default. CTC demo boxes use output #1, thus a change is needed.

⚠ Translation Errors - In looking for step names that may be using a reserved word, as its name, typically the link prior to that having a reserved word will be pink versus white. For example 'motion' is a reserved word, below shows the error:



To correct this problem the step named 'motion' would have to be edited, for example changed to 'motionx', as well as any references. To resolve the pink box and clear that error simple double click on the box and the parser will

then see the step name has been changed and clear the error. Sometimes you need to click on the clear box below the step as well should a translation error continue to occur.

5 Chapter 5: BACnet/IP

BACnet (Building Automation and Controls network) was developed by the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE, www.bacnet.org). BACnet is an ISO global standard, American national standard, a European pre-standard, and is used in more than 30 countries.

BACnet is a data communication protocol, or set of communication rules, that ASHRAE created in order to standardize communication between building automation system components. It allows systems from various vendors, such as HVAC, lighting, security and fire systems, to communicate with each other by providing standardized methods for presenting, requesting, interpreting, and transporting information.

QuickBuilder has a powerful BACnet programming implementation. Almost all devices, objects, and properties can be accessed programmatically, in real time. It is expected that the basic programming skills of QuickBuilder are understood before proceeding with this section. Also the basics of BACnet Devices and Objects is needed. Reference document 951-536105, "BACnet Communications Guide" for more detailed QuickBuilder and 5300 controller information. BACnet is not presently supported within the Incentive embedded PC platform.

QuickBuilder allows a programmer to take a standard volatile variant table and remap it to BACnet device objects. Object values can be read and written, in real time, as the instruction is executed. No values are cached and the operation is initiated immediately. Since QuickBuilder programs run as independent threads this does not affect other task operation. If a polled mode is desired then it is recommended a programmer dedicate a single task to periodically reading BACnet data and storing it into local QuickBuilder variables for other tasks to access, this may be more efficient than simply reading an online variable randomly.

As with reads, writes also occur immediately and as soon as control is returned to the task, the write has occurred. Last read and write error status is also available as well as online system status to ensure data integrity. The 5300 ‘tsm timeout’ and ‘tsm retries’ setting will determine the maximum amount of time a task will hang should the device be powered off and an attempt to access is made. Note that if the device has already been identified as offline then control will return immediately, with the appropriate error code set. It is only during the transition to offline that a delay may occur. Actual read and write times will vary, dependent upon the response time of the device you are conversing with. The 5300 typically initiates the transaction within 1-2 mS.

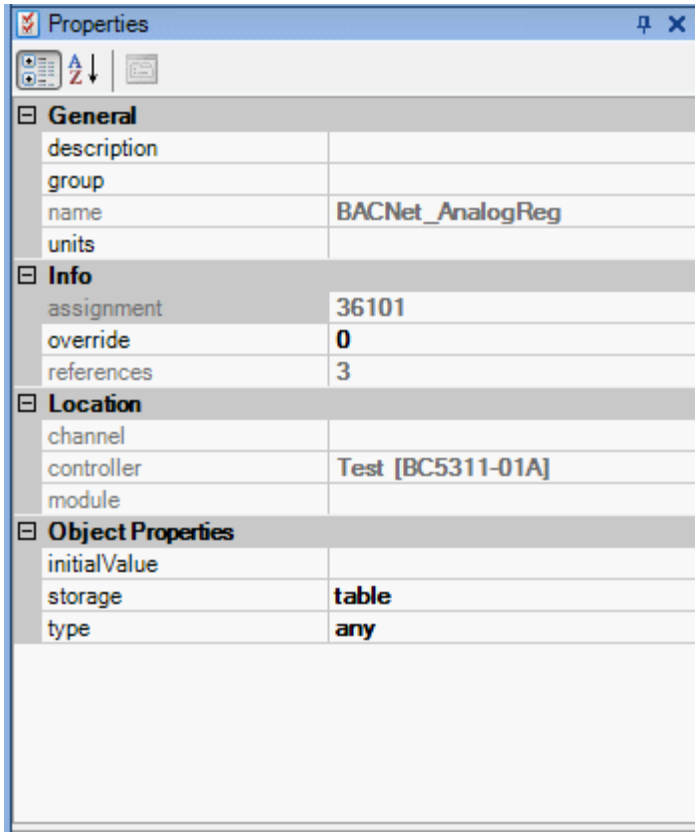


The QuickBuilder/5300 interface does not support properties that have an array. Typically only the first element will appear in a read but given the vast number of properties testing of each desired would be needed.

5.1 BACnet Volatile Tables

In order to define a QuickBuilder volatile variable to be a BACnet aware a number of things must be done. In the example below we are defining a variable ‘BACNet_AnalogReg’ to be redirected to the BACnet network. It will interact with a device called “New Virtual Device 1” and an analog output called “My New Object”. In this particular example the SCADA Engines BACnet Simulator is being used, converting a PC into a BACnet/IP server.

1. Define a variable of storage ‘table’ and type ‘any’:



2. In your program initialize the first element of the table prior to access (only do this once):

```
// Initialize the variant register we will use for BacNet, this just
// creates it by writing any value to the first cell.
BACNet_AnalogReg.i[0][0] = 0;
```

3. Set the BACnet indirection flag (only do this once):

```
// Set our Variant selection register so we can indirectly set the BACNET
remote flag
$REGISTERS[36804] = addr(BACNet_AnalogReg);
// Set the remote access flag, we are now a BACnet remote variable
$REGISTERS[36814] = 1;
```

4. Initialize the table entries for the Device to communicate with, Object Name, and Object Type. It is assumed that the Device name is unique on the network and the Object Name is unique for the Object Type defined. If not the BACnet Device Instance and Object Instance information will need to also be set (Refer to the QuickBuilder Explorer for the needed information). Each row specifies a specific device while the column is the property to be accessed for that device. The variable may reference as many different BACnet objects as desired by simply changing the row index. You may also define multiple tables of different QuickBuilder names referencing the same device and the same or different BACnet objects.

```
// Set the device entry for this array element, multiple device definitions may
be defined
// by incrementing the row value of the table. Below we are assuming the first
table entry.
```

```
// 'device' is an integer.
device = 0;

// Initialize the Device Network Name, each row can be a new or the same device
to access,
// regardless, each row must be initialized to create a mapping.
BACNet_AnalogReg.s[device][$PROP_BACNET_CTC_DEVICE_NAME] = string("New Virtual
Device 1");

// Set the object name we map to
BACNet_AnalogReg.s[device][$PROP_BACNET_CTC_OBJECT_NAME] = string("My New
Object");

// Set the object type expected, OBJECT_ANALOG_OUTPUT
BACNet_AnalogReg.i[device][$PROP_BACNET_CTC_OBJECT_TYPE] =
$OBJECT_BACNET_ANALOG_OUTPUT;

// If duplicate names exist on the network then the BACnet defined device
instance must be
// set, the default is -1, which means the first found matching the device
name.
BACNet_AnalogReg.i[device][$PROP_BACNET_CTC_DEVICE_INSTANCE] = 1;

// If duplicate object names exist for the desired object type then the object
instance must
// be set, the default is -1, which means the first found matching the object
name/type.
BACNet_AnalogReg.i[device][$PROP_BACNET_CTC_OBJECT_INSTANCE] = 0;
```

5. Duplicate the above for all BACnet objects desired, then prior to accessing wait for the device to be online. This is only needed for the first variable to be used accessing that particular device:

```
// Check the system status for this device
online_status = BACNet_AnalogReg.i[device][$PROP_BACNET_CTC_SYSTEM_STATUS];
// If 1 then am online, if 0 then offline
if (online_status == 1) then goto online;
// Wait for bit, then check again
delay 1000 ms;
goto wait_online;
```

6. Once online you may freely access the variable and properties as desired. Since the variant is of type 'any' you must use the .f32 for a float, .i for an integer or enumerated BACnet type, or a .s for a string. The type will automatically be changed to the proper type of the property you are accessing, if possible.

```
// It is online so loop, writing a value, use .f32 since an analog output is of
type Real.
BACNet_AnalogReg.f32[device][$PROP_BACNET_PRESENT_VALUE] = test_value;

// Verify wrote properly and read the last write error, will be 0 if no error,
-1 if busy,
// else error code.  Since we block during write we do not have to loop for
results
result_code = BACNet_AnalogReg.i[device][$PROP_BACNET_CTC_LAST_WRITE_ERROR];

// See if OK
if (result_code != 0) then goto error_handler;

// Wrote properly so continue
```

```
test_value = test_value + 1.0;

// continue to loop
goto online;
```

5.2 BACnet System Variables

BACnet Property Constants

```
PROP_BACNET_DEVICE_NAME =                 0
PROP_BACNET_ALARM_VALUE =                 6
PROP_BACNET_APDU_TIMEOUT =                11
PROP_BACNET_BIAS =                        14
PROP_BACNET_COV_INCREMENT =               22
PROP_BACNET_DEADBAND =                    25
PROP_BACNET_DERIVATIVE_CONSTANT =         26
PROP_BACNET_DESCRIPTION =                 28
PROP_BACNET_ERROR_LIMIT =                 34
PROP_BACNET_FEEDBACK_VALUE =              40
PROP_BACNET_HIGH_LIMIT =                  45
PROP_BACNET_INTEGRAL_CONSTANT =           49
PROP_BACNET_LIMIT_ENABLE =                52
PROP_BACNET_LOCAL_DATE =                  56
PROP_BACNET_LOCAL_TIME =                  57
PROP_BACNET_LOCATION =                    58
PROP_BACNET_LOW_LIMIT =                   59
PROP_BACNET_MAXIMUM_OUTPUT =              61
PROP_BACNET_MINIMUM_OFF_TIME =            66
PROP_BACNET__MINIMUM_ON_TIME =            67
PROP_BACNET_MINIMUM_OUTPUT =              68
PROP_BACNET_MIN_PRES_VALUE =              69
PROP_BACNET_MODEL_NAME =                  70
PROP_BACNET_OBJECT_IDENTIFIER =           75
PROP_BACNET_OBJECT_NAME =                 77
PROP_BACNET_OBJECT_PROPERTY_REFERENCE =   78
PROP_BACNET_OBJECT_TYPE =                 79
PROP_BACNET_OUT_OF_SERVICE =              81
PROP_BACNET_OUTPUT_UNITS =                82
PROP_BACNET_POLARITY =                    84
PROP_BACNET_PRESENT_VALUE =               85
PROP_BACNET_PRIORITY =                    86
PROP_BACNET_PROGRAM_STATE =               92
PROP_BACNET_PROPORTIONAL_CONSTANT =       93
PROP_BACNET_RELIABILITY =                 103
PROP_BACNET_RELINQUISH_DEFAULT =          104
PROP_BACNET_RESOLUTION =                  106
PROP_BACNET_SETPOINT =                    108
PROP_BACNET_SETPOINT_REFERENCE =          109
PROP_BACNET_STATUS_FLAGS =                111
PROP_BACNET_SYSTEM_STATUS =               112
PROP_BACNET_TIME_DELAY =                  113
PROP_BACNET_UNITS"] =                     117
```

```
PROP_BACNET_UPDATE_INTERVAL"] =          118
PROP_BACNET_AVERAGE_VALUE"] =            125
PROP_BACNET_ENABLE"] =                   133
PROP_BACNET_MAXIMUM_VALUE"] =            135
PROP_BACNET_MINIMUM_VALUE"] =            136
PROP_BACNET_VARIANCE_VALUE"] =           151
PROP_BACNET_SETTING"] =                  162
PROP_BACNET_SILENCED"] =                 163
PROP_BACNET_TRACKING_VALUE"] =           164
PROP_BACNET_ADJUST_VALUE"] =             176
PROP_BACNET_COUNT =                      177
PROP_BACNET_COUNT_BEFORE_CHANGE =        178
PROP_BACNET_INPUT_REFERENCE =            187
PROP_BACNET_PRESCALE =                   185
PROP_BACNET_PULSE_RATE =                 186
PROP_BACNET_SCALE =                      187
PROP_BACNET_SCALE_FACTOR =               188
PROP_BACNET_VALUE_SET =                  191
PROP_BACNET_LIGHTING_COMMAND =           224
PROP_BACNET_LIGHTING_COMMAND_PRIORITY =  225
PROP_BACNET_ALARM_STATE =                226
PROP_BACNET_DOOR_EXTENDED_PULSE_TIME =   227
PROP_BACNET_DOOR_OPEN_TOO_LONG_TME =     229
PROP_BACNET_DOOR_PULSE_TIME =            230
PROP_BACNET_DOOR_STATUS =                231
PROP_BACNET_DOOR_UNLOCK_DELAY_TIME =     232
PROP_BACNET_LOCK_STATUS =                233
PROP_BACNET_SECURED_STATUS =             235
PROP_BACNET_OFF_DELAY =                  236
PROP_BACNET_ON_DELAY =                   237
PROP_BACNET_POWER =                      238
PROP_BACNET_POWER_ON_VALUE =             239
PROP_BACNET_PROGRESS_VALUE =             240
PROP_BACNET_RAMP_RATE =                  241
PROP_BACNET_STEP_INCREMENT =             242
PROP_BACNET_SYSTEM_FAILURE_VALUE =       243
PROP_BACNET_ABSENTEE_LIMIT =             244
PROP_BACNET_ACCESS_DOORS =               246
```

BACnet CTC Assigned Special Property Constants

```
PROP_BACNET_CTC_DEVICE_NAME =            65000
PROP_BACNET_CTC_OBJECT_NAME =            65001
PROP_BACNET_CTC_OBJECT_TYPE =            65002
PROP_BACNET_CTC_DEVICE_INSTANCE =        65003
PROP_BACNET_CTC_OBJECT_INSTANCE =        65004
PROP_BACNET_CTC_LAST_READ_ERROR =        65005
PROP_BACNET_CTC_LAST_WRITE_ERROR =       65006
PROP_BACNET_CTC_OBJECT_PRIORITY =        65007 (1 to 16
allowed or 0 for default)
PROP_BACNET_CTC_SYSTEM_STATUS =          65008
```

BACnet Object Constants

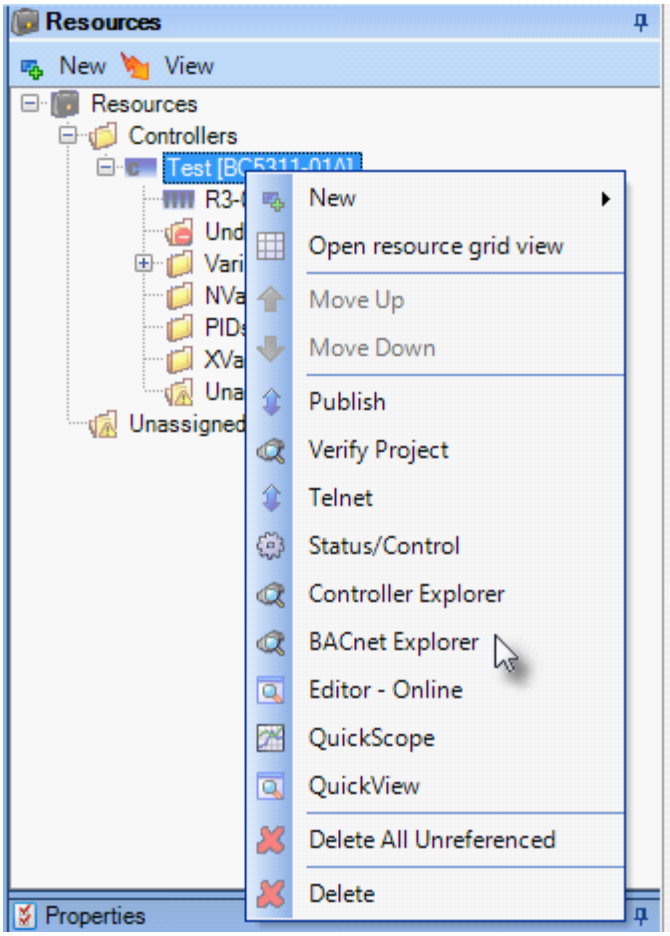
```
OBJECT_BACNET_ANALOG_INPUT =              0
OBJECT_BACNET_ANALOG_OUTPUT =             1
OBJECT_BACNET_ANALOG_VALUE =              2
OBJECT_BACNET_BINARY_INPUT =              3
OBJECT_BACNET_BINARY_OUTPUT =             4
OBJECT_BACNET_BINARY_VALUE =              5
OBJECT_BACNET_CALENDAR =                  6
OBJECT_BACNET_COMMAND =                   7
OBJECT_BACNET_DEVICE =                    8
OBJECT_BACNET_EVENT_ENROLLMENT =          9
OBJECT_BACNET_FILE =                      10
OBJECT_BACNET_GROUP =                     11
OBJECT_BACNET_LOOP =                      12
OBJECT_BACNET_MULTI_STATE_INPUT =         13
OBJECT_BACNET_MULTI_STATE_OUTPUT =        14
OBJECT_BACNET_NOTIFICATION_CLASS =        15
OBJECT_BACNET_PROGRAM =                   16
OBJECT_BACNET_SCHEDULE =                  17
OBJECT_BACNET_AVERAGING =                 18
OBJECT_BACNET_MULTI_STATE_VALUE =         19
OBJECT_BACNET_TRENDLOG"] =                20
OBJECT_BACNET_LIFE_SAFETY_POINT =         21
OBJECT_BACNET_LIFE_SAFETY_ZONE =          22
OBJECT_BACNET_ACCUMULATOR =               23
OBJECT_BACNET_PULSE_CONVERTER =           24
OBJECT_BACNET_EVENT_LOG =                 25
OBJECT_BACNET_GLOBAL_GROUP =              26
OBJECT_BACNET_TREND_LOG_MULTIPLE =        27
OBJECT_BACNET_LOAD_CONTROL =              28
OBJECT_BACNET_STRUCTURED_VIEW =           29
OBJECT_BACNET_ACCESS_DOOR =               30
OBJECT_BACNET_LIGHTING_OUTPUT =           31
OBJECT_BACNET_ACCESS_CREDENTIAL =         32
OBJECT_BACNET_ACCESS_POINT =              33
OBJECT_BACNET_ACCESS_RIGHTS =             34
OBJECT_BACNET_ACCESS_USER =               35
OBJECT_BACNET_ACCESS_ZONE =               36
OBJECT_BACNET_CREDENTIAL_DATA_INPUT =     37
OBJECT_BACNET_NETWORK_SECURITY =          38
OBJECT_BACNET_BITSTRING_VALUE =           39
OBJECT_BACNET_CHARACTERSTRING_VALUE =     40
OBJECT_BACNET_DATE_PATTERN_VALUE =        41
OBJECT_BACNET_DATE_VALUE =                42
OBJECT_BACNET_DATETIME_VALUE =            44
OBJECT_BACNET_INTEGER_VALUE =             45
OBJECT_BACNET_LARGE_ANALOG_VALUE =        46
OBJECT_BACNET_OCTETSTRING_VALUE =         47
OBJECT_BACNET_POSITIVE_INTEGER_VALUE =    48
OBJECT_BACNET_TIME_PATTERN_VALUE =        49
OBJECT_BACNET_TIME_VALUE =                50
```

5.3 BACnet Explorer

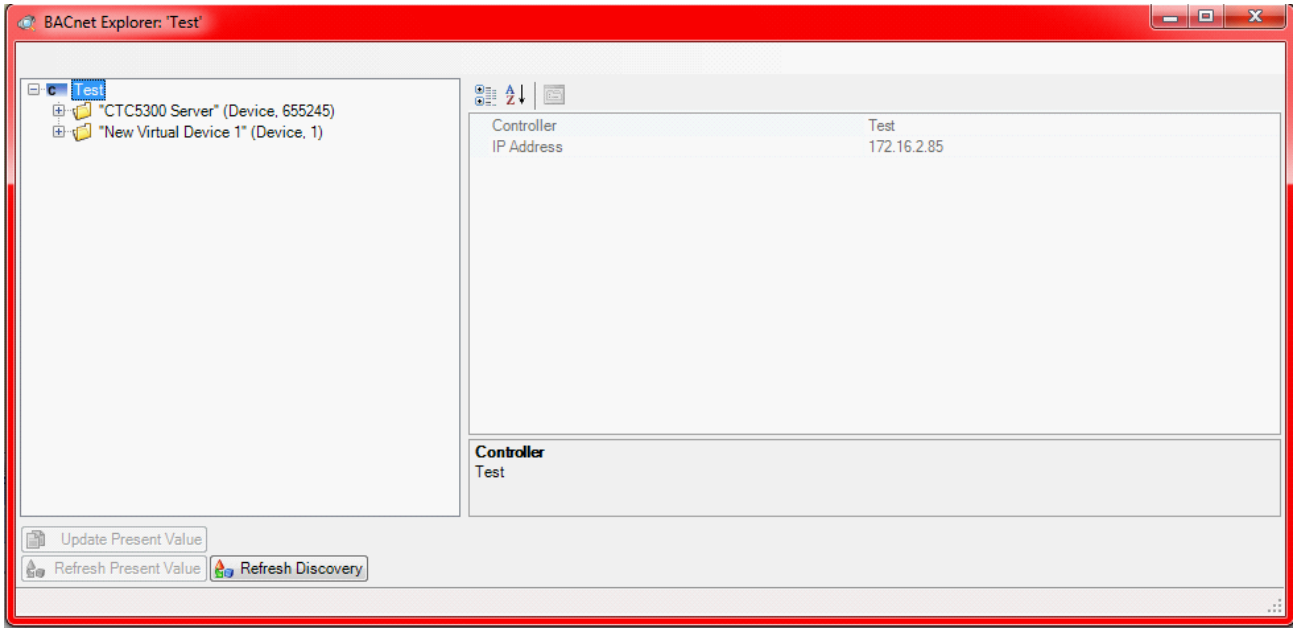
QuickBuilder provides a basic BACnet Explorer whose information comes from the controller your project is defined for. QuickBuilder will initiate a telnet session with the controller and graphically display the information retrieved with the 'get bacnet devices' command. You may then scroll through what that 5300 views as its current BACnet network connections. Present Value properties may be refreshed and updated using graphical buttons which translate to the telnet 'get bacnet data' and 'set bacnet data' commands.

Note that the Present Value properties displayed are those when the device was first on-lined, not at the moment the display was rendered, thus may be old data. Use the 'Refresh Present Value' button to update the selected object.

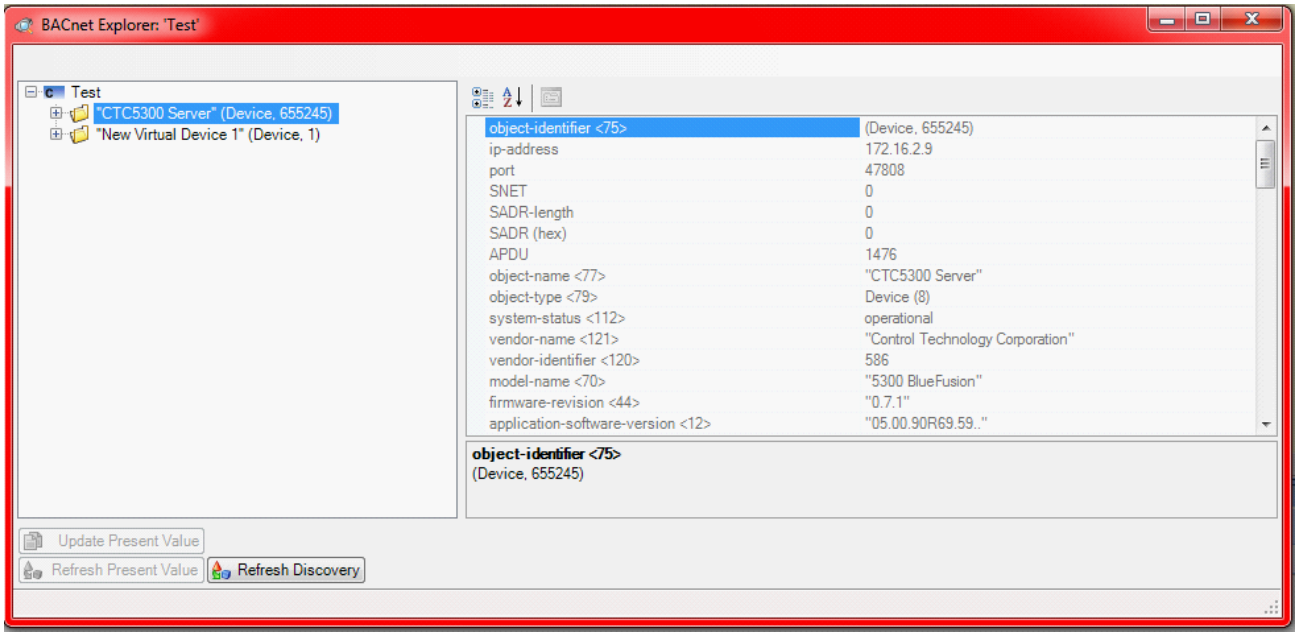
The Explorer is very useful when attempting to determine the names of devices and objects, as well as instance numbers (object-identifier) for inclusion in your QuickBuilder program. To invoke the Explorer, right click the controller of interest and click 'BACnet Explorer':



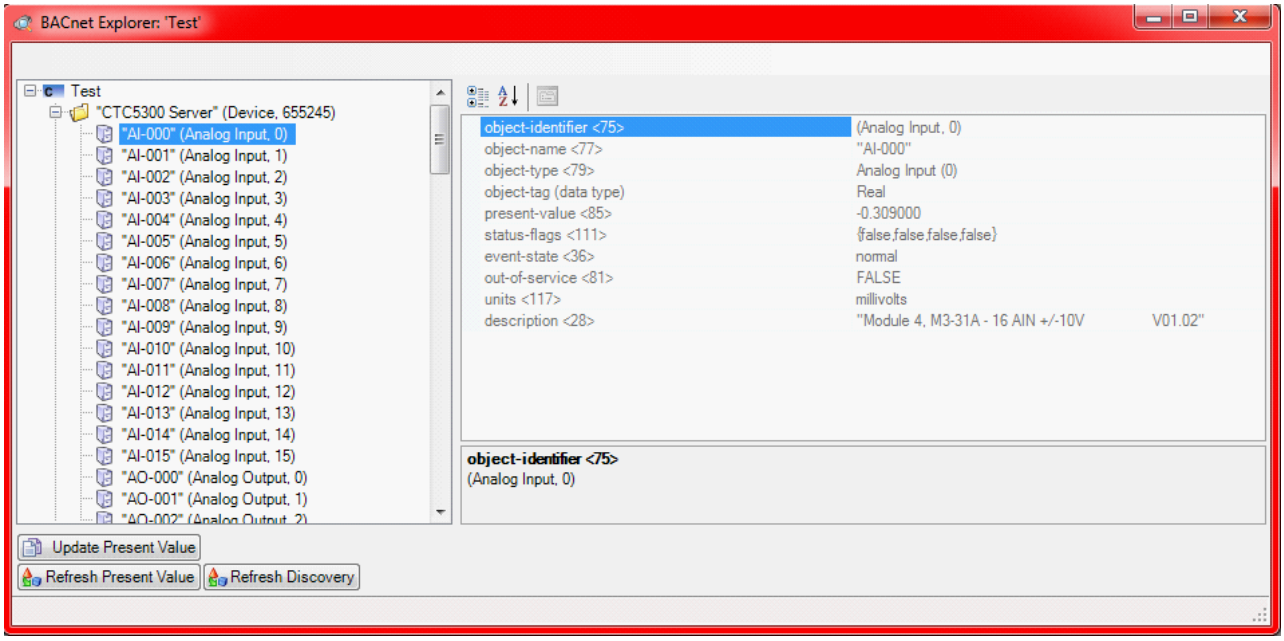
All the devices currently connected will appear after expanding the Controller folder (Test):



Clicking on the device will display the device object information available. The number listed within the <###> is the BACnet property identifier which is used as the QuickBuilder table column reference to read this property.



Upon expanding a device entry, all the public objects will appear:



Once an object is selected you may then use the 'Update Present Value' and 'Refresh Present Value' buttons to modify and refresh respectively. The 'Refresh Discovery' button will initiate a telnet session with the 5300 and update the display with the current discovery information within that controller.

6 Chapter 6: EtherCAT Explorer

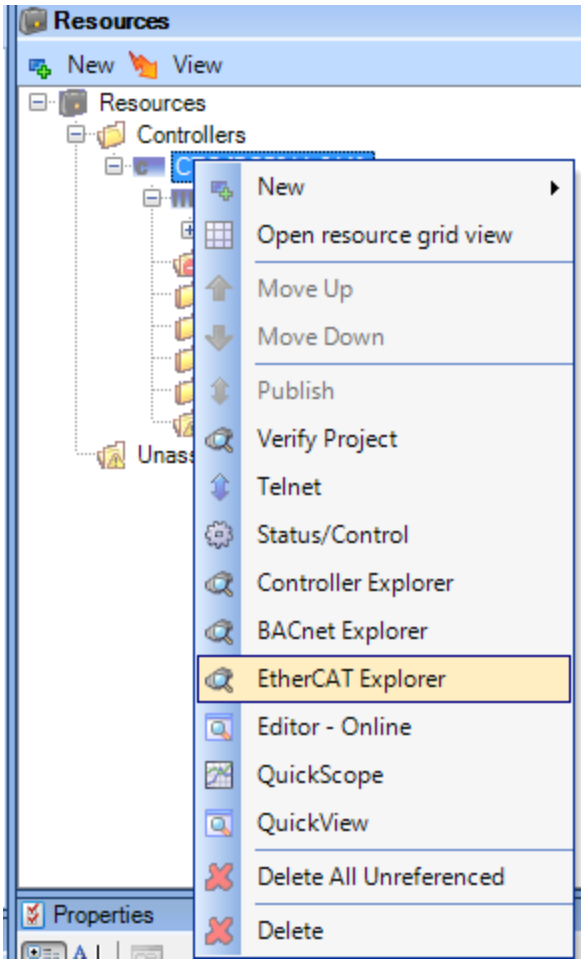
EtherCAT is an open real-time Ethernet network protocol that sets new standards for real-time performance and topology flexibility. It is detailed in the IEC standard IEC61158. Both the 5300 and Windows based IncentivePLC/ECAT offer an advanced EtherCAT Master. The M3-41 EtherCAT Master is available both as a hardware device, module within the 5300 Controller, or as a soft device, executing in real-time on a Windows® based platform (IncentivePLC/IncentiveECAT). In both environments the same application programming environment is used, QuickBuilder. Additionally, a .Net API (IncentiveAPI), exposes the entire MSB language is available on a Windows® platform. The biggest difference between the two is the PC environment is limited to EtherCAT only IO while the embedded 5300 controller has numerous local IO and network possibilities, in addition to EtherCAT. Execution on a Windows based PC presents the developer with an open and diverse architecture with which to implement their automation solution. Both products offer an advanced EtherCAT Master providing distributed motion control, using the CAN application protocol over EtherCAT, supporting such devices as servo drives, RFID readers, and I/O control.

Unlike EtherCAT Masters from other vendors, Incentive attempts to isolate the user from the complexity of the EtherCAT environment by automatically scanning the network and configuring supported devices. The programming interface uses the same high-level language that the previous 5300 M3-40 series of modules uses: QuickBuilder MSBs (Motion Sequence Blocks). You no longer have to deal with a complex configurator, poking drive objects, or figuring out how an interface works. Each of the supported motion and/or I/O devices has been verified, and all setup and initialization is done for you. This greatly simplifies an EtherCAT installation, enabling you to concentrate on motion control and your system, not a complicated configurator. Multiple EtherCAT Master Network modules (M3-41 and PC based) can be intermixed with other networks such as BACnet®, Modbus®, and other modules offered within the embedded 5300 controller family.

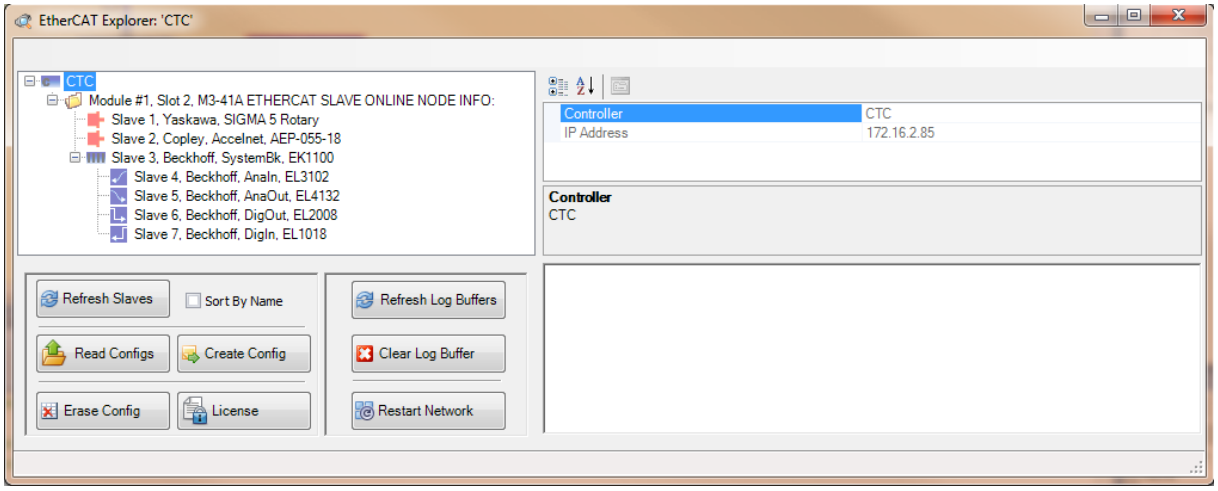
QuickBuilder provides a simple-to-use EtherCAT Explorer. The Explorer communicates directly with the EtherCAT Master and graphically presents the network information. It also provides a high-level diagnostics capability. When using Incentive PC each EtherCAT Master process which is executing appears as a M3-41 module.

6.1 Status Window

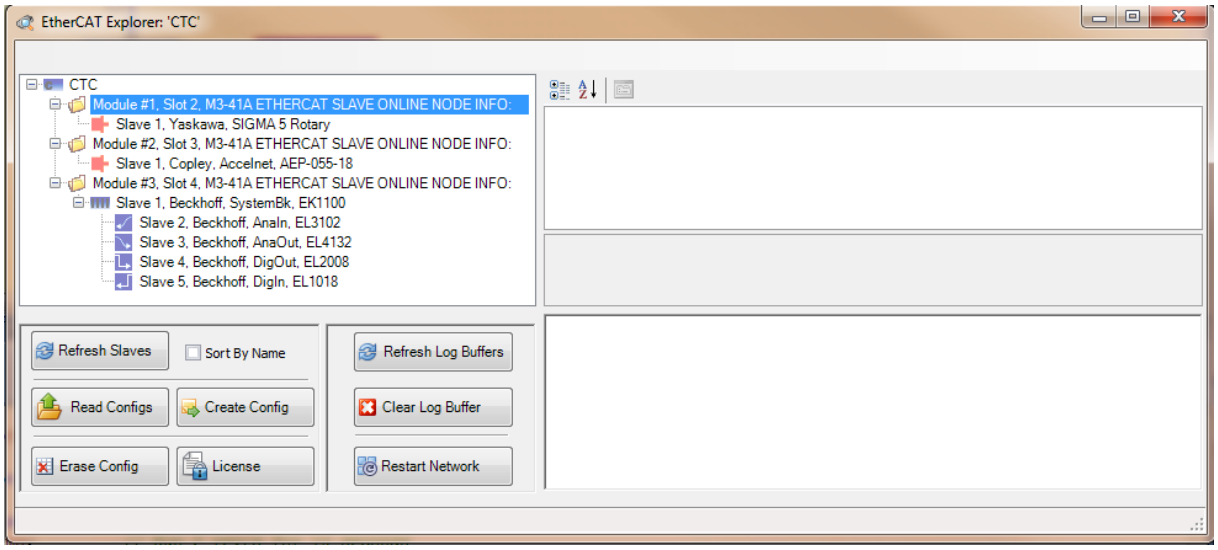
The EtherCAT Explorer window is a feature of the QuickBuilder environment. To open the EtherCAT Explorer, right click on the controller available in Resources. A menu of options will appear; select ‘EtherCAT Explorer’ to connect to the defined controller.



Once invoked, a window similar to the one below will appear, enabling you to monitor the EtherCAT module and its Master configuration.

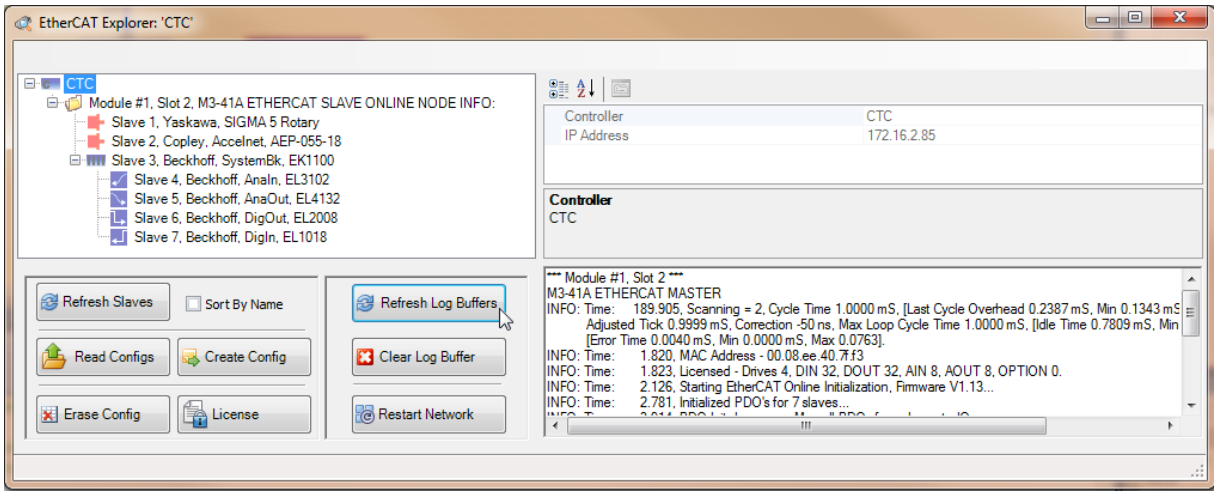


If multiple M3-41 modules (or IncentiveECAT) are present, a folder will appear for each module, with the slaves it controls listed below. Here is an example of three M3-41 EtherCAT Master networks in one model 5300 controller with the first M3-41 module selected (highlighted).



The top left tree is known as the Slave Discovery Window. Both online slaves and the expected slave configurations appear here. Select a slave entry, and the available property information appears within the window to its right.

The window on the bottom right is known as the Message Window. As the EtherCAT Master executes, diagnostic log information is stored in the M3-41 module. By selecting 'Refresh Log Buffer' the most current contents of the log buffer will be displayed. Note that at power up the Licensing information appears in the list, "Licensed: Drives 4…". This is the total number of I/O and drives that your EtherCAT Master is authorized to control.

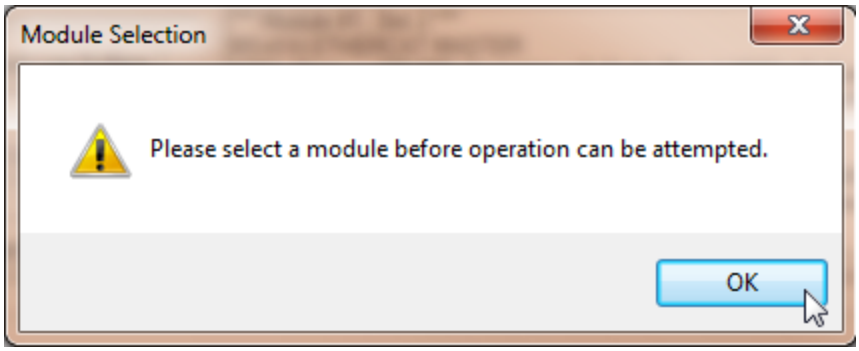


The panel on the lower left contains a number of buttons. Some are for global access; others are for individual M3-41 modules. Those operating an individual EtherCAT Master Module (M3-41) require the Slave Discovery Tree entry with the folder icon (Module #, Slot) to be selected to identify which module is to be accessed. The following operations are available:

Refresh Slaves – Updates the Slave Discovery Window with all online slaves observed at the last restart for all installed M3-41 modules. Each slave's properties are also refreshed to the most current. These properties vary by drive. Drive information will contain present PDO contents, position and state information, etc.

Read Configs – Updates the Slave Discovery Window with any saved configuration file whose content resides in the module's non-volatile storage for all installed M3-41 modules. The information displayed is what is required to be online for the network to become active.

Create Config – This button erases the non-volatile memory stored in the selected module and writes XML information matching the current online slave's to the EtherCAT Master module. This is an alternate approach to using an EtherCAT Configurator, such as Beckhoff's, allowing the configuration to be dynamically created from within QuickBuilder. The actual creation and storage is performed by the EtherCAT Master module, thereby requiring no file transfer. Note that this operation takes about 20 seconds to complete, because of the length of time required to erase non-volatile memory. The appropriate Module # must be selected from within the tree list prior to pressing the **Create Config** button, or an error message will be displayed:



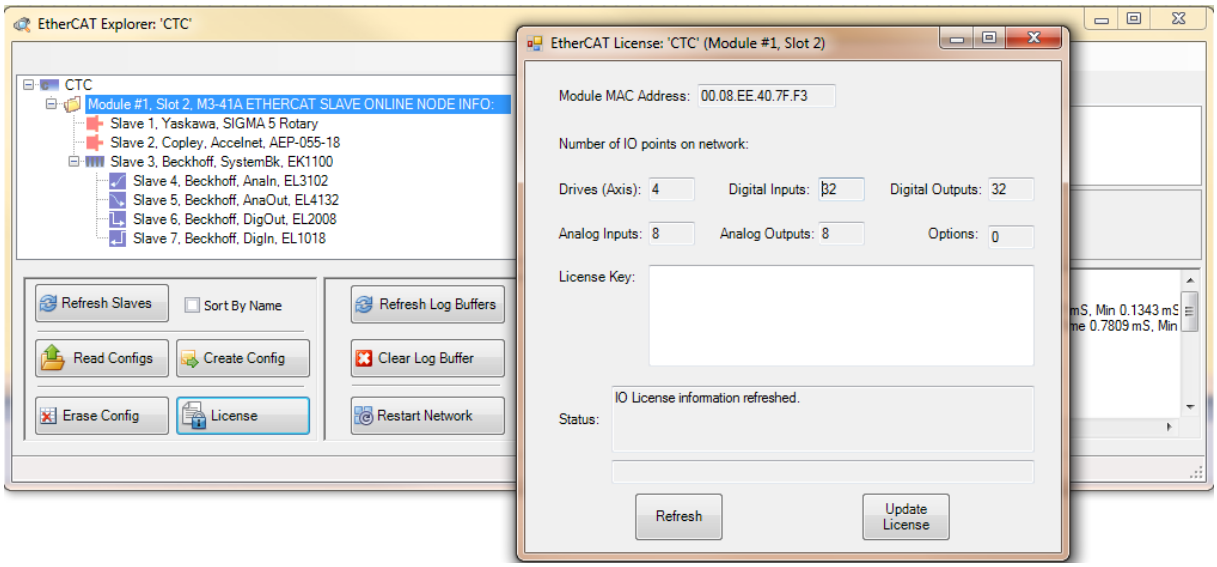
Note that the PC runtime stores this information in a file called _slaveConfig_[MAC Address].txt located in the _system\Programs directory.



Configuration files must be used in a production environment to ensure all the required devices are online prior to executing their controller MSBs. Differing devices power up at different times and may not initially respond to the EtherCAT Master online broadcast. Having a configuration file to compare against informs the master that it must wait for devices to come online prior to proceeding with the boot operation.

Erase Config – This button erases the current configuration file stored in the selected EtherCAT Master Module's non-volatile memory. By default, when no file is stored, and the network is restarted, no verification of online slaves occurs, and the controller begins operation with whatever devices and I/O are found on the network. This is known as Slave Discovery Mode and is useful when initially setting up a network. It can take up to 20 seconds for this command to complete. The appropriate Module # must be selected from within the tree list prior to pressing the **Create Config** button or an error message will be displayed.

License – This button displays the EtherCAT Master License form for the selected module. The MAC Address of the module appears along with the type and number of devices authorized for control by the master. New license keys can be purchased from CTC technical support, and entered within this form to change the current authorization.



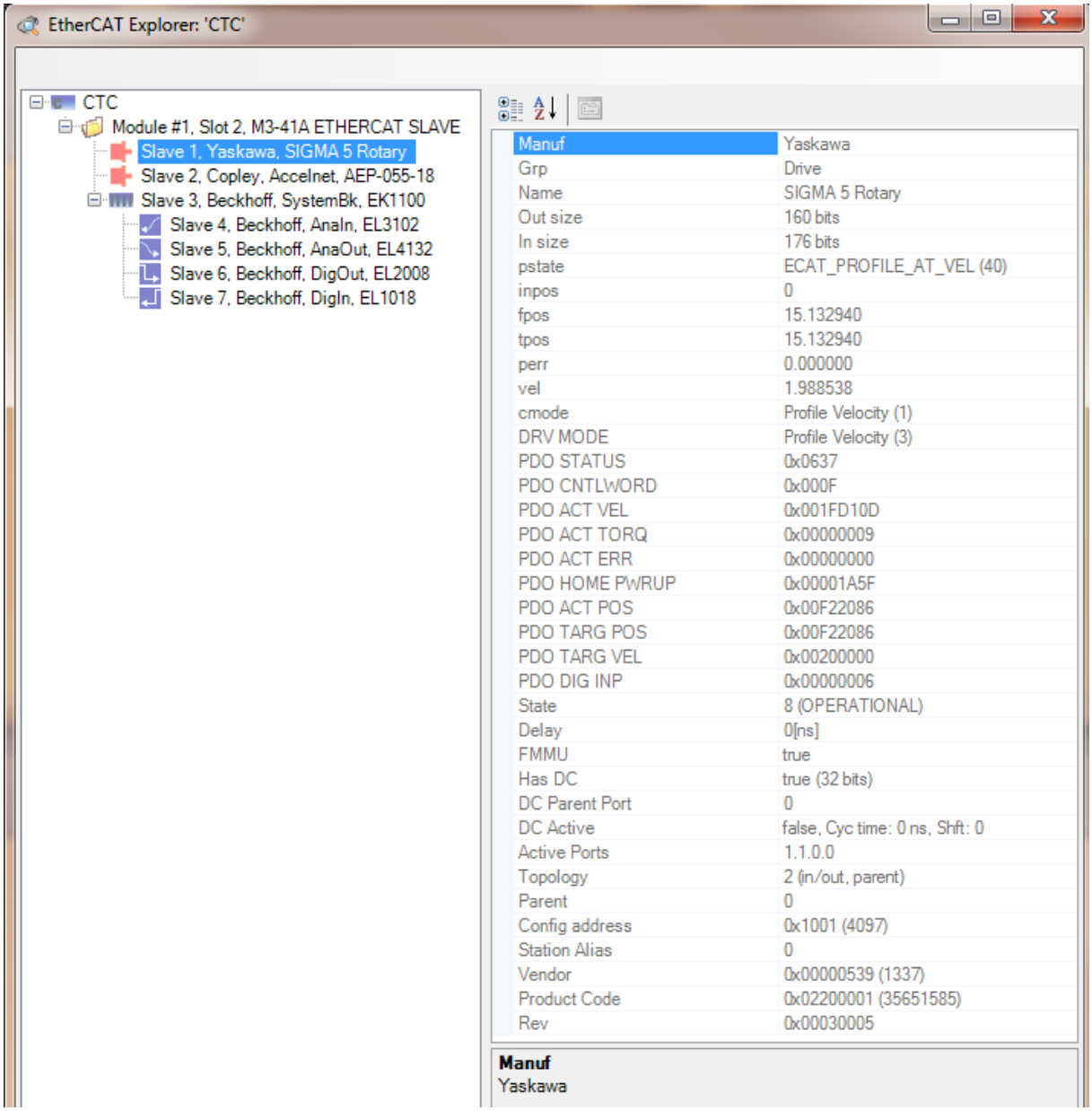
Copy the license key you receive by email and paste it into the 'License Key" text box. Click the **Update License** button to update the number of I/O authorized. Click the **Refresh** button to verify the changes have been made. You must reboot the controller for the changes to take effect at the network level. Note that the PC runtime stores this information in a file called _ioOptions_[MAC Address].txt located in the _system\Programs directory.

Refresh Log Buffers – This button displays log messages residing in all EtherCAT Master Modules in the lower right window. It is useful for diagnostic purposes.
Clear Log Buffer – This button clears the log messages for the selected EtherCAT Master module. Only new messages that occur after the clear operation will appear after the **Refresh Log Buffers** button is selected.
Restart Network – This button causes the selected EtherCAT Master module to re-scan the network and display whatever slaves are found. Prior scan results are overwritten. Available I/O in the controller will not be updated, and a reboot is required if the configuration changes. The network will not be available until after the restart is completed. Restarting the network is useful when connecting new slaves to the network or after power cycling a slave to verify that it is seen on the network. Note that MSBs will also restart.

Reboot Controller – This button causes the controller to be rebooted remotely. This is a hard reset and can take up to 30 seconds before the controller will be back online.

User Options – This button allows customization of the EtherCAT Master parameters, such as PDO cycle time, number of virtual drives, timeouts, and retries. Note that the PC runtime stores this information in a file called _options_[MAC Address].txt located in the _system\Programs directory.

6.2 Properties



Manuf - Manufacturer description

Grp – Group description

Name – Device Name

Out size – Number of bits the device consumes in the output Ethernet packet

In size – Number of bits the device consumes in the input Ethernet packet

Program Variables (MSB variables, if a drive):

pstate – present MSB program operational state
inpos – 1 if motor is in position, 0 if it is not
fpos – Current motor position in revolutions
tpos – Present target position in revolutions
perr – Present error (tpos – fpos)
vel – Present motor velocity in revolutions/second
cmode – MSB program commanded mode for the motor

DRV MODE – CANOpen DS402 mode the drive is in for motion control

PDO STATUS – Object 0x6041 representing the device state

PDO CNTLWORD – Object 0x6040 representing the currently written Control Word

PDO ACT VEL – Object 0x606C representing the current velocity

PDO ACT TORQ – Object 0x6077 representing the current torque. On some drives this is the Actual Current when torque is not available.

PDO ACT ERR – Object 0x60F4 representing the current servo position error

PDO HOME PWRUP – Power up position first seen by the EtherCAT Master

PDO ACT POS – Object 0x6064 representing the actual current position in increments

PDO TARG POS – Object 0x607A representing the target position in increments. This is relevant in Cyclic Sync Position and Profile Position modes. Interpolated motion mode uses 0x60C1 subindex 1.

PDO TARG VEL - Object 0x60FF representing the target velocity in increments/sec. This is only relevant in Profile Velocity mode.

PDO DIG INP – Drive Inputs as reported by the cyclic PDO scan

State – Last seen EtherCAT state of this device

Delay – Propagation delay, in ns, of this device as cabled on the network.

FMMU – True if Fieldbus Memory Management unit is bit oriented, false if it is not.

Has DC – Set to 1 if the drive can support being the source of the distributed clock. The EtherCAT Master selects the first device that 'Has DC' as the source of the clock and then periodically reads the time from that device and writes it to the rest of the slaves. The Master attempts to sync its internal clock to that device as well.

DC Parent Port – The slave named as the Distributed Clock master

DC Active – Information relative to DC Sync, if enabled

Active Ports – Each device can typically have up to 4 ports. This represents the ports being used on this device.

Topology – Each EtherCAT slave has up to 4 internal ports, each represented by a bit.

Parent – Set to the parent node. 0 means it is the parent.

Config address – EtherCAT assigned device address representing its place with regards to being cabled on the network. First device would be 0x1001.

Station Alias – Programmable station alias used to define axis numbers for MSB assignment

Vendor – Device vendor code

Product Code – Device product code

Rev – Firmware revision of the device

6.3 Log Buffer Timings

When the Log Buffer is viewed the first line after the module identification information contains internal timing information. This information can be critical in troubleshooting problems or possibly preventing them. The timing information contains the state of the EtherCAT network scanning, control loop overhead, idle, and slave sync timings. Below was observed in a six axis system do simple back and forth motion on all drives:

*** Module #1, Slot 4 ***
M3-41A ETHERCAT MASTER
INFO: Time: 275.943, Scanning = 2, Cycle 1.0000 mS, [Overhead 0.3180 mS, Min 0.1580 mS, Max 0.3995 mS, Avg 0.2951 mS],
Adjusted Tick 1.0000 mS, Correction -6 ns, Max Tick 1.0000 mS, [Idle 0.6640 mS, Min 0.5234 mS, Max 0.8462].
[Sync Error 0.0002 mS, Min 0.0000 mS, Max 0.0730, Avg 0.0022].

The following is how to interpret these timings:

[Scanning = 2] – The EtherCAT network scanner has three possible states.
- Initializing, 0.
- Scanning but for initial sync, 1.
- Online and executing, 2.

[Cycle 1.0000 mS] – Network control loop scan time, typically 1 mS or 500 µS.

[Overhead] – The time needed to process the PDO packet from the slaves, calculate the new trajectories, update IO, and prepare the PDO packet for transmission. The Cycle time minus the Overhead time is how much time the rest of the system has to execute. The Overhead may never be greater than the Cycle time. If it occurs once in a while and is less than 50 µS a warning will result and recovery attempted, otherwise a fault error. The first time listed is the time for the last completed cycle.

[Adjusted Tick] – The time that the FPGA timer was last set to, on the last cycle. This will shift slightly to sync with the reference slave.

[Correction] – The amount of correction added to the last time cycle in order to more closely sync to the reference slave.

[Max Tick] – The maximum time that the FPGA timer was set to for its periodic interrupt.

[Idle] – The amount of idle time available for the rest of the system to run, with the current time for the last completed cycle listed first.

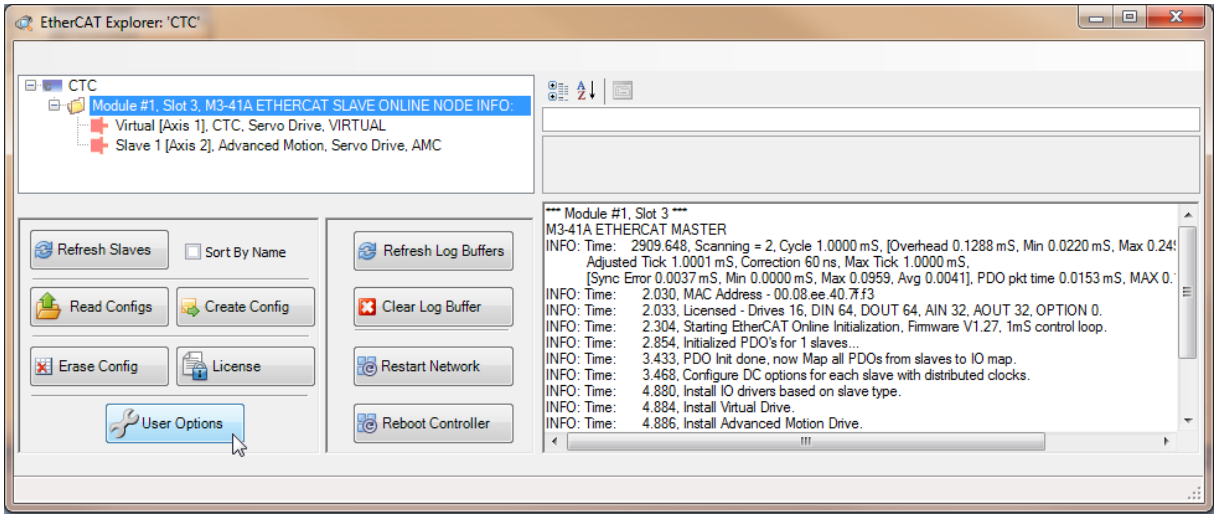
[Sync Error] – This is the amount of error or jitter that the master has experienced while attempting to sync to the slave reference. The first listed time is from the last completed control loop cycle. Note that the average jitter is only 2.2 µS with a maximum of 73 µS. The 73 µS only lasts for a single cycle as corrections are applied.

6.4 User Options

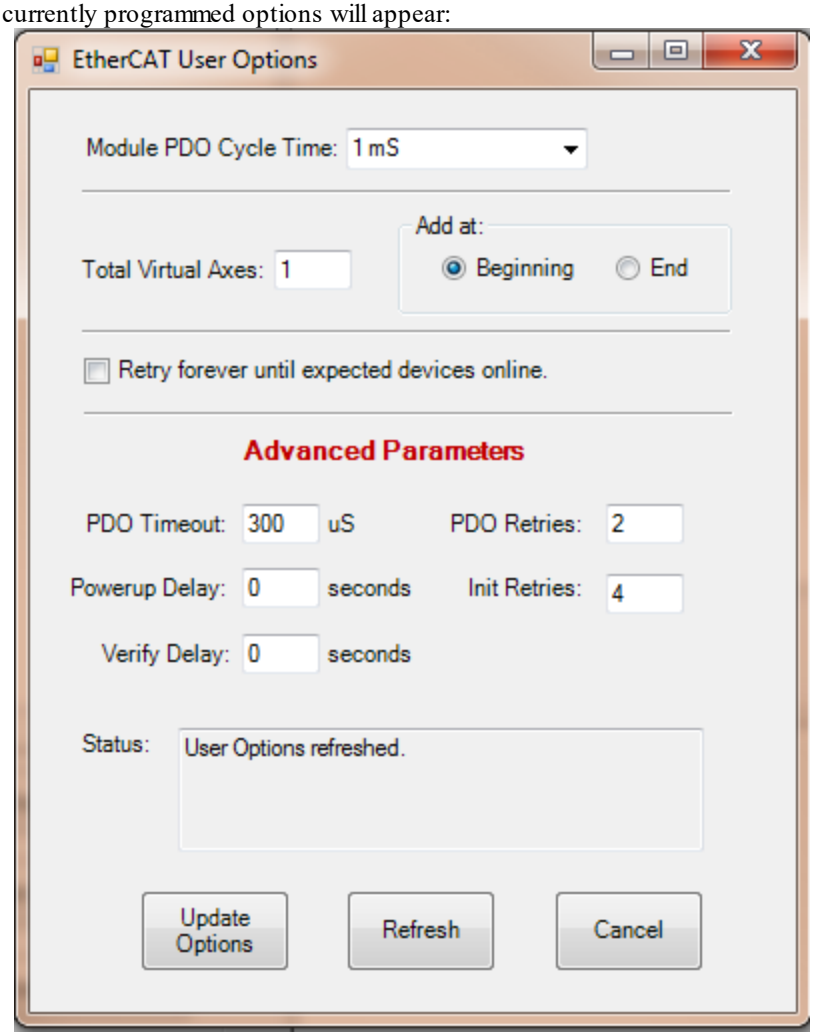
The EtherCAT Explorer allows the user the ability to customize the EtherCAT Master's operation. The customization currently supported consists of:

- Master PDO cycle loop times of 500 µS, 1 mS, 2 mS, or 4 mS.
- Automatic virtual axis creation.
- Capability to add the virtual axis to the end or beginning of those drives online.
- Retry forever option.
- PDO Timeout & retries.
- Initialization retries.

Invoking the 'User Option' form is done by clicking that button within the EtherCAT Explorer:



Once invoked the currently programmed options will appear:



Module PDO Cycle Time – This option sets the EtherCAT master control loop cycle time. The default of 1 mS is typically fine but in some situations the user may wish to speed up or slow down the loop. For example if the system is heavily loaded a 2 or 4 mS control loop will work well with most drives. Selections of 500 µS, 1 mS, 2 mS, and 4 mS are available.

Total Virtual Axes – This option sets the number of virtual axis to 'Add at' the 'Beginning' or 'End' of the online drive list. A virtual axis runs an MSB just like an online axis except that its feedback position (fpos) is updated automatically to its incremental tpos on each control loop, thereby simulating motion. The Virtual Axis is reported to the QuickBuilder as a normal axis.

Retry forever – This option, when selected, will cause the controller and EtherCAT network not to boot until the stored online configuration is observed. If the option is not selected then the 'Init Retries' parameter within the Advanced Parameters will be referenced and that many retries attempted prior to reporting the fault state to the controller and aborting operation.

PDO Timeout – This option should not be set unless instructed by CTC technical support. It will automatically be optimized to the proper setting based upon the PDO Cycle time selected. The option is the amount of time that the EtherCAT Master will wait for the response to the cyclical PDO packet transmission. PDO Timeout X PDO Retries should be less than the cycle time to ensure no DC Sync errors.

PDO Retries – This option should not be set unless instructed by CTC technical support. It will automatically be optimized to the proper setting based upon the PDO Cycle time selected. The option is the number of PDO Timeouts that are allowed before aborting operation and faulting. PDO Timeout X PDO Retries should be less than the cycle time to ensure no DC Sync errors.
Powerup Delay – This option defines how long, in seconds, the M3-41 module should wait, after power up, prior to beginning its identification of network slaves and initializing the EtherCAT network. It is useful when attempting to prevent timeouts on equipment that may take a long time to power up and come online.

Init Retries – This option is the number of times the EtherCAT Master will attempt to activate the network and initialize devices. If a Network Configuration is saved and those devices observed online do not match the network will be re-initialized and scanned again, with this count decremented by 1. Once a count of 0 is reached the module will abort, fault, and report an error.

Verify Delay – This option sets the amount of time, in seconds, the M3-41 module should delay after it identifies all required slave devices online (INIT state) and initializes their PDO's (PRE_OP state). After the delay occurs one more INIT cycle will be done and delay prior to updating the PDO mappings within the slave and marking the devices as online. This is required in some networks where slaves can report that they are online and ready but in fact other equipment is powering up or the slave still needs a small amount of time to continue initialization.

Available buttons:

Update Options – Clicking this button will cause the module to be set to the settings currently displayed. The status window will display the results of the operation.

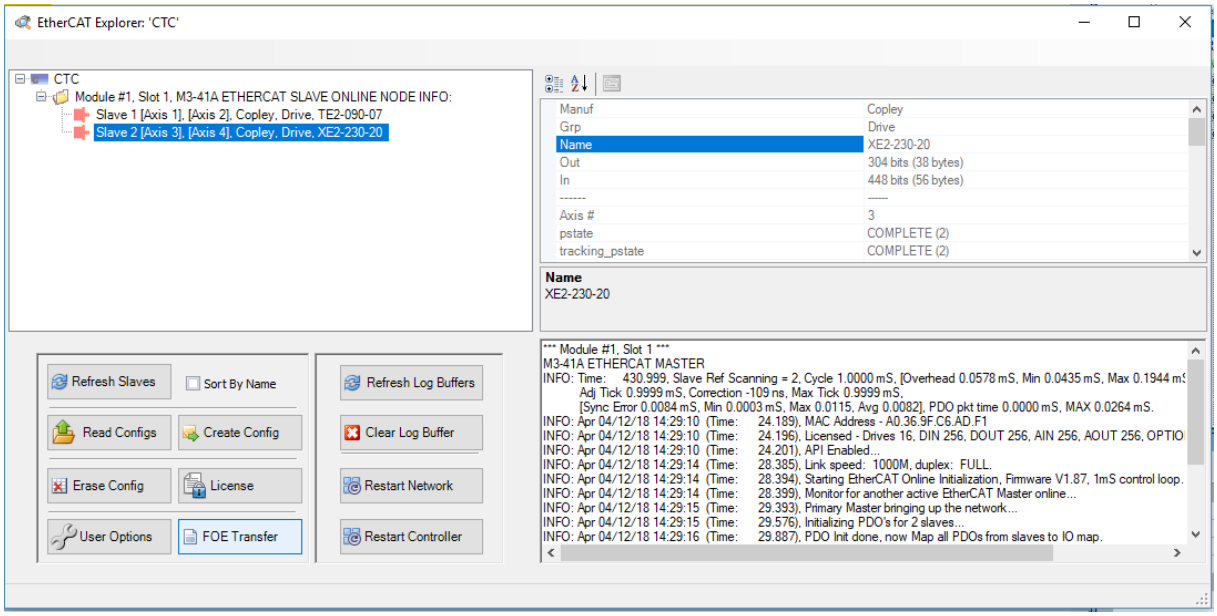
Refresh – Clicking this button will cause the form to be updated with the options currently programmed within the module.

Cancel – Clicking this button will close the form, without changes and return to the EtherCAT Explorer form.

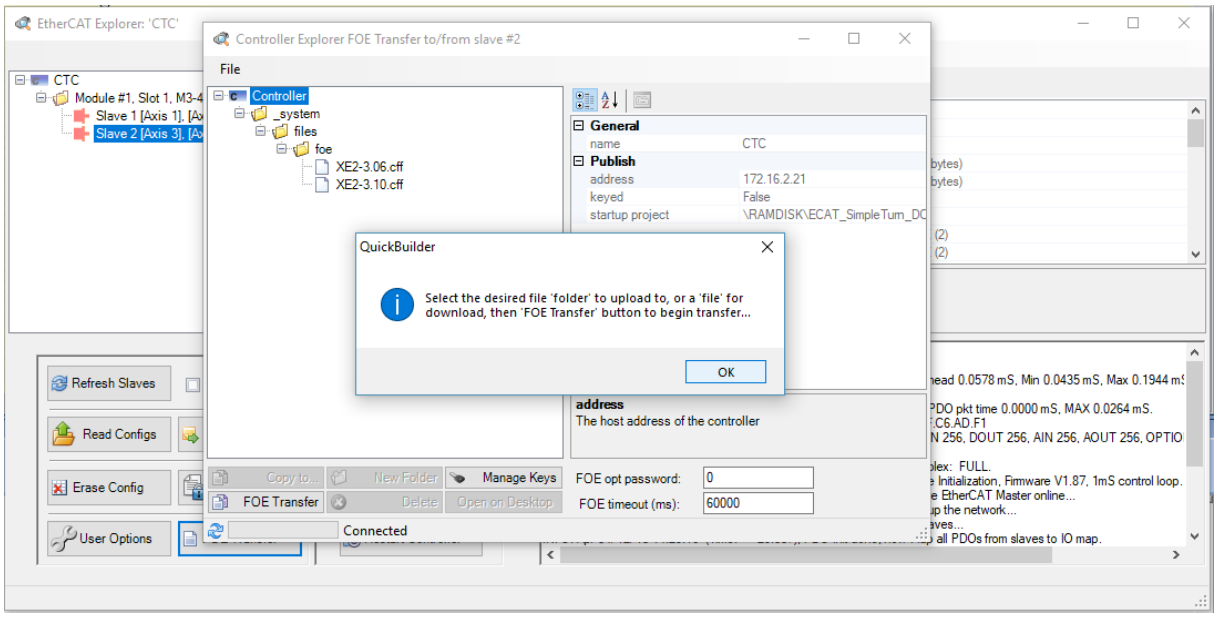
6.5 File Over EtherCAT (FOE)

The EtherCAT Explorer supports an EtherCAT protocol known as FOE or File over EtherCAT. This protocol allows a means to transfer configuration and firmware files to slave devices on the network using EtherCAT. Previously slave devices had to be manually taken offline and serial ports or other network connections used to update manufacturer specific information. Not all manufacturers support this protocol and for the moment it is only available and tested on Copley drives. Mitsubishi also supposedly has support but it has yet to be tested and requires special firmware in their drives.

Only Incentive PC supports the protocol. It could be made available on the 5300 PLC but given its limited storage abilities it is yet to be fully implemented. If the need arises it could be easily added. To invoke the FOE transfer window you must first select the slave device of interested followed by pressing the 'FOE Transfer' button:

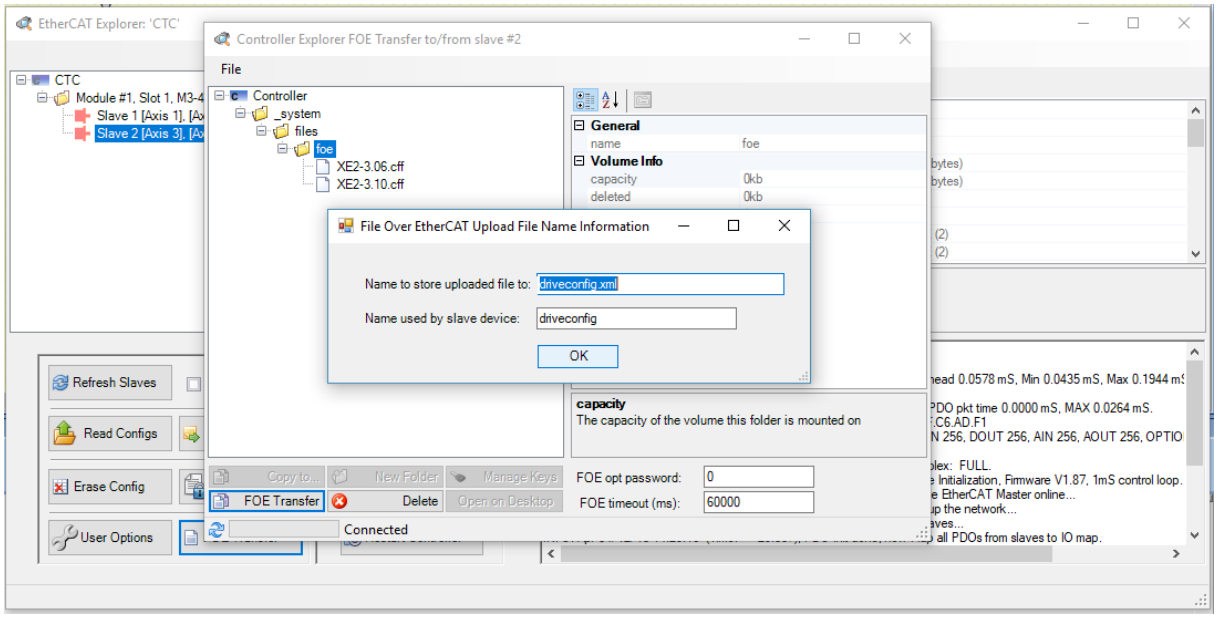


The Controller Explorer window will then appear showing the default FOE storage directory C:_system\files\foe. This, and additional subfolders is where files for transfer should be placed on the Incentive PC. The Controller Explorer uses the FTP protocol to obtain a directory of the remote PC for your presentation:



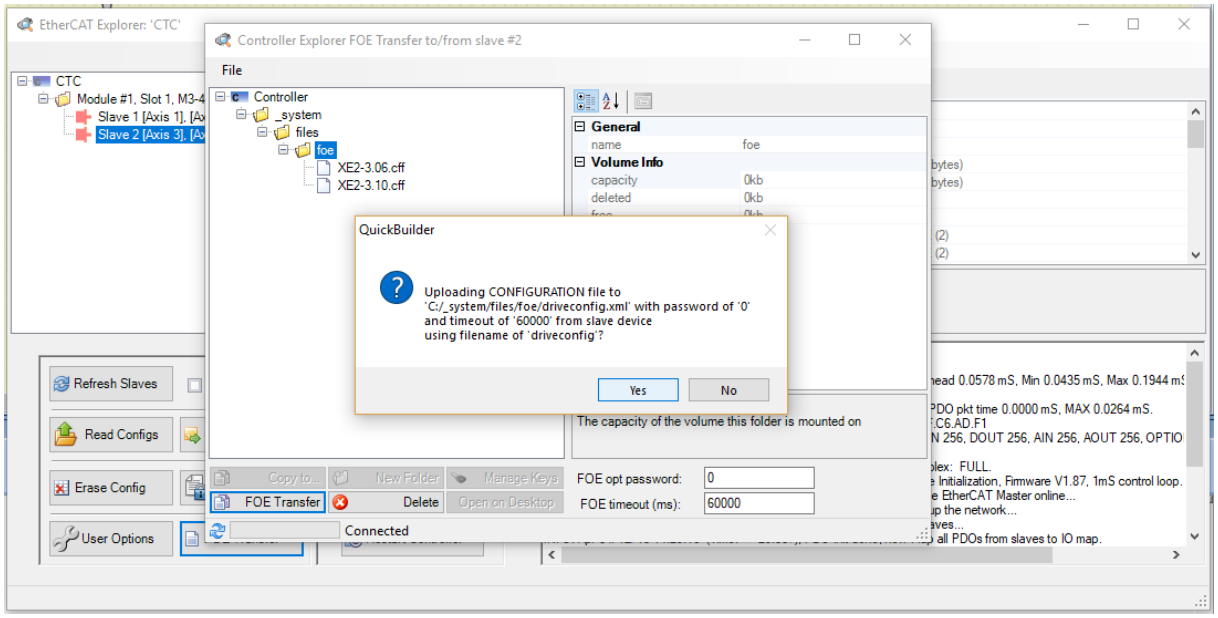
Upon opening a dialog box will tell you that if you select just a folder followed by pressing the 'FOE Transfer' button then you will be uploading a file to that directory and will have additional forms which will let you set any custom file names. The default is driveconfig.xml since at the moment all uploads are drive configuration files. This may change as the protocol is more widely implemented. If you select a file followed by the 'FOE Transfer' button then that is for download to the slave device. Note that you can set a generic 'FOE opt password' (numeric only) and a timeout. Typically 60000 milliseconds or 1 minute is needed as that is the length of time it takes for drives to re-flash their firmware.

Below is an example of uploading a configuration from a drive. It may be used to download to other drives that may use the same configuration (drive settings per the manufacturer). First the folder to store the uploaded file into is selected, followed by pressing the 'FOE Transfer' button:

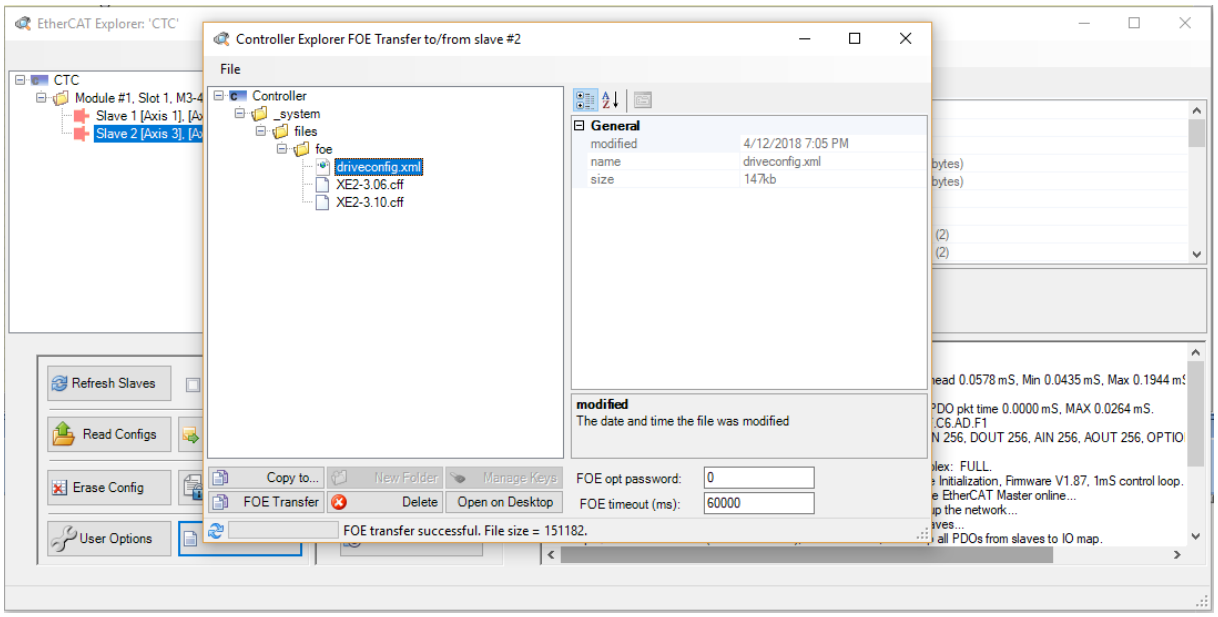


A form will appear which allows you to change the default system file name from 'driveconfig.xml' to something else. The name used by the slave device defaults to 'driveconfig' which is the standard for all drives. Upon

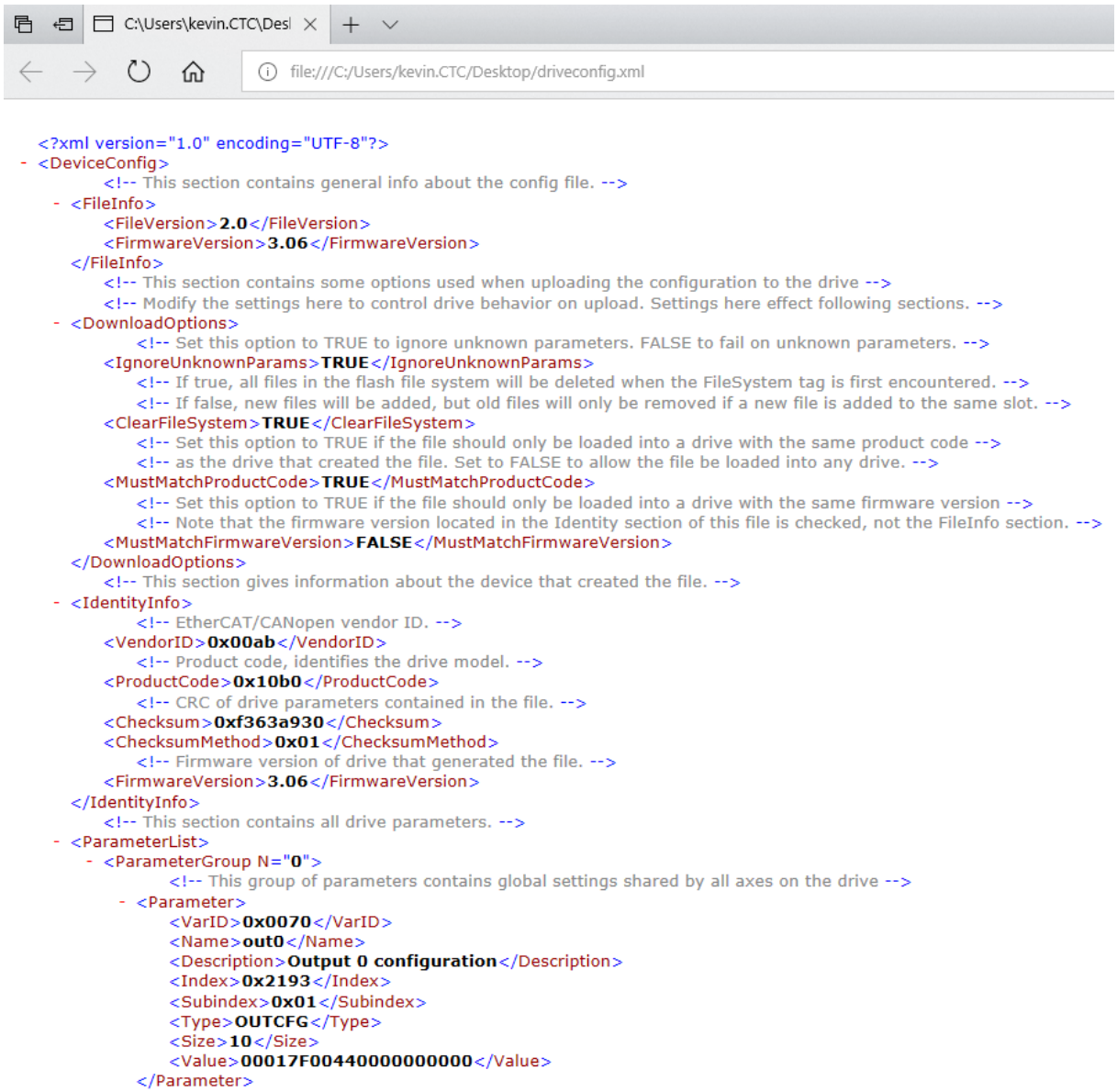
pressing OK another dialog will appear giving you a summary of what is about to occur. Hit the 'No' button to abort or 'Yes' to continue:



Upon selecting 'Yes' QuickBuilder will interact with the remote Incentive PC requesting the file be uploaded, the results will be displayed at the bottom of the form and the directory will be updated to show the new directory:



The highlighted file 'driveconfig.xml' was uploaded and at the bottom of the form it says it was successful and the file size was 151182 bytes. You can further select 'Copy to' to copy the file to you local QuickBuilder PC and/or the 'Open on Desktop' button to open the file locally for inspection. Below is 'Open on Desktop' showing the file with the Microsoft Edge browser:



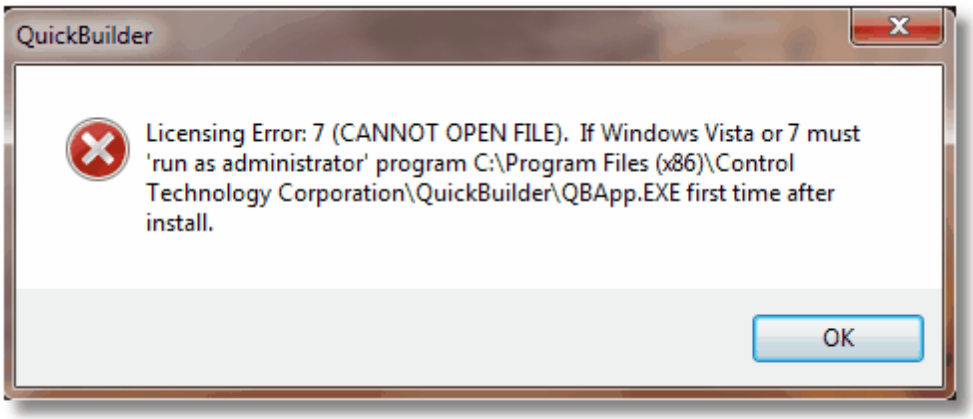
To download a file it is very similar to uploading except the file is selected instead of the folder. If a .cff file is selected (firmware) a notice will appear that the EtherCAT network will be stopped prior to downloading the file for safety reasons. Firmware can take up to a minute to flash to be patient. The status window will appear with the results after flashing. If the .xml file is downloaded then it is assumed to be a configuration file and the network will not be stopped. Be careful as the configuration file may change parameters within the drive that effect operation.

7 Chapter 7: Windows 7 and 10 Support

QuickBuilder has been fully tested with Windows 7 and Windows 10, both 32 and 64 bit, 64 bit is the preferred platform. In fact QuickBuilder is now developed on both of those platforms to ensure compliance. Install 'QuickBuilder Support' first, followed by the latest 'QuickBuilder Setup'. For legacy users a new 'QuickBuilder Setup' was made available in February 2010, mainly to address security issues that effect the 'C' compiler, cygwin1.dll.

After the initial install QuickBuilder must be invoked using the 'run as administrator' menu option. This is needed to establish machine wide licensing. Windows is very security minded and thus many things are limited to a user level. Once run you can exit immediately and reinvoke using the desktop shortcut, QuickBuilder will then be run as a normal application.

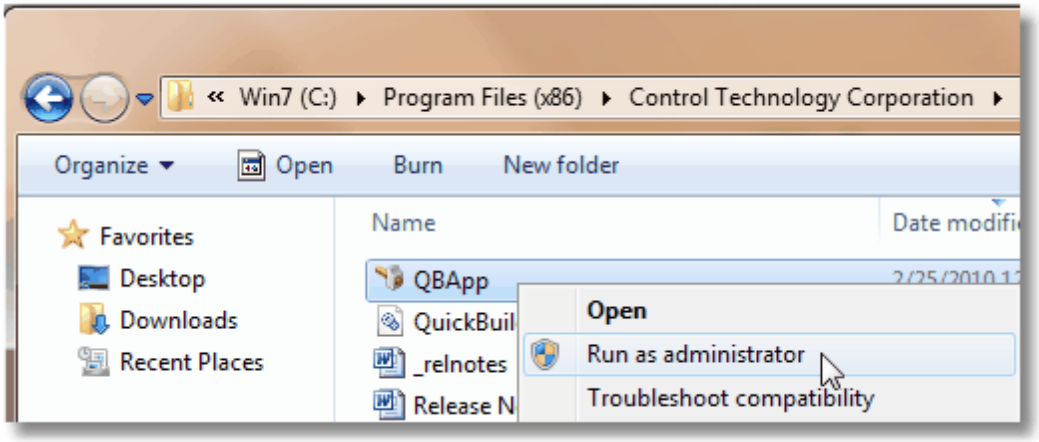
Failure to 'run as administrator' after install will cause the following to appear:



Followed by:

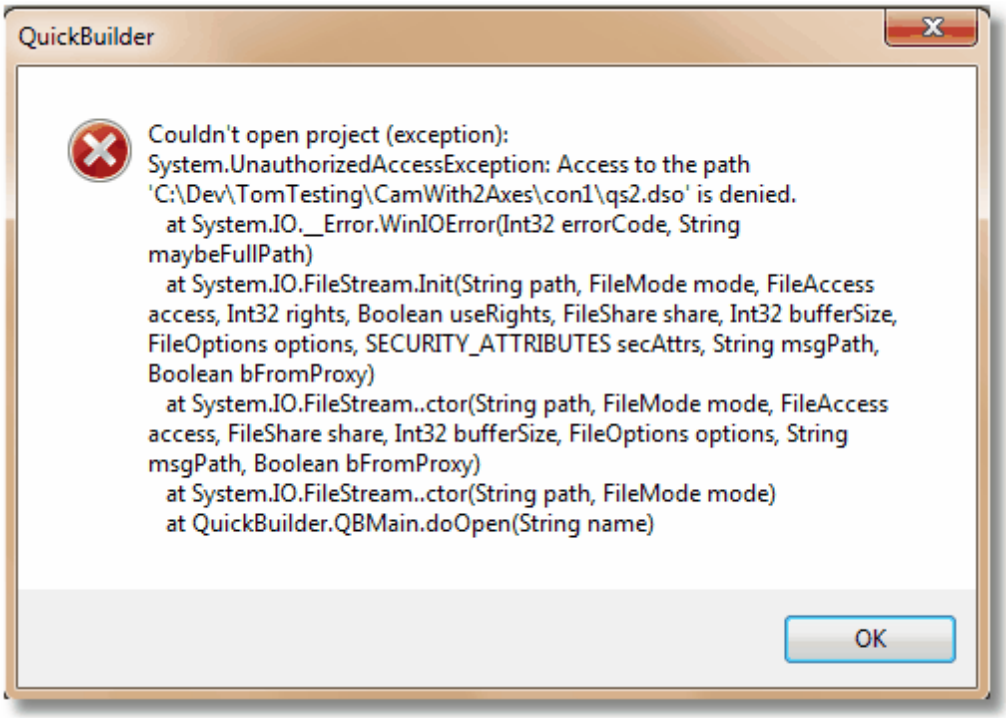


To ensure this does not happen simply invoke the QBApp.exe file as suggested by the Licensing Error dialog, right clicking on the exe file as shown:

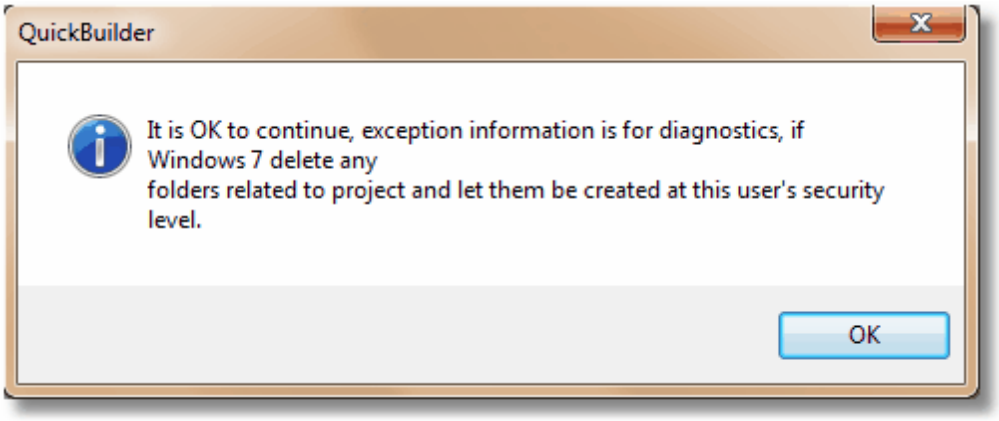


Simply execute and exit immediately. Note that this is needed regardless of whether you have administrator privileges. It is not needed if UAC (User Access Control) is turned off.

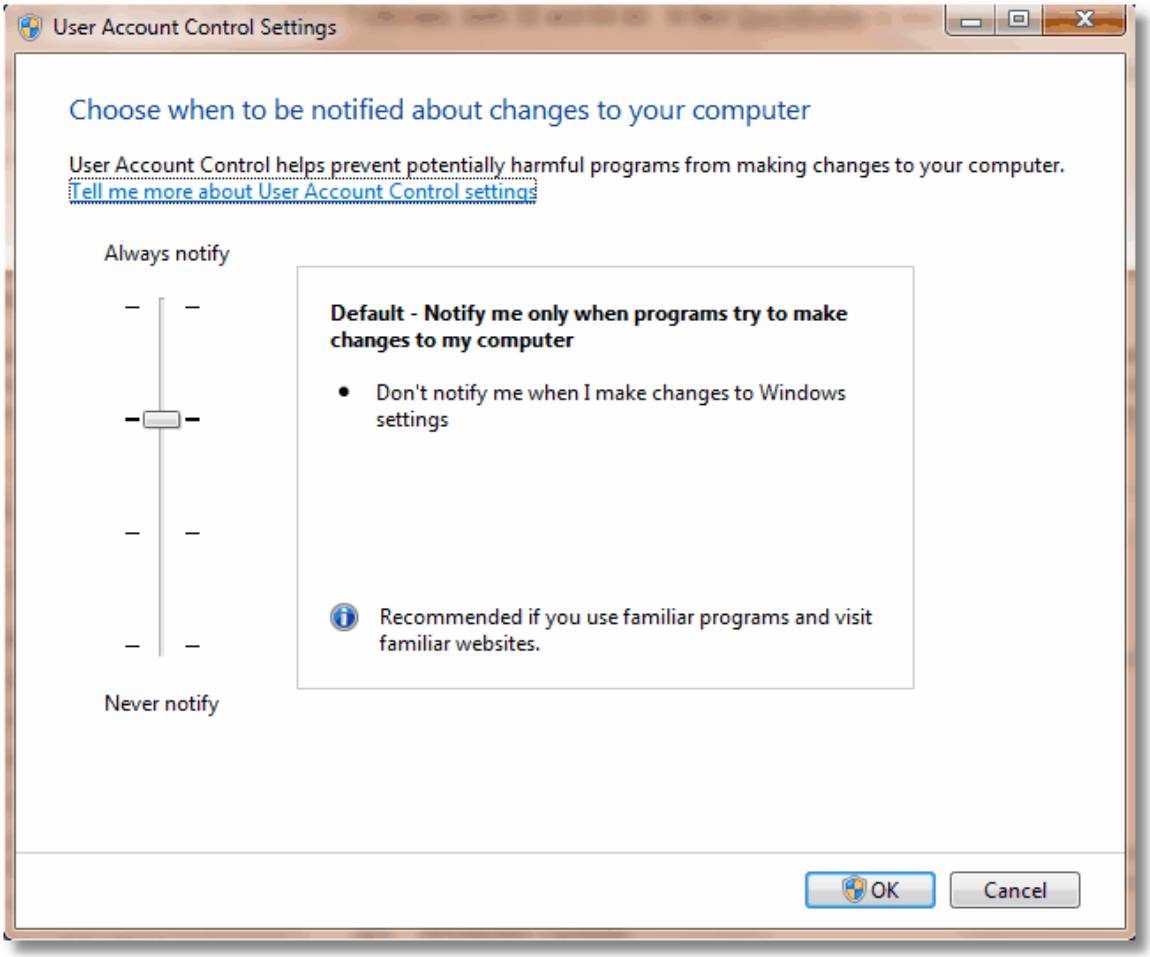
Numerous problems can arise due to security access problems. This is typically caused by a user attempting to write to something above their user security level. Although QuickBuilder conforms to Windows 7/10 security, something as simple as copying a full project from another computer to your disk can cause a security violation when you open the project. This is because the sub-folder of the project is from another system and not owned by the user. The simplest way around this is to only copy a project file (qdp) and not the sub-folders. The folders will automatically be created during translation. If a problem still persists make sure that the user had full control permissions on the sub-folder. Ideally it is best to keep files in the User's Documents area to avoid problems. Failure to have the proper security level can cause this message:



Followed by:



Windows 7/10 has been tested with the following User Access Control setting:



Setting it to "Never notify" will require a restart to become active and will disable UAC. This will resolve most security confusion if problems should occur.

8 Appendix A: Shortcut Keys

There are a number of shortcut keys that are useful within QuickBuilder. The following table describes these keys.

Key	Description
Control-R	Insert Resource This key is available in the code editor window to automatically insert a resource by name from a drop-down combo box. This key is also available in the QS4 step editor when building expressions, or when a variable name is required.
Control-Arrow	SFC Diagram selector The arrow keys "move around" the SFC diagram while editing in the code editor window. For example, if you are on a step, and want to edit/view the next connected step, then pressing Control-Down-Arrow would take you to that step.
Control-E	Switch editors This key toggles between the code editor and the QS4 editor windows.
Control-M	*Make* a new step This key (while in the code editor) creates a new SFC element. Essentially equivalent to the SFC construct toolbar, this is a non-mouse method to add a new SFC step.
Control-N	Add a new statement (after) This key brings up the QS4 editor to add a new statement *after* the present statement. It is equivalent to the *Insert Statement After* command in the QS4 editor window.
Control-Shift-N	Add a new statement (before) This key brings up the QS4 editor to add a new statement *before* the present statement. It is equivalent to the *Insert Statement Before* in the QS4 editor window.
F3	Find Next This key finds the next occurrence of the *Quick Find* text entered in the code editor window.

9 Appendix B: Known Anomolies & Warnings

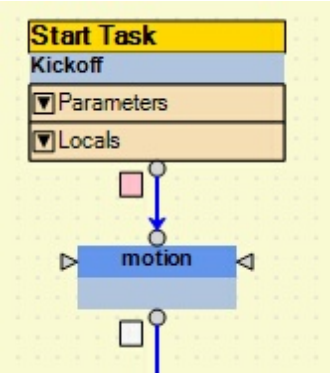
This section lists known issues with the current revision QuickBuilder and suggested workarounds:

Quickstep/QuickBuilder Project Upgrade Warnings

1. **Digital Input ActiveState –** The digital input activeState property is functional, previously the property was ignored. Thus setting it to 'false' will cause the input to be inverted, 'true' being the default state.

2. **Reserved words –** To handle the importing of Quickstep some new reserved words were added whose use in older programs will now generate a compile error: _rol, _ror, _servoInfo, at, accel, and, const, cw, ccw, D, deadband, down, hardstop, I, maxspeed, motion, of, position, P, profile, reset, rotate, scount, search, servo, shift, softstop, tgoto, timeout, turn, up, zero. If any of these words are used as resources or step/task names in your program they must be changed. This includes MSB properties referenced from a main QuickBuilder step using the 'dot' property.

3. **Reserved Words In Step Names** - In looking for step names that may be using a reserved word as its name, typically the link prior to that having a reserved word will be pink versus white. For example 'motion' is a reserved word, below shows the error:



To correct this problem the step named 'motion' would have to be edited, for example changed to 'motionx', as well as any references. To resolve the pink box and clear that error simple double click on the box and the parser will then see the step name has been changed and clear the error. Sometimes you need to click on the clear box below the step as well should a translation error continue to occur.

Negative Numbers in Boolean Expressions

Expressions which are used to generate a boolean (true/false) result must have negative number enclosed in their own parenthesis, for example -5 would be (-5). Some statements which this would occur are in 'if' and 'while'. For example the correct way to detect to see if a number is within a particular range would be:

```
if ((x > (-5)) && (x < 5)) then
```

Note the extra parenthesis around the -5. Failure to do this will result in the -5 being interpreted as a 0.

Menus can hang open on the Resource Tree

Periodically, and only on some computers, if you right click on the Resource Tree you must make a selection to close the menu window. This appears to be a Windows issue with the mouse driver and especially occurs with a wireless mouse. The present fix for this is to either make a selection to close the menu or simply restart QuickBuilder and the problem typically goes away.

Controller CRC Mode

An enhanced version of the controller CRC mode has been added. Previously every translation generated a 'unique' CRC but now with the introduction of source level debugging the CRC is used to detect code changes in the controller versus what is resident within your project on the PC. By default the CRC mode is now set to 'common' and is imported that way from previous revisions. A new feature allows you to set the mode to 'common' where by the CRC will only change when the source code changes, not on every translation.

XVar Constants and Strings

Usage of XVar Constants and strings within a project will make that project not backward compatible to releases including and prior to September 2009, since those programs do not support this feature.

Filenames and Revision Numbers

If a user names a file with more then 20 characters and changes a rev number (i.e "thisfilenameisover20charectersV1.2.qbp" vs. "thisfilenameisover20charectersV1.3.qbp") when making changes it appears as though the new version 1.3 will over write his old one 1.2.

Events and Tasks

There can be no more than 96 Events in a system and the total number of running tasks and pending events can not exceed 96.

M3-40 MPPR

The M3-40 motion module contains numerous properties. If one is not being used it is recommended to leave them set to the defaults. The default for 'mppr' is currently 4096, this can be changed to any non-zero value. Setting to zero will cause an error in the motion module.

M3-40 Module Definition in Rack

If the M3-40 motion module is included as a rack resource and the 40 card is not in the controller at runtime, a fault will occur. The controller initialization code will attempt to initialize the 40 card but its presence will not be seen, which is an error. Remove the card from the QuickBuilder resource rack if not present in the controller.

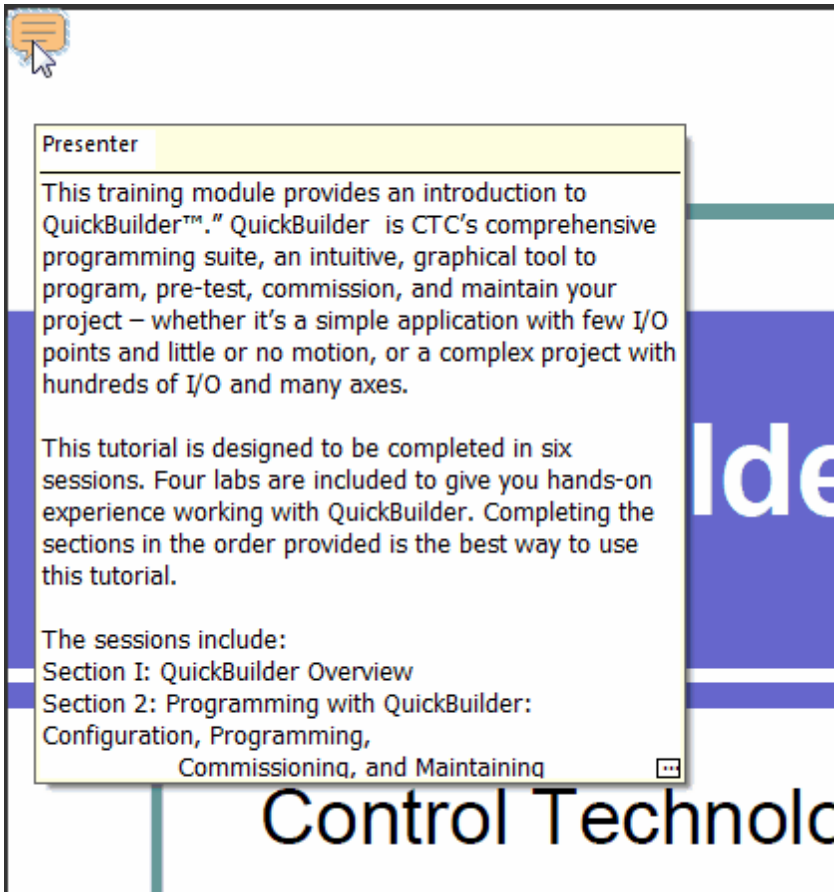
10 Appendix C: Training

PDF versions of the Training slides for QuickBuilder as well as Motion control are available via the below hyperlinks:

QuickBuilder Training

Motion Control Training

Note that in the top left corner of each presentation page there are notes which should be helpful in understanding the slide:



QuickMotion Reference Guide

Copyright © 2007-2018 Control Technology Corp. All Rights Reserved.

Control Technology Corp.
25 South Street
Hopkinton, MA 01748
Phone: 508.435.9595 • Fax 508.435.2373

Document No. 951-530017-019

⚠ **WARNING:** Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.

The information in this document is subject to change without notice. The software described in this document is provided under license agreement and may be used and copied only in accordance with the terms of the license agreement. The information, drawings, and illustrations contained herein are the property of Control Technology Corporation. No part of this manual may be reproduced or distributed by any means, electronic or mechanical, for any purpose other than the purchaser's personal use, without the express written consent of Control Technology Corporation. Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions. See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

Model Number	QuickBuilder Revision	Controller Firmware Revision	M3-40 Firmware Revision	M3-41 Firmware Revision	IncentiveECAT Embedded PC
5300	>=1.2.6620 02/15/2018	>= 5.00.90R70.11	>= 1.45	>= 1.84	>=1.84

1 Chapter 1: Introduction and Overview

This document provides details about adding motion control to a QuickBuilder project. QuickBuilder is CTC's integrated desktop development environment for the 5300 and Incentive PC series automation controllers. The primary programming language used in QuickBuilder is QuickStep4 (QS4). The QuickStep multi-tasking state language was invented by CTC in the 1980s to simplify the process of programming high performance machine control applications. Over the years QuickStep has been continually refined, and now it has been extended with the addition of QuickMotion to be able to easily handle even the most demanding motion control applications in a very intuitive manner.

The focus of this document is the QuickMotion extension to QuickBuilder. It is assumed that the reader is already familiar with the QuickBuilder environment and programming language. This document should be used in conjunction with the QuickBuilder Reference Guide.

This document is valid for use with the following motion modules:

- M3-40A: 2 Axis Servo Module
- M3-40B: 3 axis Stepper/High-speed Counter Module (24V)
- M3-40C: 3 axis Stepper/High-speed Counter Module (5V)
- M3-41A: 5300 Hardware Module for EtherCAT (reference this guide and EtherCAT Application Guide for added features)
- IncentiveECAT: Embedded PC Real-time Software for EtherCAT (reference this guide and EtherCAT Application Guide for added features)

Detailed data sheets for these motion modules may be found on CTC's website, www.ctc-control.com.

⚠ Note: The M3-41A is available both as hardware, for the 5300 PLC, and as a software component as part of a PC based EtherCAT master, IncentiveECAT. IncentiveECAT executes the exact same M3-40 motion control code except with enhancements for a virtual hardware environment. IncentivePLC runs on the PC as well, running the same QuickBuilder software, hence full compatibility with their hardware (5300 PLC) counterpart. From a software perspective almost all of the commands used on the M3-40 module apply directly to the M3-41A and IncentiveECAT. References to software features and language instructions that follow for the M3-40 can be assumed to apply to the M3-41A/IncentiveECAT except as noted.

1.1 Guide to Symbols

Features that warrant caution or special consideration are denoted by a ⚠.

A command or statement that is supported by a given mode or block is denoted by a checked box ☑.
Unsupported commands and statements are denoted by an empty checkbox ☐.

1.2 Brief Overview of Motion Control

1.2.1 Servo Motor Applications

Background

A servo motor is used in a closed loop control system, where the controller has information about both the actual position and velocity of the motor as well as the desired position (or velocity). The controller then adjusts the motor's output to remove the difference between the actual and desired values. Because this system has information about the error, and the output (which is usually proportional to the motor power) increases as the error increases, it can require very little power when the error is small.

This means that the average power needed for a high performance application may be considerably less than the peak power, so smaller motors and drives may be used.

There is usually a Servo Drive module between the motion controller and the motor that accepts the control signal (torque or velocity command) from the motion controller (a low current signal in the range -10 Volts to +10 Volts) and converts it into the high power (depending on the motor, several amps of current at 24V to 200+V) signals required by the motor. The Servo Drive must usually be configured to match the Motor (or is designed specifically for the motor). The drive and the motor are often, but not necessarily, made by the same manufacturer. The motor may be a simple brush type DC motor (which is low cost but requires periodic replacement of brushes) or a Brushless DC or AC motor, which requires additional circuitry in the Servo Drive to handle electronic commutation and will generally require additional sensors and signals from the motor to the driver.

Controlling the Servo Motor

The Model 5300 automation controller can be used to control up to 64 axes of servo motors. To control motion, an M3-40A motion module is added to the system. The M3-40A module is a dual axis servo controller that can control 1 or 2 servo motor systems with Analog Torque or Velocity command and Quadrature encoder feedback. Additionally M3-41 EtherCAT modules can be add network based drives and IO.

Servo Command Output

The output of the Servo Controller is an Analog signal that can vary from -10V to 10V with 16 bit resolution. The analog output is used, via a servo amplifier, to control the current in a DC motor generating torque at the shaft. The amplifier may also handle other functions such as commutation for a brushless motor or it may use the analog input to control the velocity. Care must be taken in the wiring to minimize the possibility of errors being introduced into the signal by noise induced from any high power switching circuitry near to the system, since this will directly affect the quality of the control.

⚠ Shielded cabling must be used between the Servo Controller and the Servo Drive and the distance between them should be minimized.

Encoder Feedback

An encoder mounted to the motor generates two pulse signals (A, B) that are used by the M3-40A module to track the motor position. The M3-40A module can also accept a third encoder channel (the Z axis or Index) that can be used to identify a specific point in the motor rotation. This Z pulse is typically used for accurate homing of the motor.

The M3-40A encoder inputs accept a quadrature differential signal for the A and B encoder channels. The index pulse, or Z channel, is also accepted as a differential signal. The direction is counted positive, or clockwise (CW)

when the A encoder phase leads the B encoder phase. Indicator LEDs for each servo axis on the module indicate the states of the A and B channels.

For some Brushless Servo systems, the Servo Drive also uses an encoder for information about the position and will provide a set of suitable encoder outputs for connection to the Servo Controller. In this case the power for the encoder is usually provided by the Servo Drive and it is not necessary to connect power for the encoder, but it is recommended that the controller's 5V return be connected to the common or return for the servo drive's encoder outputs. This limits the common mode voltage between the drive and controller and helps protect the encoder input circuits from damage caused by over voltage.

⚠ 5 VDC power is available from a dedicated 5V connector on the Model 5300 power supply modules. This connector also has a 5V return that is common to the controller's 24V return.

⚠ Shielded cabling should be used for the encoder wiring between the Servo Controller and the Servo Drive and the distance between them should be minimized.

⚠ *When the encoder output is provided by the Servo Drive, care must be taken that the signals are actually encoder signals and are not a simulated encoder generated by the Servo Drive from other signals. When the outputs are simulated encoder signals, there is generally a delay between the movement of the motor and the encoder signal generation. When this delay is small this is not a concern, but since the M3-40A updates the servo command at a rate of over 1250Hz, delays as small as 200µs can be significant.*

⚠ *Reference the EtherCAT Application Guide for details on the M3-41 and IncentiveECAT modules.*

The M3-40A module also has five high speed inputs and five high speed outputs that can be configured for a wide variety of functions via software. See IO Assignments later in this chapter and the Model 5300 module data sheets at http://www.ctc-control.com/customer/techinfo/idxdocs5300.asp for more details.

1.2.2 Stepper Motor Applications

Background

Stepper motors are typically used in open loop applications. A stepper motor has a fixed number of magnetic poles that determine how many steps the motor will move through during one revolution. Most stepping motors have 200 full steps that can be subdivided into smaller increments via microstepping technology built into the stepper drive. Microstepping drives can boost the number of steps per revolution to 50,000 or more providing smoother motion and more precise positioning.

Controlling the Stepper Motor

The Model 5300 automation controller can be used to control up to 64 axes of stepper motors. The motors are connected to a matched stepper drive, and then the stepper drive is commanded by the Model 5300 motion module. To control motion, an M3-40B or M3-40C stepper motion control module is added to the system. These modules are configured in QuickBuilder to match the steps per revolution of the stepper drive so that programming can be done in user units. The M3-40B/C module is a dual axis stepper controller that can control up to three stepper axes by putting out a step and direction command to the drive.

Encoder Feedback (optional)

Normally, stepper motor applications are designed to operate in an open-loop mode where there is no encoder feedback. However the M3-40B/C modules have one encoder input for each primary axis and they can be configured to monitor position via the encoder as a check on the commanded position. The encoder inputs accept a quadrature differential signal for the A and B encoder channels. The index pulse, or Z channel, is also accepted as a differential signal. The direction is counted positive, or clockwise (CW) when the A encoder phase leads the B encoder phase. Indicator LEDs for each servo axis on the module indicate the states of the A and B channels.

⚠ 5 VDC power for encoders is available from a dedicated 5V connector on the Model 5300 power supply modules. This connector also has a 5V return that is common to the controller's 24V return.

⚠ Shielded cabling should be used for the encoder wiring and the distance should be minimized.

⚠ *When the encoder output is provided by the Stepper Drive, care must be taken that the signals are actually encoder signals and are not a simulated encoder generated by the Stepper Drive from other signals. When the outputs are simulated encoder signals, there is generally a delay between the movement of the motor and the encoder signal generation. When this delay is small this is not a concern, but since the M3-40B/C updates the stepper command at a rate of over 1250Hz, delays as small as 200µs can be significant.*

The M3-40B/C modules also have five high speed inputs and five high speed outputs that can be configured for a wide variety of functions via software. See IO Assignments later in this chapter and the Model 5300 module data sheets at http://www.ctc-control.com/customer/techinfo/idxdocs5300.asp for more details.

1.3 Brief Overview of M3-40/41 Motion Module Features

High performance motion control can be easily achieved with Blue Fusion Model 5300 automation controllers by adding one or more M3-40 motion modules. The M3-40 series modules are two axis motion control modules specifically designed for the Blue Fusion Model 5300 controller. They can be used to command motion on both servo and stepper motor drive systems. The M3-40 uses space saving design features that enable it to fit into a single rack slot in the Model 5300 controller. Up to 32 of the M3-40 modules can be installed into a single Model 5300 system, allowing for up to 64 axes of motion control.

Motion performance is maintained even as axes are added because each M3-40 has its own on-board processors that handle all motion related processing for two axes. CTC has fitted each dual axis module with a powerful RISC processor as well as CTC's new Motion Accelerator Chip (MAC). This gives the M3-40 modules the ability to run CTC's latest 64-bit floating point motion loops and handle local high-speed I/O events.

There are currently three M3-40 modules that can be used in the Model 5300 automation controller and one M3-41:

- M3-40A: 2 Axis Servo Module
- M3-40B: 3 Axis Stepper / High Speed Counter Module, 24V command
- M3-40C: 3 Axis Stepper / High Speed Counter Module, 5V command
- M3-41A: EtherCAT Master for both motion and IO automation control

Hardware Features

Each module is capable of controlling two axes of closed loop motion. The M3-40A can be connected to either stepper or servo drives. Each M3-40A axis has a precision 16-bit analog command signal that can command both torque and velocity mode drives, giving the designer great flexibility in motor and drive selection. Alternatively, each axis can also be set up to output step and direction signals to interface to stepper drives or intelligent servo indexers. The M3-40B and M3-40C do not have analog command capability and therefore are best suited for stepper applications.

All modules have two primary axes of control and most hardware and software functionality is divided accordingly. Each primary axis has encoder feedback inputs that operate at rates up to 17.5 MHz, accommodating even the fastest linear motors. Each primary axis has five fast user assignable inputs and five fast user assignable outputs. In addition, it is possible to configure two of the outputs on the M3-40 module (40A/B/C) to command a third open loop stepper. See the Alternative Stepper Output statement in the I/O Statements section of Chapter 4 for more on this topic.

Software Features

While the M3-40's hardware is impressive, its software capabilities are what really set it apart from the competition. The software has been designed to simplify and speed every step of the machine development process. To set up a motion axis, simply "drop" an axis object into the QuickBuilder Resource Manager. Then it can be easily configured using convenient user-units and other fill-in-the-blank properties. Dialog boxes and tuning wizards de-mystify the whole servo setup and tuning process.

CTC has taken a very modular approach to QuickBuilder's motion control capabilities. To create motion on an axis, one or more motion commands are placed in an object called a Motion Sequence Block (MSB). After creation, that MSB can be used by any of the axes at any time. A simple example would be a homing MSB – write it once, and then use it on as many axes as desired.

To further simplify the motion programming process, CTC has created an extension to the QuickStep language within QuickBuilder called QuickMotion, which has more than 50 new commands and more than 100 specialized motion variables. QuickMotion makes programming motion applications very intuitive. For example, if one wanted to move an actuator 3.76 inches in 1.25 seconds the command would be:

Move to 3.76 ***in*** 1.25

Of course, 3.76 could just as easily be a variable or an expression that is calculated on the fly.

M3-40 and M3-41 Module Data sheets

Refer to Document No. 950-534001: Model M3-40A data sheet at http://www.ctc-control.com/customer/techinfo/docs/5300_950/950-534001.pdf for more detailed information on the M3-40A module.

Refer to Document No. 950-534002: Model M3-40B data sheet at http://www.ctc-control.com/customer/techinfo/docs/5300_950/950-534002.pdf for more detailed information on the M3-40B module.

Refer to Document No. 950-534003: Model M3-40C data sheet at http://www.ctc-control.com/customer/techinfo/docs/5300_950/950-534003.pdf for more detailed information on the M3-40C module.

Refer to Document No. 950-534101: Model M3-41 data sheet at http://www.ctc-control.com/customer/techinfo/docs/5300_950/950-534101.pdf for more detailed information on the M3-41 module.

1.3.1 M3-40 & M3-41/IncentiveECAT Motion Module Features

M3-40

- Two axes of servo or stepper control per module
- Up to 64 axes per Model 5300 system
- Position loop update times of 800µs / 2 axes (as fast as 500µs under software selection)
- Encoder feedback up to 17.5 MHz
- 64-bit floating point loop control
- 16-bit analog command (M3-40A only)
- 5 user assignable inputs / axis
- 5 user assignable outputs / axis
- High speed registration capture
- High speed PLS outputs
- 48 user variables per axis

M3-41/IncentiveECAT

- 64-bit floating point loop control
- Up to 16 axes per network. 64 axis with IncentiveECAT. Multiple networks supported as well as redundant master (PC only) and secondary master
- Virtual axis/master support
- Up to 2000 I/O points using remote I/O
- 500uS, 1 mS, 2 mS, or 4 mS updates on all axes
- Any axis can track/gear/cam off any other
- Registration Capability
- Commands: linear, S-curve, Cam, Spline, Gear, Move on a gear, Segmented moves
- Syncs master to slaves - provides simultaneous motions
- EtherCAT motion modes:
  - Cyclic Sync Position
  - Interpolated Position
  - Profile Position
  - Profile Velocity
- Network auto configuration
- Support for absolute encoding
- Link software counters to any input
- Link PWM outputs to any output
- 48 user variables per axis
- M3-41 has 3 encoder inputs which can also be used as a master encoder for EtherCAT axis
- Segmented EtherCAT packets for large networks

1.3.2 Special M3-40 I/O Functions

- **16 HS Counters (10 MHz):** All five inputs as well as the A, B, and Z signal pins on each axis connector can be configured as high-speed counters.
- **Period Measurement (0.1 µsec accuracy):** Two pairs of inputs on each axis can be set up to measure the time between activation of the first and second input in the pair. Ideal for high-speed measurement and frequency measurement.
- **Frequency Outputs:** Three outputs on each axis can generate a programmable frequency up to 500 KHz.
- **Pulse Outputs:** All ten outputs can be pulsed for a programmable time value with an accuracy of 0.5 msec.
- **Programmable Limit Switch Outputs:** Three outputs on each axis can be configured to automatically turn on and off as a function of the encoder position. Up to sixteen on/off positions can be configured per axis. The on/off positions can be changed programmatically on-the-fly. This is especially useful to compensate for lead or lag time based on operating speed.

1.3.3 Drives & M3-41 IO

Some drives support inputs and outputs at the remote drive level. The MSB property 'dins' represents the raw inputs provided by the drive, up to 32 inputs (EtherCAT object 0x60FD.0). The first 10 inputs may be accessed using 'din1' to 'din10' bit properties; as with the M3-40 modules.

Outputs operate as they do on the M3-40 module, limited to 8 outputs at the remote drive level (object 0x60FE.1). Use the 'setout' and 'clrout' MSB instructions for access, where the first output is 1.

Local I/O is also present on the 5300 M3-41 EtherCAT module. This module has 6 inputs and 2 outputs which are global to all MSBs. The outputs are referenced as 9 and 10 when using the 'setout'/'clrout' instructions. The MSB property 'global_inputs' is used to read the 6 inputs, with the first bit being the first input. The MSB property 'global_outputs' can be used in addition to 'setout'/'clrout' for read/write operations of the local outputs.

Local global Inputs, P1 connector pins:

P1 -11 DIN1
P1-13 DIN2
P1-15 DIN3
P1 -12 DIN4
P1-14 DIN5
P1-16 DIN6

Local global Outputs, P1 connector pins:

P1 -9 DOUT1
P1-10 DOUT2

1. 'global_inputs' and 'global_outputs' axis properties may be accessed by QuickBuilder using the Axis name/property method: axisname.property. These two properties will contain the same value on all EtherCAT axes.

2. Chapter 6 discusses additional IO capabilities available from the MSB language using various IO arrays. These arrays give access not only to drive and module based IO but remote EtherCAT IO blocks such as those from Wago, Turck and Beckhoff. Some of the features include PLS, PWM, pulse, and atomic multi-bit access of 32 drive inputs/outputs, local and remote IO.

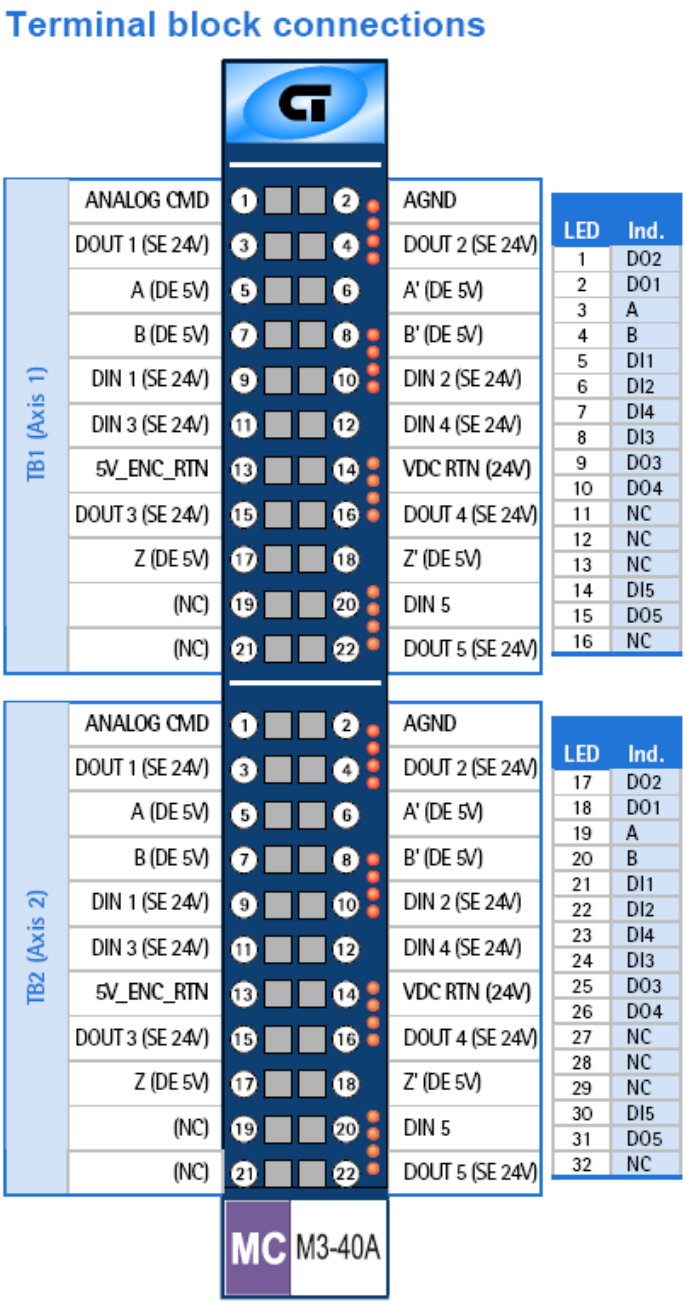
3. IncentiveECAT Version only supports drive I/O.

1.3.4 QuickBuilder Motion Control Features

- Axis objects configured in the Resource Manager
- New tuning wizard simplifies tuning
- Monitor motion parameters in multiple watch windows
- Use QuickScope to chart motion and I/O timing
- Simple English commands
- Over 100 new motion variables
- Full user-unit support
- Soft limits and hard limits
- Asynchronous event handlers

1.3.5 IO Assignments

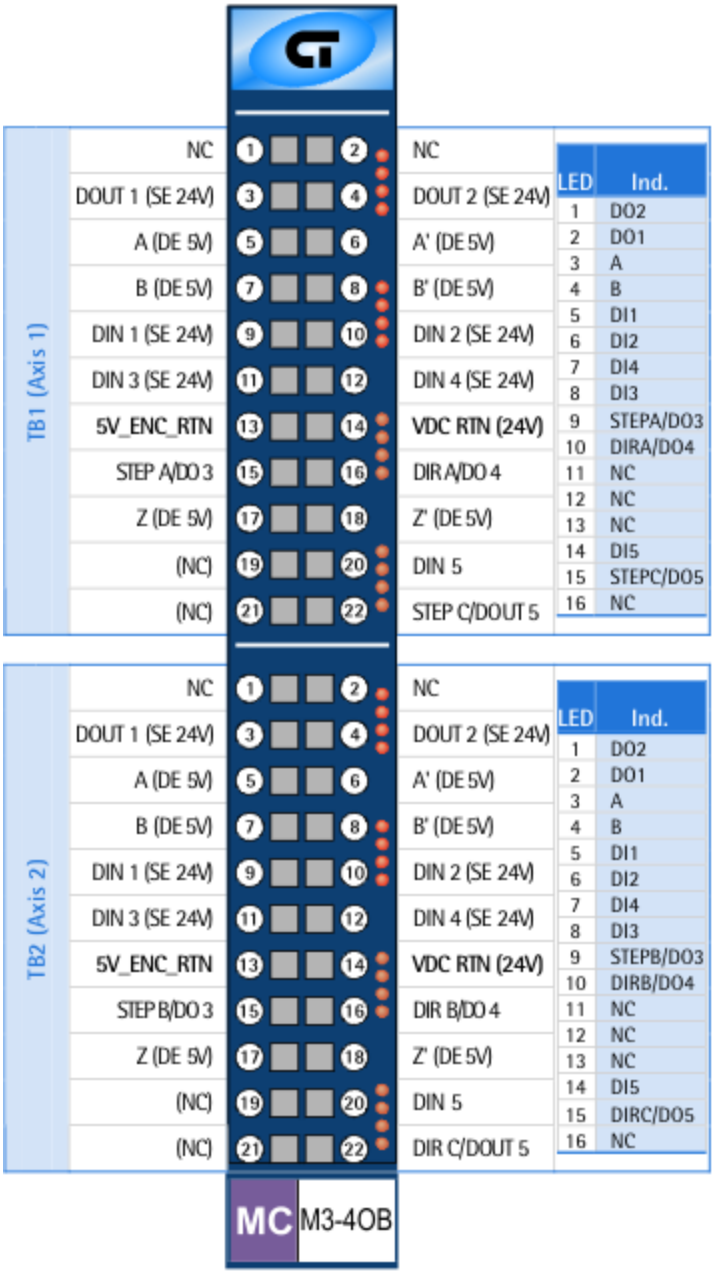
1.3.5.1 IO Assignments - M3-40A



⚠ Any two digital inputs can be configured in QuickBuilder to function as *registration inputs* 1 and 2. These digital inputs still function as general purpose inputs even when configured as *registration inputs*.

1.3.5.2 IO Assignments - M3-40B

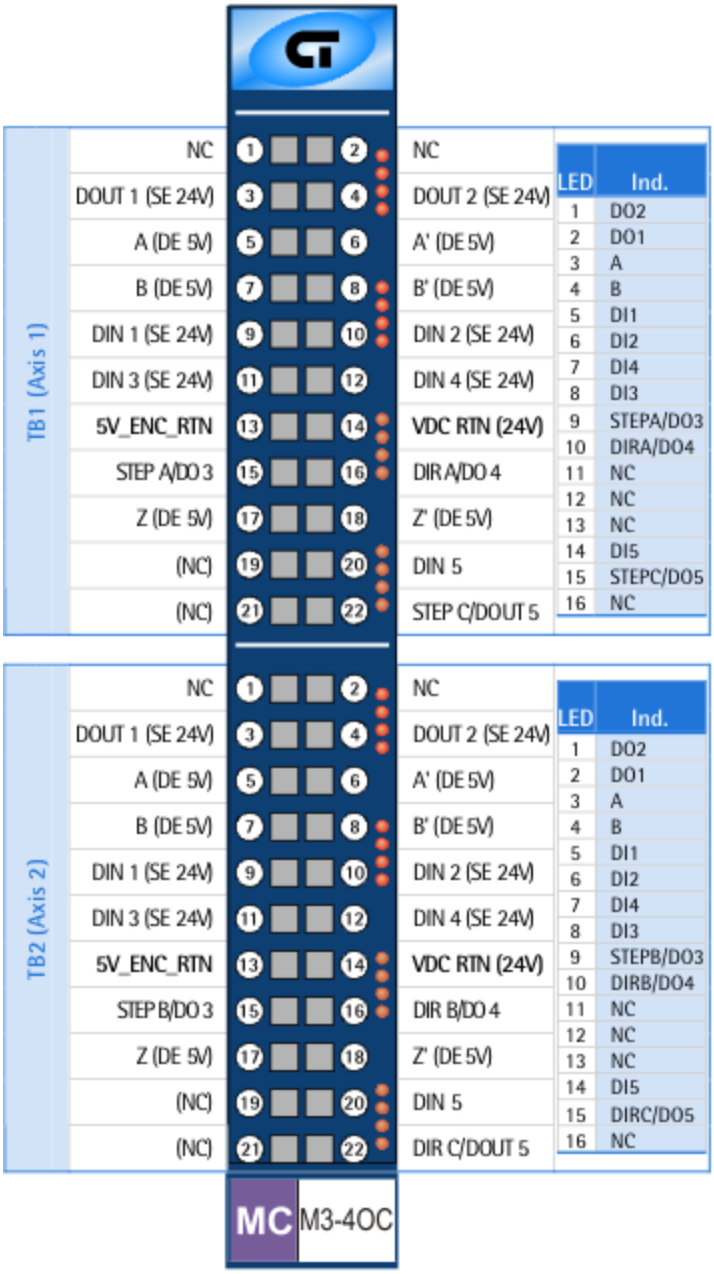
Terminal block connections



1. All step and dir connections are single-ended 24V.

1.3.5.3 IO Assignments - M3-40C

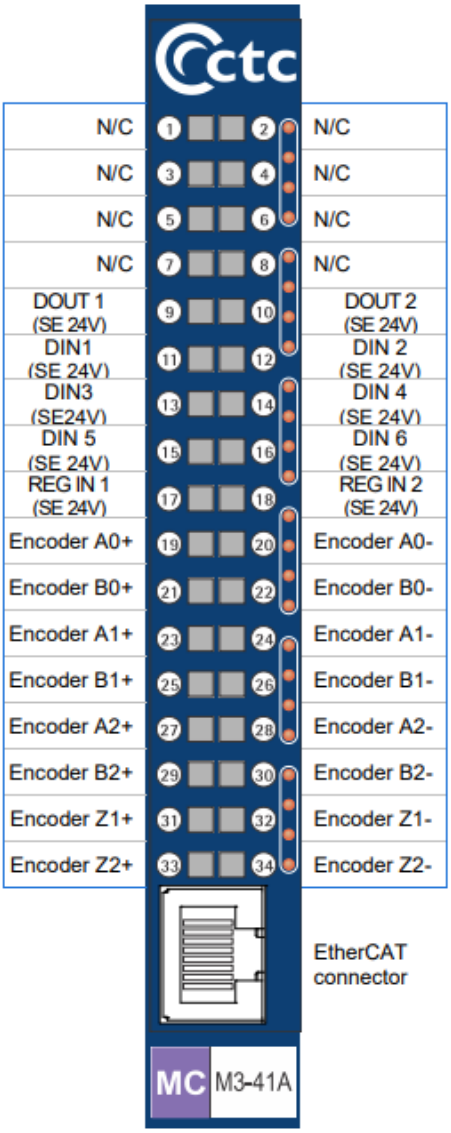
Terminal block connections



1. Step A/B and Dir A/B connections are single-ended 5V.
   Step/Dir C connections are single-ended 24V.

1.3.5.4 IO Assignments - M3-41A

Terminal block connections



Special I/O Functions

- **Three encoder inputs.** May be used as a master to any EtherCAT axis or as a general encoder
- **Six digital inputs**
- **Two digital outputs**
- **Two registration inputs**
- **Software counters**
- **PWM outputs**
- **Up to 16 RFID channels**

For more information refer to the CTC EtherCAT Applications Guide

2 Chapter 2: Motion Architecture

The Model 5300 PLC uses a powerful distributed architecture approach to solving machine control applications. The overall machine control program – called a QuickBuilder project – runs on the main CPU of the Model 5300 Automation Controller. It provides the primary guidance for the application and is in charge of communications with the outside world and the local Model 5300 I/O, motion, and specialty modules. The distributed nature of the Model 5300 design allows portions of the project to be passed to intelligent Model 5300 modules for local processing. This distribution of processing tasks and the overall coordination between modules and the main CPU is taken care of automatically by QuickBuilder.

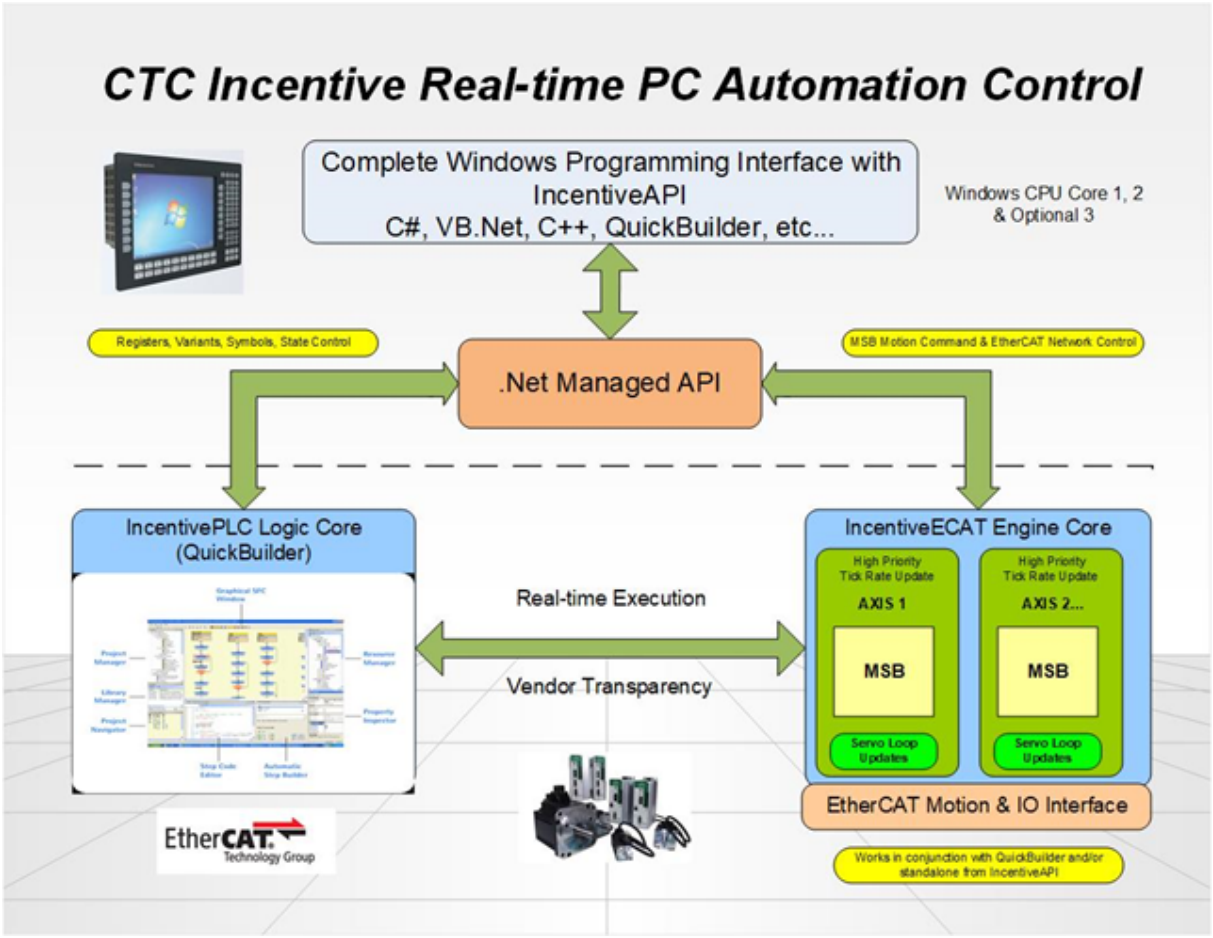
The result is a significant improvement in machine performance by off loading demanding processor-intensive functions like motion control tasks to specialized motion control processors on the Model 5300 Motion Modules. Even though this process takes place automatically, it's important for the automation engineer to have a basic understanding of the architecture of the Model 5300 controller and how it interacts with the QuickBuilder project.

The 5300 backplane can accommodate multiple modules of the following type:

- M3-40A: 2 Axis Servo Module
- M3-40B: 3 axis Stepper/High-speed Counter Module (24V)
- M3-40C: 3 axis Stepper/High-speed Counter Module (5V)
- M3-41A: 5300 Hardware Module for EtherCAT (reference this guide and EtherCAT Application Guide for added features)

With the release of EtherCAT motion support for the 5300 a new architecture was introduced, that of virtual network based devices. Once perfected it became apparent that there was a strong industry need to move the PLC and motion architecture to an embedded PC platform where a seamless interface could be provided to .Net programmers. This is known as the IncentivePLC and IncentiveECAT PC based software modules. These modules run the exact same environment as the 5300 PLC, in real-time, using dedicated cores of the PC. Both a real-time operating system and Windows run in parallel, communicating through shared memory. IncentiveAPI is provided to provide a seamless interface to the real-time world from programming languages such as C#, VB.NET, and C++.

The structure of the virtual PLC environment looks something like the following:

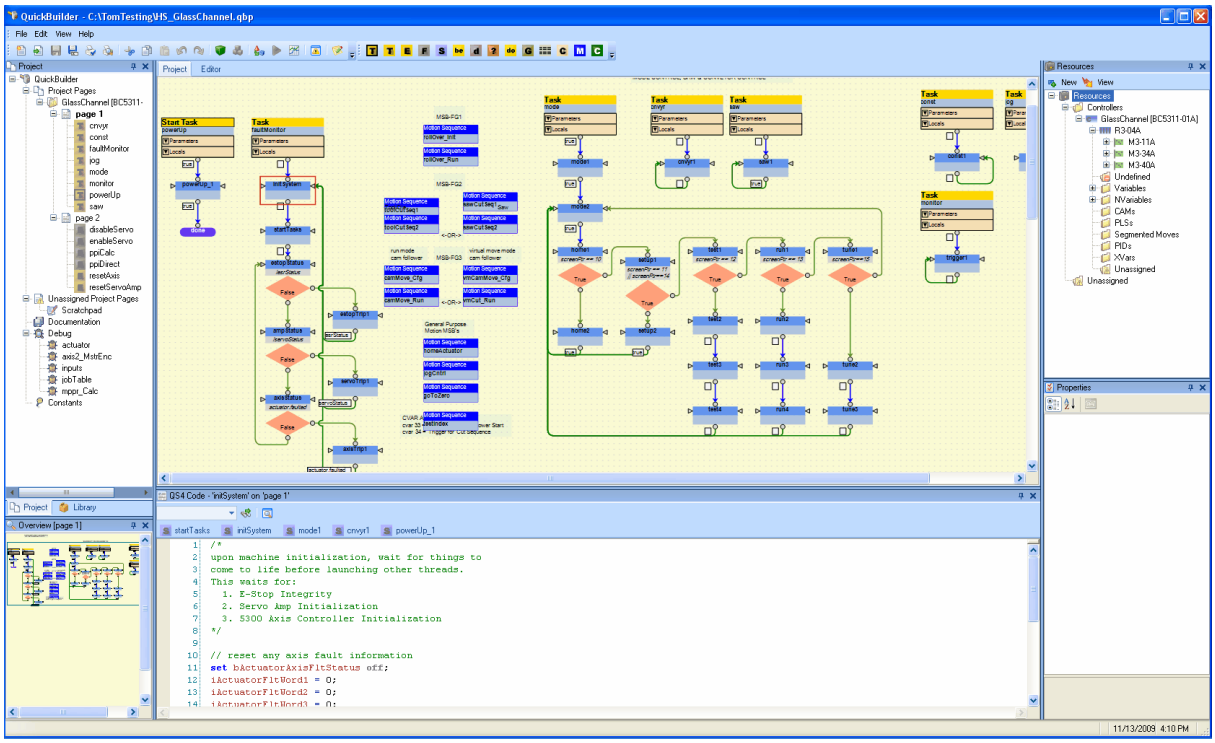


Both the 5300 PLC and Incentive products are programmed using the exact same tools. Before we get into the details of how to add motion to a QuickBuilder project, we'll first review the major elements of the software architecture:

- QuickBuilder, the software application used to program Model 5300 controllers
- QuickStep, the programming language used in QuickBuilder
- QuickMotion, an extension to the QuickBuilder application that is tailored to handling motion control.

2.1 QuickBuilder

QuickBuilder is CTC's innovative graphical development environment built using the latest .NET technology, making it very intuitive to use. It combines all the aspects of an automation project into one easy to use desktop application. This holistic approach to solving automation projects leads to quicker machine startups and simpler understanding of even the most advanced automation tasks. The key to simplifying the automation process is to break the overall process down into the operating states of each of its elements.



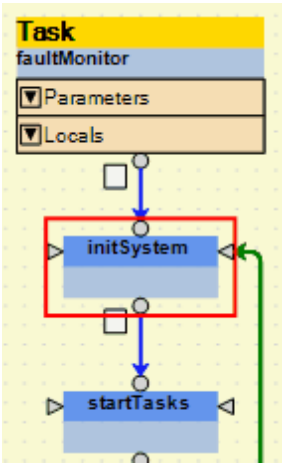
QuickBuilder desktop showing three tasks. A single step is highlighted in red.

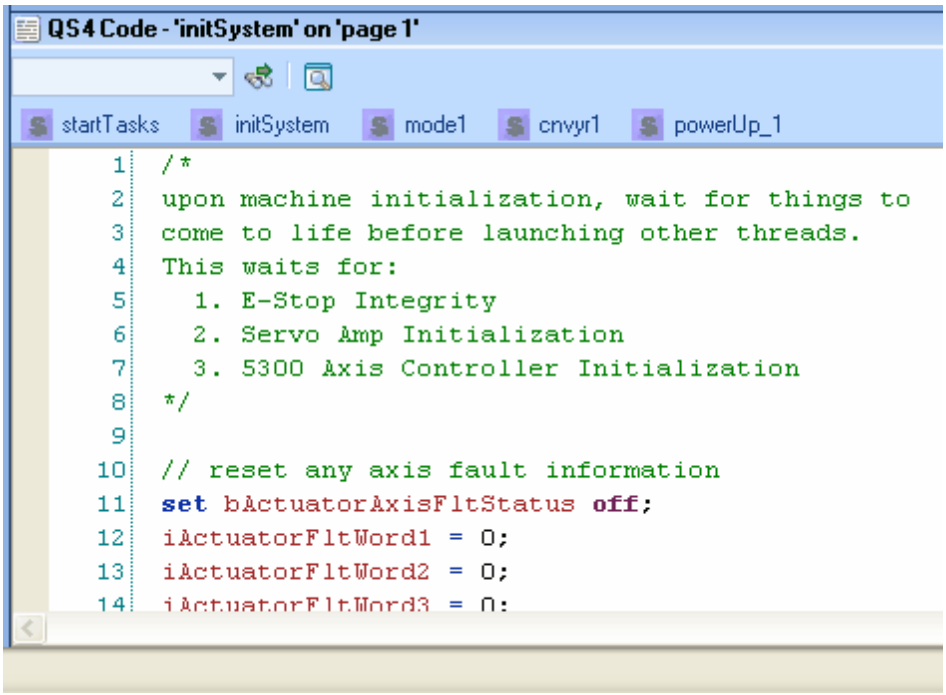
A QuickBuilder project is comprised of one or more tasks. Breaking the program into separate easily defined tasks greatly simplifies the programming process. A task contains multiple steps – where the steps represent a given operating state of the machine. Within the step are the actual instructions such as *wait for input*, *turn on output*, *move an actuator*, etc. It is also here at the instruction level that motion is initiated.

2.2 QuickStep

QuickStep is CTC's programming language used for the instructions within the steps. QuickStep was originally invented by CTC in the 1980's and has been proven in thousands of automation projects. Over the years CTC has continually refined and upgraded the language. The current version of QuickStep is QuickStep4 (QS4). The screen captures below show a highlighted step from the flow chart window that is automatically linked to the QS4 editor.

The use of QuickBuilder and QuickStep are covered in their respective manuals, and the user should be familiar with their use prior to starting a motion application.



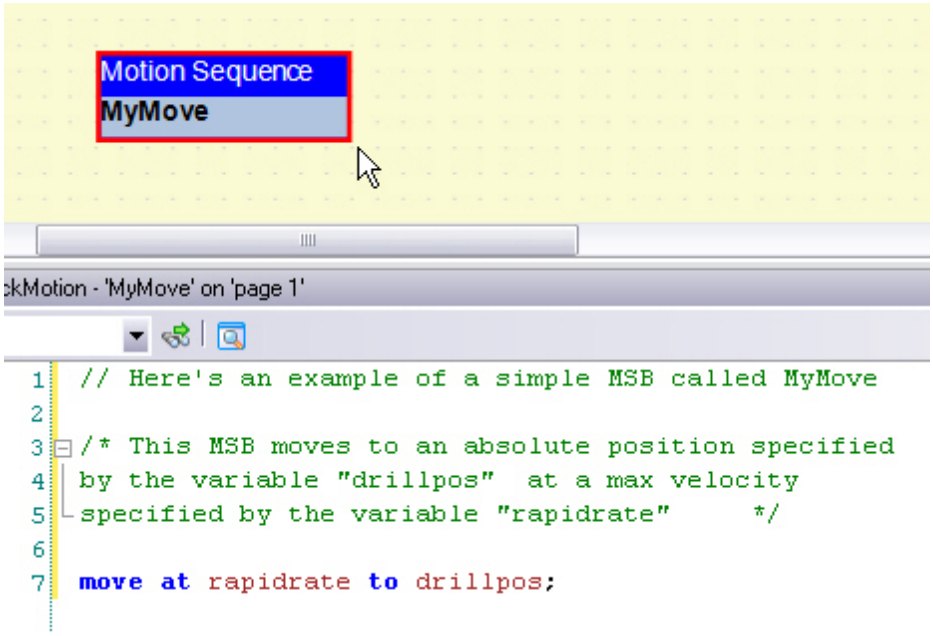


2.3 QuickMotion

QuickMotion is a specialized extension of QuickBuilder that is designed for motion control applications. It has been optimized to simplify the motion control process and to take advantage of the distributed architecture. QuickMotion instructions are entered into specialized tasks called Motion Sequence Blocks (MSBs).

The MSBs are coded within the QuickBuilder environment in the same way as steps are coded in QuickStep: Drag the MSB symbol onto the graphical desktop, give it a name, then use the editor to add the appropriate instructions. But there are two big differences:

1. an MSB is both a step and a task
2. a single MSB may be used by any number of axes.



By way of a practical example, think of the common motion control operation of homing an axis. In older control schemes, designers were either forced to write this homing code over and over in the program or call some generic homing routine hard coded by the motion control manufacturer. With QuickMotion, it is easy to create a customized homing MSB once, give it a name, and then use a QuickStep statement to start that MSB on any axis whenever an axis needs to be homed.

2.3.1 Adding Motion to the 5300/Incentive Application

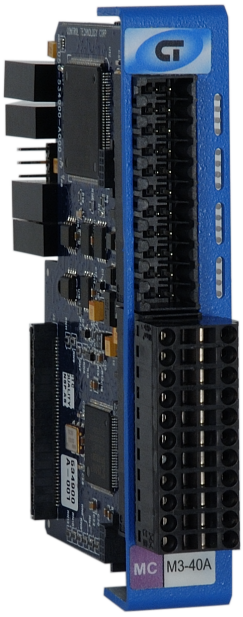
The main components used in motion control are:

- The Axis Module: The physical motion module in the rack
- The Axis Object: The QuickBuilder Resource representing an axis on that physical module.
- The MSB: The Motion Sequence Block, which contains one or more motion statements that execute on the Axis Module's CPU under the supervision of QuickStep.

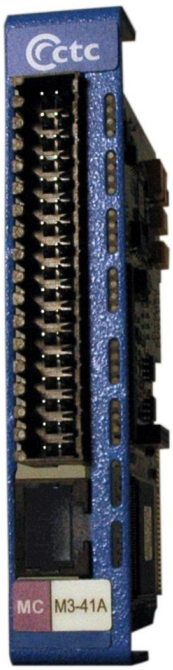
2.3.1.1 The Axis Module

A Model 5300 axis module is inserted into the Model 5300 rack just like any other I/O module. CTC offers axis modules that can control one or more motion axes. Each motion module contains its own CPU and Motion Accelerator Chip (MAC), ensuring consistent high performance motion control regardless of the number of axes to be controlled.

M3-40A: Example of a Model 5300 Axis Module



M3-41A: Example of a Model 5300 EtherCAT Module



IncentiveECAT is a software module running on am embedded PC that is responsible for all the motion control. IncentivePLC runs the main QuickBuilder logic application. Something called MSB's, or Motion Sequence Blocks executes in real-time within the IncentiveECAT process, detailed later. A popular choice for an embedded PC integrates the HMI with automation control as shown below:



2.3.1.2 The Axis Object

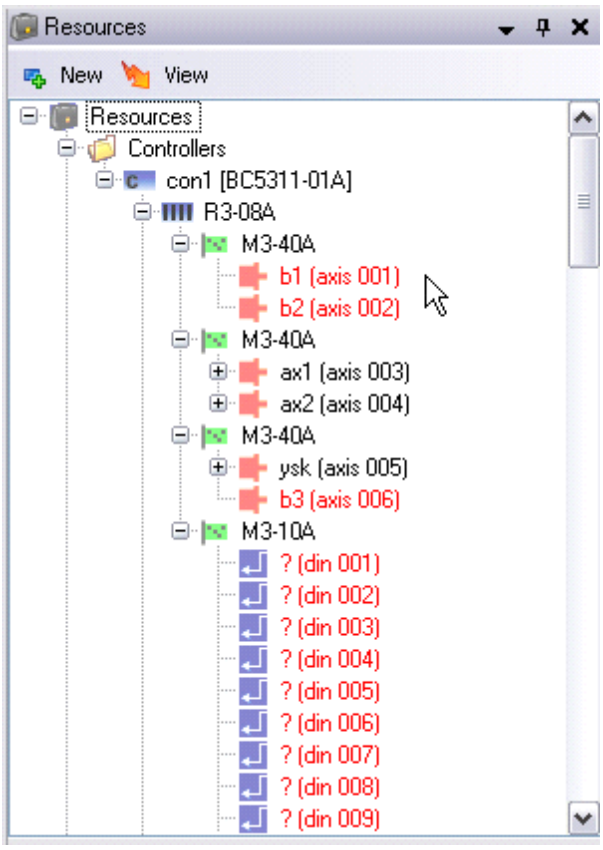
The *Axis* object represents a hardware-based or virtual axis associated with a servo or stepper drive. It is automatically created when a motion module is added to a rack in the Resource Manager. It typically consists of a controller module with various inputs and outputs that control the servo (or stepper) and usually feedback signals that are used to monitor position. Each axis can be commanded to perform some sequence of motion commands by the use of motion sequence blocks (MSBs). These MSBs appear in the QS4 program as stand-alone graphical elements.

Axis objects have many specialized properties that can be configured using the Property Inspector. Most of these properties can also be changed dynamically in the QuickBuilder project. Axis Objects have various inputs and outputs that control the servo (or stepper) and usually feedback signals that are used to monitor position.

When an MSB is selected, the programmed motion command sequence appears in the text editor window – the same window that is also used to edit QS4 code.

QuickStep4 can only start one motion sequence at a time for a given axis, but the active motion sequence can start other motion sequences (with some exceptions) that can run in parallel.

An MSB is not associated with any particular axis, which allows the same sequence to be reused many times for different axes.



2.3.1.3 The Motion Sequence Block

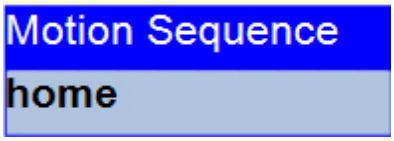
The Motion Sequence Block (MSB) element holds QuickMotion statement sequences. MSBs appear in the QuickBuilder project as stand-alone graphical elements. MSBs are not associated with any particular axis. This allows the same sequence to be reused many times for different axes, much like how a function works. MSBs are programmed using the QuickMotion language. An MSB may have only one QuickMotion statement, or it may have hundreds of statements.

The MSB is started on a given axis from QuickStep by using the *Start MSB* statement.

Once started, an MSB can start another MSB on its own that can run in parallel on the same axis. An MSB cannot start an MSB on another axis. This can only be done by QuickStep.

Up to 4 foreground MSBs can be running simultaneously. This limitation is imposed to guarantee high performance deterministic execution. A foreground MSB executes each of its statements at the loop update time of the Axis Module. This keeps them fast and in sync with the position loop.

In addition to the foreground MSBs, any number of background MSBs can be running simultaneously. The number of background MSBs is limited only by available memory on the Axis Module.



2.4 Controlling Motion from QuickStep

As mentioned earlier, QuickStep is in overall control of the project and as such, QuickStep has the ability to start and stop MSBs. There are actually only two Quickstep instructions pertaining to motion: *Start* and Stop.

- *Start*: Begins execution of the named motion sequence block (MSB) on the specified axis as a background MSB. This background MSB can then launch foreground MSBs on that axis. QuickStep can also directly launch foreground MSBs by using the FG option (start <axis> <msb> {optional FG/BG}, where FG is foreground and BG is background task.
- *Stop*: Stops execution of all foreground and background MSBs and thereby all motion.

In addition to these commands, QuickStep has extensive abilities to monitor and control MSBs on the axes while they are running via pre-defined and user-defined variables.

2.4.1 QS4 start Statement

This statement begins execution of the named motion sequence block (MSB) on the specified axis.

It is not an error to start another MSB when there is one already running for a given axis – however, if the named MSB is already running on a given axis, the start is effectively ignored.

```
start axis1 MSB1;       // start MSB1 on the axis called 'axis1', as
a background thread.

start axis1 MSB1 BG;    // start MSB1 on the axis called 'axis1', as
a background thread.

start axis1 MSB1 FG;    // start MSB1 on the axis called 'axis1', as
a foreground thread (run on each loop ticks, limited to 4).
```

2.4.2 QS4 stop Statement

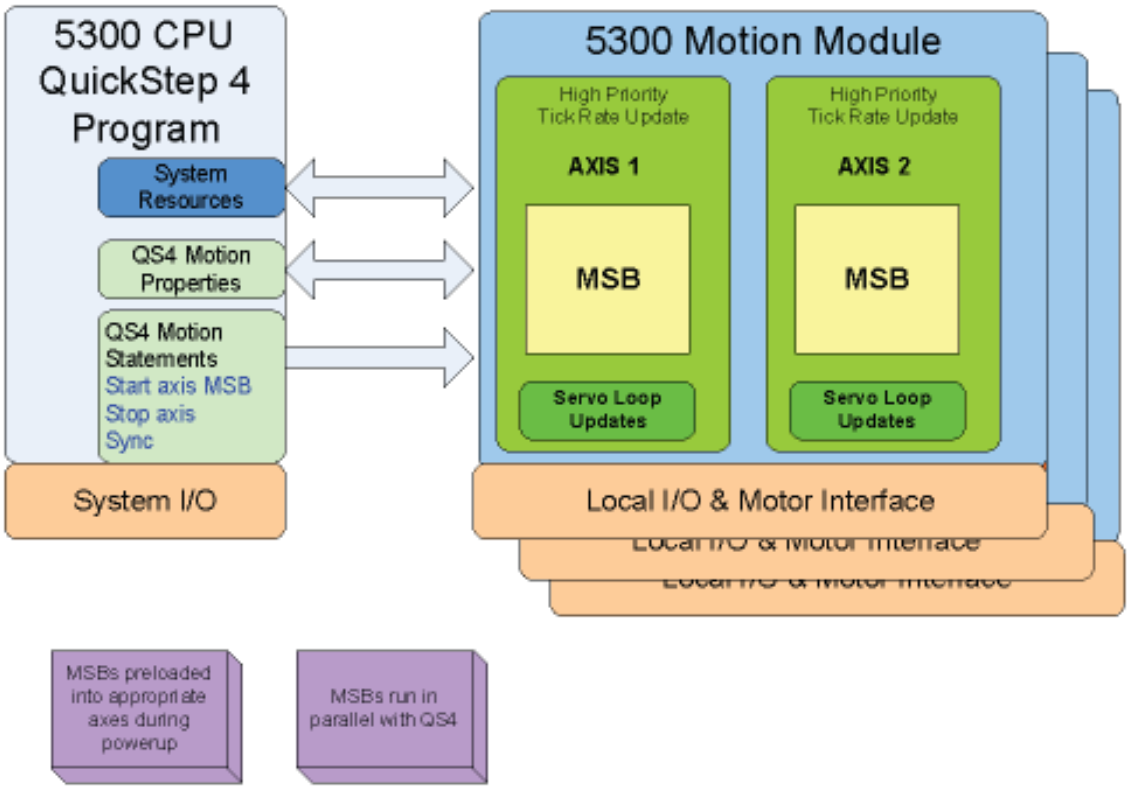
This statement stops execution of all MSBs on the named axis.

Example:

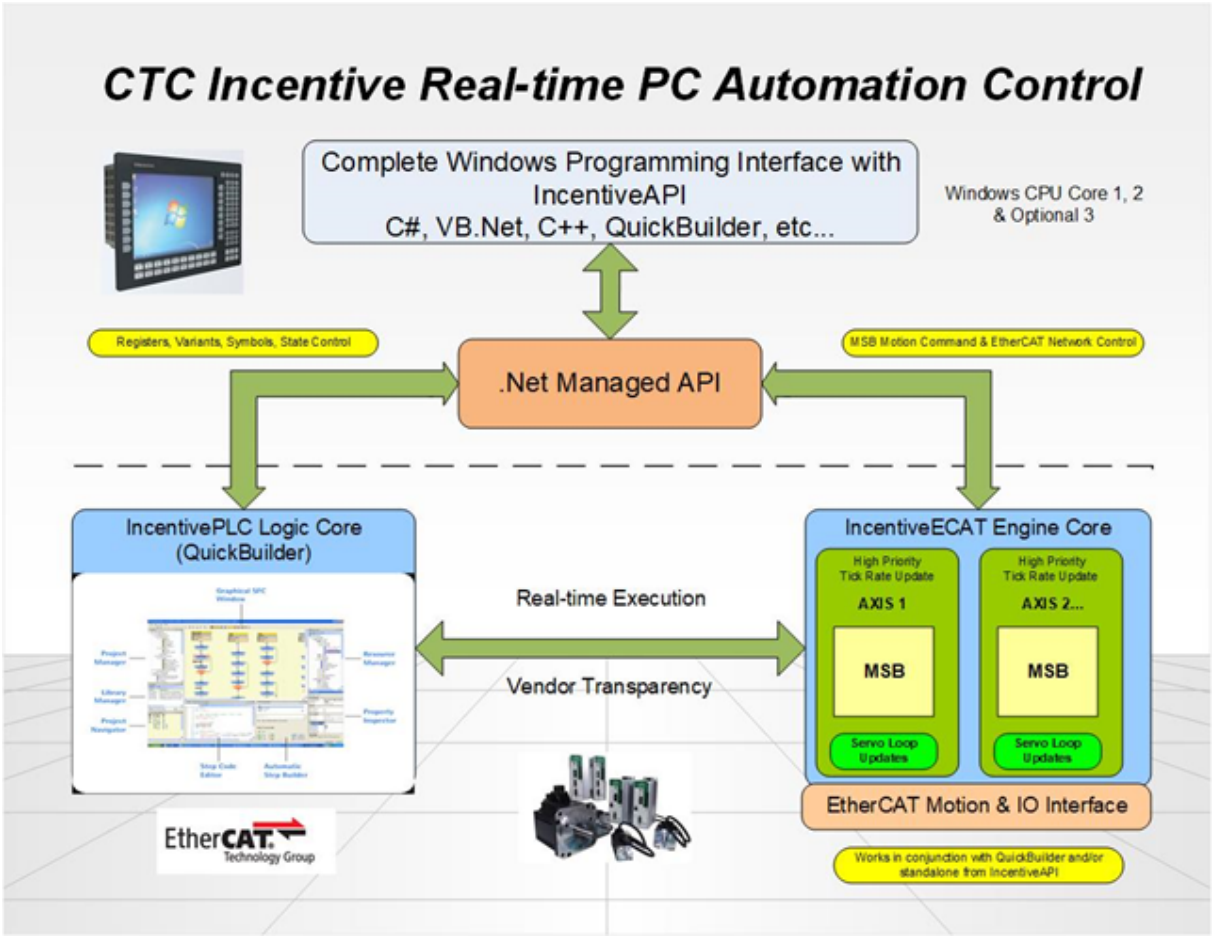
```
      stop axis1;       // stop execution of all MSBs on 'axis1', this
stops immediately

      stop axis1 slewed using 100; // stop execution of all MSBs on
'axis1', slewed stop at 100 user-units/sec/sec.
```

2.4.3 Motion Architecture Summary Diagram

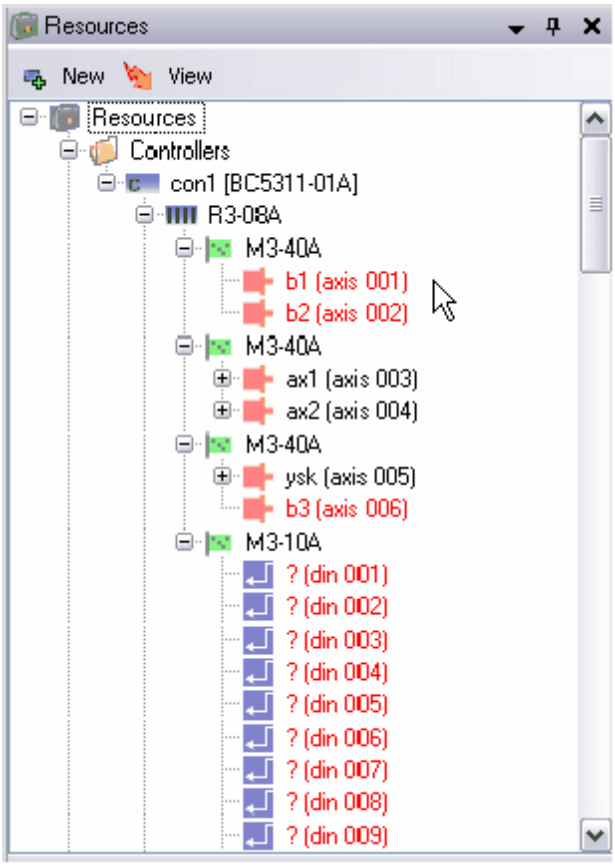


IncentivePLC & IncentiveECAT Soft PC



3 Chapter 3: QuickMotion Axis Setup

Adding a motion axis to a QuickBuilder project is very similar to adding any other resource. The first thing that needs to be done is to add the axis module to the appropriate rack in the controller. This is done by right clicking the rack and selecting the appropriate module. For this discussion we will be adding a third M3-40A module to our first 8-slot rack. As with other module types, axes are automatically numbered from left to right starting at the CPU. So in this case the two axes on the third module are numbered 5 and 6.

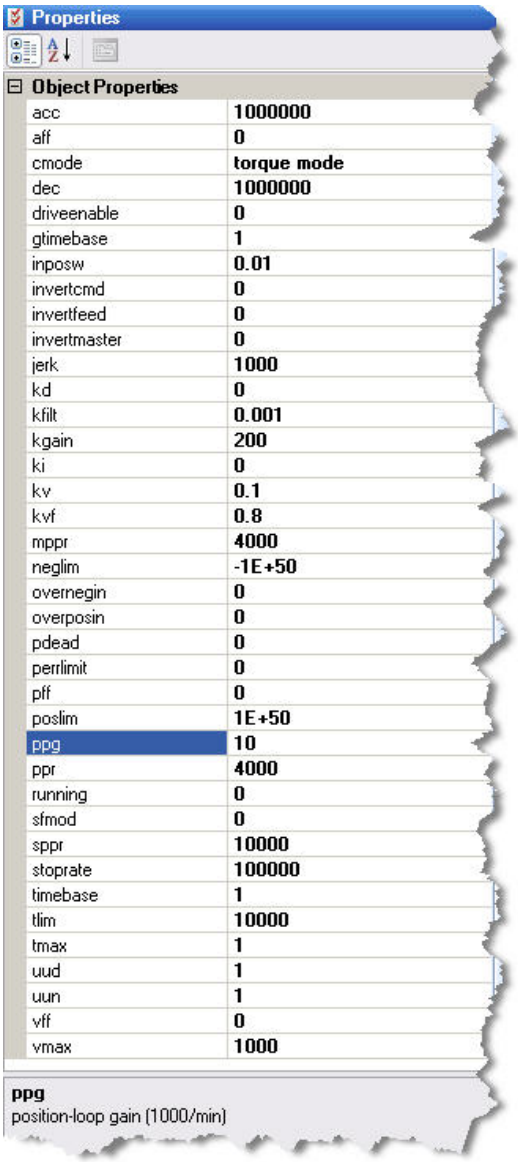


The axes first appear with question marks in their names, which must each be edited to a unique name. It is an error to have unnamed axes in a project. Right click and name the axis.

If the project changes, or the physical connection of the axes to the modules changes, axes can easily be rearranged in the Resource Manager. A single axis may be moved in the tree or a whole module can be moved as needed so that the named axes in the Resource Manager correspond to the actual wired axes.

After placing the *Axis* object in the proper place and naming it, the axis properties should be checked and updated as necessary. This is done in the property inspector window.

3.1 Axis Properties



When an *Axis* object is highlighted in QuickBuilder's Resource Manager, the following alphabetical property list for the axis is displayed. Required and Recommended properties to set up are reviewed below. Default values are given in [ ]. To learn more about these as well as the other properties, see the Variables Chapter later in this guide.

Required — When setting up an axis, the following properties must be set up in order for the Servo or Stepper Control module to properly interface with the connected motor and drive:

- **cmode:** Determines the command signal the controller sends out. Set to [Torque], Velocity, or Stepper.
- **tmax / vmax:** Depending on the drive type set, the maximum torque or velocity that will be realized by a 10V command from the controller. [1Nm / 1000RPM]
- **ppr:** The number of feedback counts per revolution [4000]
- **sppr:** When operating in stepper mode, this value must be set to correspond to the steps/rev of the controlled stepper drive.

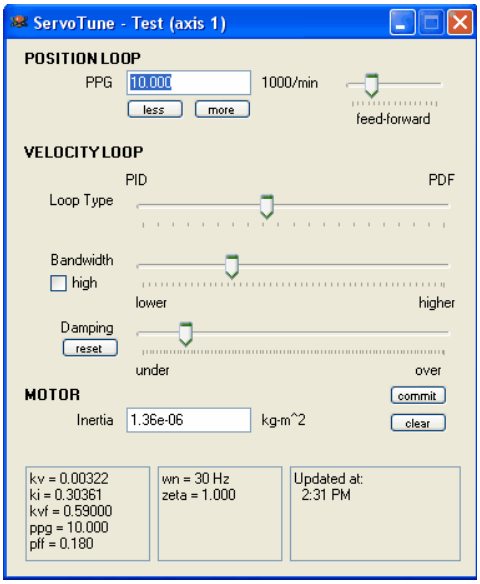
Recommended — Once the required properties have been entered, the axis can be tuned. However, it is recommended that the following properties also be checked and adjusted as necessary.

- **acc / dec:** Check that the acceleration / deceleration rates are appropriate. [10000000/10000000]
- **driveenable:** Set this to the output number that will be used to enable the servo drive. (Highly recommended that this be used. Use positive input number for true state=high; use negative number for true state=low.) [0=not used]

- **inposw:** The in-position window scaled in user units. This is used to determine when the drive has reached the commanded position. Use positive input number for true state=high; use negative number for true state=low. [0.01]
- **overnegin / overposin:** (Hardware over-travel limits) Set these to the input number to be used to signal positive and negative over-travel. Use positive input number for true state=high; use negative number for true state=low. [0=not used]
- **neglim / poslim:** (Software over-travel limits) Set these to the input number to be used to signal positive and negative overtravel. [-1E+50 / 1E+50]
- **perrlimit:** This is the maximum allowed following error in user units before a fault is generated. [0=disable checking]
- **uun/uud:** User-units numerator and denominator. This fraction is used to convert revolutions to user units. [1/1]

Other — Many of the other properties are either automatically adjusted by the tuning wizard or are used for more specialized functions. Refer to *Chapter 5: Variables* for more details.

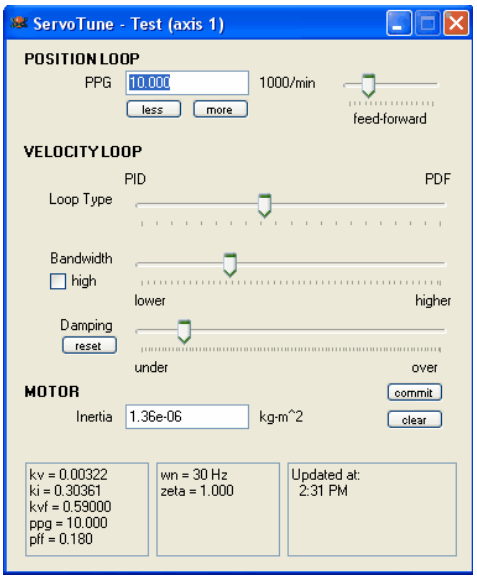
3.1.1 Basic Tuning



For basic tuning of an axis there is only one adjustment needed: adjust the Bandwidth slider until the desired performance is reached. Moving the slider to the right increases the servo loop bandwidth and hence the move performance. By checking the *high* box, the slider impact is doubled. If moved too far, the motor will become unstable and begin to emit a buzzing sound and vibration even with the motor at rest. If this occurs, move the slider back to the left until this condition is eliminated.

⚠ Note: Tuning parameters adjusted using the wizard are updated in volatile memory. To save them to the non-volatile memory of the controller it is necessary to download the project to the controller after tuning.

3.1.2 Fine Tuning



While the Basic Tuning method just discussed works well for most general purpose applications, higher performance applications or those with unusual loads or friction will typically require more adjustments. For best results in fine tuning an axis, it is useful to observe the velocity profile of the axis and how it responds to various adjustments to the tuning properties. This can be done by using QuickScope within QuickBuilder or by using an external oscilloscope to monitor the velocity output signal of the drive. The other wizard adjustment items are listed below:

PPG: This is the position loop proportional gain scaled in 1000/min units. This increases the response of the position loop and stiffness.

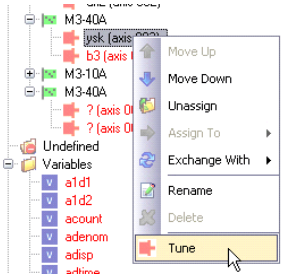
Feed-forward: This increases the position loop velocity feed-forward gain.

Loop Type: Adjust the loop type from 100% PID to 100% PDF structure.

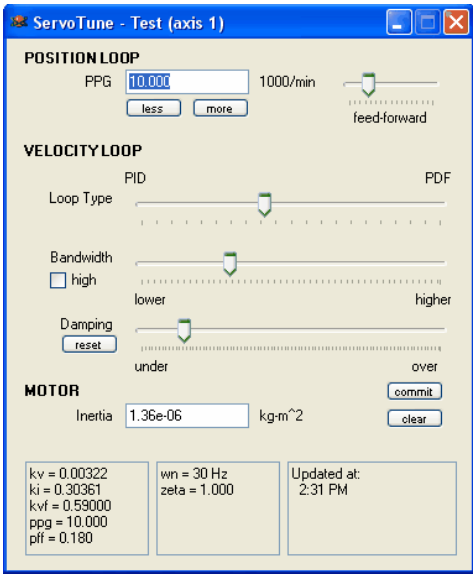
Damping: This has the effect similar to adding or removing friction from the system.

For advanced applications, all of these parameters with the exception of motor inertia can be changed programmatically or interactively through a QuickBuilder Watch Window. There are also several other tuning variables available for the experienced motion engineer. Refer to *Chapter 5: Variables* for details.

3.2 Tuning an axis (5300 M3-40 Only)

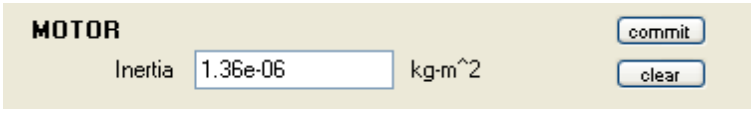


QuickBuilder simplifies the tuning process by utilization of an innovative new tuning wizard for each axis. To access the tuning wizard, simply right click on the axis and select *Tune*. Doing so will bring up a window like the one shown below. Note that each axis has its own tuning wizard window. Multiple windows may be active and displayed simultaneously.



To tune an axis with the wizard, the first step is to enter the motor inertia in the bottom box of the wizard. Once this is entered, the wizard is set up to critically damp the motor. Since the wizard adjusts tuning parameters in real time, the best way to use it is to set up a safe repeating move for the axis and then make adjustments in the wizard to optimize the motion profile.

Once tuning has been configured it may be save to the axis non-volatile memory but clicking on the 'commit' button. To remove tuning parameters from non-volatile motion board storage click the 'clear' button. By default the tuning parameters are saved with the QuickBuilder program and re-written each time the project is loaded. Committing the parameters to the motion board will override those in the program. This will set the nonvolatile axis variable to 1 when active.

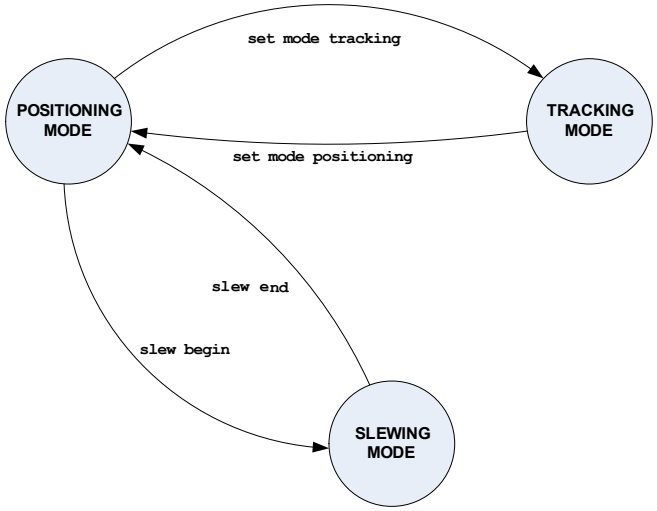


4 Chapter 4: QuickMotion Programming

This chapter covers the QuickMotion commands and their usage with MSBs.

⚠ These statements cannot be used in a QuickStep step! The MSB statement set has been created to simplify the motion programming process and make powerful motion control applications accessible to a wide range of users. These statements are optimized for high performance execution on motion modules. In addition to the motion statement set, CTC has provided over 100 pre-defined motion variables that greatly simplify development. The motion related variables are covered within their own chapter, later in this guide.

4.1 Operating Modes



Positioning

In this mode, the axis is able to perform absolute and incremental time-based motion, including *SegmentedMoves* and time-based CAMs.

The axis must have completed any pending positioning operations before changing to a different operating mode.

Slewing

In this mode, the axis generates a series of interpolated positions based upon a constant (but alterable) velocity.

The axis must be stopped by using slew end in order to perform any positioning operations.

Tracking

The axis is able to perform position-tracking in this mode. This includes following, gearing and position-based CAMs. The axis must complete all pending tracking operations before changing to a different operating mode.

4.2 Expressions

In QuickMotion, expressions consist of *variables*, *constants,* and *operators*. Variables are listed in *Chapter 6: Variables*.

The following *operators*, listed in order of grouped precedence, are available in QuickMotion:

```
( )     parenthesis
||      logical-or
&&      logical-and
|       bitwise-or
&       bitwise-and
!=      not-equal
==      equal
<=      less-than-or-equal
<       less-than
>       greater-than
>=      greater-than-or-equal
+       add
-       subtract
*       multiply
/       divide
%       modulo
!       logical-not
~       bitwise-not
-       negate
```

4.3 Utility Statements

Summary:

stop { slewed using *rate* **}**
drive enable
drive disable
delay *time* **ms**
variable = expression
zero feedback position
zero target position
zero following error
reset
if *condition* **then** *variable* **=** *expression*
wait until *condition*

Stop	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
stop { slewed using variable }				
parameters				
variable optional rate at which to stop the axis in user unit				

```
stop;              // stop the axis
stop slewed using rate;       // stop the axis by slewing to 0 at
specified rate
```

In positioning mode:

Non-slewed – This statement immediately aborts the present motion operation as well as halts the target position generator from updating the target position (*tpos*), thereby (eventually) stopping motion. This form of *stop* may not be desirable in all cases (such as when the axis has excessive following error), since the target position may be greatly different than the feedback position and the feedback position will still seek the target position.

Slewed – This statement first copies the current feedback position (*fpos*) into the target position (*tpos*) and then generates a controlled deceleration by *slewing* to zero velocity using the rate specified.

If the axis is in *slewing* mode, a *slew end* is issued thereby placing the axis in *positioning* mode. The optional stop mode *slewed* is ignored in this mode.

If the axis is in *tracking* mode, the numerator of the gear ratio is set to zero – but the axis remains in *tracking* mode. The optional stop mode *slewed* is ignored in this mode.

Enable/Disable Drive	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☐ FG MSB
syntax				
drive enable				
drive disable				

Enables or disables the drive associated with the axis, thereby allowing motion to occur. If the *driveenable* variable has been set to an output number, that output is automatically turned on when the drive enable command is encountered or turned off when the drive disable command is encountered. In some cases, the motor may slowly decelerate to a zero velocity when disabling.

```
drive enable;            // enable the drive for this axis
drive disable;           // disable the drive for this axis
```

⚠ Invoking the drive enable command sets the target position (*tpos*) to the feedback position (*fpos*).

Time Delay	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
delay *time* **ms**				
parameters				
`time`	an expression representing time in milliseconds			

This statement delays execution of the active MSB for the specified number of milliseconds.

```
delay 2500 ms;                  // delay for 2.5 seconds
```

Timeout Initialization	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set timeout *ticks*				
parameters				
`ticks`	Number of ticks until a timeout occurs causing any active 'on timeout' event handlers to take action.			

This command initializes a private msb timer which is decremented on every tick if the 'on timeout' command is active. To disable execute an 'on timeout ignore' command. The timeout value must be set after every timeout, it acts as a down counter, invoking the event handler when 0 is reached. .

```
set timeout 100;  // Set timeout to 100 ticks
```

Assignment	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`variable = expression`				
parameters				
`variable`	a variable to change the value of			
`expression`	an expression			

The value of the specified expression is evaluated and stored to the named variable.

```
//calculate a new value for result
result = 34.857 * oldresult;
```

Zero Feedback Position	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`zero feedback position`				

Zeros the target position, but maintains following error (fposc = fposc - (ppr * tpos) then tpos = 0). Operates the same as *zero target position*.

```
//set the current position as zero
zero feedback position;
```

Zero Target Position	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`zero target position`				

Zeros the target position, but maintains following error. Operates the same as *zero feedback position*, but is more readable in stepper mode.

```
//set the current position as zero
zero target position;
```

Zero Following Error	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`zero following error`				

This statement zeros the feedback position (fpos/fposc) and target position (tpos), thereby removing any following error.

```
// relax the system by zeroing the following error
zero following error;
```



Unless you are current limiting and driving into a hard stop (or similar application), there is no reason to use "zero following error" (and it's probably wrong in most applications to use it). Zero position feedback is what should normally be used. Remember that following error is maintained when zeroing the position

feedback and 99.99% of the time that is what is desired. Think of it like this:

```
tpos = 1.000

fpos = 0.999
```

After "zero feedback position":

```
tpos = 0.000

fpos = -0.001
```

You don't want to lose that 0.001 of error, but you still want to call wherever you are zero — that is generally the case. Because tpos (the target to seek) runs the show, that is what you want to be precisely zero. All motion is relative/absolute to the target position, NOT the feedback position, as that wouldn't make sense.

Zero following error is used, for example, in nut-driving applications where one limits the torque, drives to an unreachable position (because as the nut is torqued, the torque limit is hit), and then watches for current limit and then zeroes the following error — thus, removing the torque, etc.

Reset Faults	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
reset				

Resets any fault (if possible to).

```
reset;      // reset axis faults
```

If/Assignment	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				

if *condition* **then** *variable* = *expression*	
parameters	
condition	a Boolean test condition
variable	a variable
expression	an expression

This statement evaluates the specified *condition*. If *true*, the *expression* is evaluated and *variable* is set to the resulting value. If *false*, MSB program flow continues at the next MSB statement.

```
// if the position error for the axis exceeds
// 0.25 set a variable 'fault' to 2
if perr > .25 then fault = 2;
```

Wait Until	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
wait until *condition*				
parameters				
condition	condition to test			

This statement waits for until the specified *condition* is *true*.

```
// wait here until chamber temp exceeds min
wait until temp > 32.849;
```

4.4 Program Flow Statements

Summary:

[*label*]
start *MSB mode*
end { and start *MSB mode* **}**
abort *MSB*
goto *label*
if *condition* **goto** *label*
on *asynchevent asynchhandler*

Statement Label	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
[*label*]				

A label within an MSB is used as a marker for the destination of a *goto* or similar statement.

It is often required to *iterate* or *branch* depending on the state of some external input/output or internal condition – a label is used to mark the destination.

```
// this label is called Top
[Top]
```

Start	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
start *MSB mode*				
parameters				

MSB	the name of the MSB to start
mode	**FG** start as a high-priority (tick) MSB **BG** start as a low-priority (non-tick) MSB

This statement activates an MSB – if the MSB is already active, this statement is effectively ignored.

Up to 4 foreground (FG) MSBs may be running simultaneously.

There is no logical limit to the number of active background (BG) MSBs.

```
// start the MSB called PressCap and run as a foreground MSB
start PressCap FG;
```

End	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
end { and start *MSB mode* **}**				
parameters				
MSB	the name of the MSB to start			
mode	**FG** start as a high-priority (tick) MSB **BG** start as a low-priority (non-tick) MSB			

This statement ends execution of this MSB. An optional MSB can be specified to start after this one completes.

An *end* or *goto* statement should be the last statement in any MSB.

```
// this is the end of the MSB
end;

// end the current MSB and then start the MSB called WeldCap
// as a foreground MSB
end and start WeldCap FG;
```

Abort	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
abort *MSB*				
parameters				
MSB	the name of the MSB to abort (stop) the execution of			

This statement ends execution of another MSB. If the named MSB is not active, the statement is effectively ignored.

```
// kill only the WeldCap MSB
abort WeldCap;
```

Goto	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
goto *label*				
parameters				
label	the name of the label to branch to			

This statement changes program flow to the specified label.

```
// jump to the MSB label called Top
goto Top;
```

If/Goto	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
if *condition* **goto** *label*				
parameters				
condition	a Boolean test condition			
label	the name of the label to branch to			

This statement evaluates the specified *condition*. If *true*, MSB program flow continues at the specified *label*. If *false*, MSB program flow continues at the next MSB statement.

```
// If the axis's input1 is on goto the label MakeMove
if din1 goto MakeMove;
```

Asynchronous Event Handling	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
on *asynchevent asynchhandler*				
parameters				
asynchevent	One of the following: **riseof** *n* Rise of specified general purpose input. **fallof** *n* Fall of specified general purpose input. **hardfault** When a non-recoverable fault occurs. **capture** Capture of specified input trigger. **pls** *output* PLS output 1 to 5 activated. **timeout** 'timerticks' variable decrements to 0 (use 'set timeout' to initialize msb private value).			
asynchhandler	One of the following: **ignore** Cancel asynchronous event monitoring. **start** *MSB {FG/BG}{arm}* Starts the specified MSB in BG (background) mode unless FG is			

	specified, if capture then optional {arm} at end of statement. **goto** *label {arm}* Branch on event, if capture then optional {arm} at end of statement.

This statement controls asynchronous event handling.

If *asynchhandler* is set to *start…*, then an MSB is started automatically when the specified event occurs. If the *MSB* is already active when the event occurs, a second instance is *not* started. If not specified background mode is used (BG).

If *asynchhandler* is set to goto…, then a branch to that label occurs upon the event, within the same MSB.

If *asynchhandler* is set to cancel, then no operation will occur upon event. Each event is unique to a specific MSB although only one MSB may monitor a capture or specific pls output event.

If *asynchevent* is set to timeout then the 'set timeout <ticks>' command must be set for down counting to begin (500uS/tick). Branching based upon a timeout will occur regardless of motor operations and it is up to the MSB to properly recover and/or stop motors.

Example 'on timeout':

```
x = 0;
y = 0;
set timeout 5000 * 2;          // 5 second timeout
on timeout goto timedout;
[top]
// x will increment for 5 seconds and then a branch to [timedout] will occur
x = x + 1;
delay 100 ms;
goto top;
[timedout]
// y will increment after 5 seconds and continue forever
y = y+1;
delay 100 ms;
goto timedout;
```

4.5 Set Statements

Summary:

set common bit *number state*
set common var *number value*
set loopperiod *value*
set mode positioning
set mode tracking
set timeout *ticks*
set target position *value*
set feedback position *value*
set target position counts *vcounts*
set feedback position counts *vcounts*
set simulated feedback on/off
offset position *value*
offset position counts *vcounts*
set master *mode* { **using** *global* }

Set Loop Period	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set loopperiod *value*				
parameters				
`value`	The desired loop time in uS, default value is .0008 (800uS). The minimum is 500uS.			

This statement sets motion interrupt loop period. The current loop period and rate are available via the axis 'loopperiod' and 'looprate' variables (looptime group). The period selected should be evenly divisible for accuracy. Thus .0005 has a rate of 2000 ticks/second, .0008 is 1250 ticks/second (1/.008). Setting one axis sets the other and it is recommended to only change the loop time at initialization, prior to the 'drive enable' command.

```
set loopperiod 800;      // Set loopperiod to the default, 800 uS
                         // (not needed since powerup default
```

Set Positioning Mode	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set mode positioning				

Sets the operating mode of the axis to *positioning*.

```
set mode positioning;    // switch to positioning mode
```

Set Tracking Mode	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set mode tracking				

Sets the operating mode of the axis to *tracking*.

```
set mode tracking;       // switch to tracking mode
```

Set/Offset Target/Feedback Position(s)	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set target position value **set feedback position** value **set target position counts** vcounts **set feedback position counts** vcounts **offset position** value **offset position counts** vcounts				
parameters				
value	new or offset for the named position (user-units)			
vcounts	new or offset for the named position (counts)			

These statements modify the target and/or feedback positions. The new value (or offset) may be specified in user-units, or in feedback counts (by use of the keyword counts). The first two forms set the target or feedback position to a specific absolute value in user-units. The third and fourth forms set the target or feedback position to a specific absolute value in counts. The last two forms modify the target and feedback positions simultaneously by adding the specified offset to both.

⚠ Following error is maintained when these statements are executed.

⚠ The axis must not be active (i.e. actively generating a target position by use of a move statement) when any of these statements are executed.

```
// set the feedback position (fpos) to 2.149
set feedback position 2.149;

// offset both the target and feedback positions by 1100 counts
offset position counts 1100;
```

Set simulated feedback	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				

set simulated feedback *on/off*	
parameters	
on/off	off - normal operation, feedback from encoder. on - feedback simulated and from tposc on each servo loop.

Enables or disables simulated feedback, setting fposc to originate from the encoder (off) or tposc (on). 'tposc' is the incremental amount to move on the next servo loop. Thus when simulated the desired increment will be achieved on each loop. This command is useful for both test purposes and when using a virtual master. The simulated axis can publish its master position across the controller backplane, based upon its moves. See the 'Virtual Master' section. This command is also useful during open loop stepper operation when using the pls functionality.

```
set simulated feedback on; // this will cause fposc to = tposc after each loop period, drive must not be enabled
```

Set Master Encoder Source	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set master *mode* { **using** *global* }				
parameters				
mode	**feedback1**	master position sourced from axis 1 feedback		
	feedback2	master position sourced from axis 2 feedback		
	target1	master position sourced from axis 1 target		
	target2	master position sourced from axis 2 target		
	feedbackZ	master position comes from axis 1&2 Z-inputs		
	virtual	master position on this axis is to be calculated as a virtual source, reference 'move master at' for setup (master axis).		
	common	master position from controller backplane as determined by variant register 36827 (slave axis)		
global	**global**	(optional) This position information is made public to the controller backplane. Distributed to 'common' nodes as determined by variant register 36827.		

This statement sets the source of the axis master encoder. The default source for MSBs executing on the first axis is *feedback2*. This means the first axis is using the second axis as the master. This command executes independently on each access thus to change axis 1 to be the master a 'set master feedback1' must be executed by MSB's on both axis. The default for the second axis is *feedback1*.

The source *feedbackZ* is derived by using the first axis' Z-channel input as the "A"-channel for the master encoder and the second axis' Z-channel input as the "B"-channel for the master encoder.

For an axis to make its master public the 'using global' option is used. This allows the axis to publish its position information to other axis that executes the 'set master common' command.

Variant register 36827 is used to define how global master information is distributed amongst slaves. The variant is a 4 row, 3 column array with the first 4 rows defining possible global master sources to reference and the columns referenced as follows:

[0] – enabled position information updates (every 4 mS to all slaves), set 1 to enable, 0 to disable.

[1] – master axis whose position information is to be distributed to slaves, 1 to N where N is all the axis in a controller rack. Note that the master axis MSB must have executed the 'set master global' command.

[2] – 32 bit field with each bit representing a slave axis to whom the master axis information is to be distributed. Bit 0 would be axis 1, Bit 31 is axis 32.

4.6 Common bits and variables

Summary:

set common bit *number state*
wait common bit *number state*
set common var *number value*
wait common var *number range*

Common bits and common vars are used to communicate state information:

a. between QuickMotion based modules
b. between QuickMotion and QuickStep 4
c. between axes on a single module such as an M3-40A

There are 256 common bits and 256 common vars. Common bits are Boolean, and common vars are bytes and therefore have values from 0 through 255.

The common bits are globally shared between all QuickMotion modules as well as QuickStep 4. Any changes made to common bits are "seen" by all QuickMotion modules and the main CPU running QuickStep 4.

The first 32 common vars are *overlaid* on top of the 256 common bits – changes made to a common var may alter up to 8 common bits.

The remaining 244 common vars are *module-local* – changes are only seen local to the module. This is useful to communicate state information between axes on a two-axis QuickMotion module such as an M3-40A.

A user may decide whether to use just common bits or just common vars or even a combination of the two depending on the application.

From QS4, common bits are accessed via the $CBITS[ ] system variable and common variables are accessed via the $CVARS[ ] system variable.

There are several QuickMotion instructions that deal with common bits and vars:

- *set common bit*
- *wait common bit*
- *set common var*
- *wait common var*

Within QuickMotion, common bits and vars can be used in expressions through the notation:

`cbit[n]` where n is 0 through 255

`cvar[n]` where n is 0 through 255

For example:

```
[top]
if cbit[10] goto op10;
if cbit[11] goto op11;
if cbit[12] goto op12;
goto top;

[op10]
move to 1.0;
wait for in position;
goto top;

[op11]
move to 0.0;
wait for in position;
goto top;

[op12]
move to 2.25;
wait for in position;
goto top;
```

Set Common Bit	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set common bit *number state*				
parameters				
`number`	bit number (0-255)			
`state`	**true** or **false**			

This statement sets the specified “common bit” to the given state.

Wait For Common Bit	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`wait common bit` *`number state`*				
parameters				
`number`	bit number (0-255)			
`state`	**`true`** or **`false`**			

This statement waits until the specified “common bit” is at the desired state.

Set Common Var	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`set common var` *`number value`*				
parameters				
`number`	variable number (0-255)			
`value`	an integer value (0-255)			

This statement sets the specified “common state variable” to the given value.

Wait For Common Var	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`wait common var` *`number range`*				
parameters				
`number`	variable number (0-255)			
`range`	`x` a value of *`x`* `x-y` a value of *`x`* through *`y`* inclusive `! x` a value other than *`x`* `! x-y` a value outside the range of *`x`* through *`y`*			

This statement waits until the specified “common state variable” is within/outside the given range.

4.7 I/O Statements

Summary:

setout *outputlist*
clrout *outputlist*
pulse *output* **for** *n*
pls *output* **using** *reference definitions*
pls *output state*
wait for[****]*transition*[****]**of**[****]*input* { **or**[****]*condition* }
generate output *output* **rate** *freq*
generate *n* **steps on** *pair*
variable = **ctr[n]**
ctr[n] = *expression*
ctr[n] = *offset*
generate alternate *mode*

Set Output(s)	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
setout *outputlist*				
parameters				
outputlist	a comma delimited list of outputs to set			

This statement sets one or more outputs to the *on* state.

The output number can be 1-5 (dual axis mode) or 1-10 (1½ axis mode).

```
setout 2;          // turns on the second output on the module
setout 1, 3;       // turns on the first and third outputs
```

Clear Output(s)	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
clrout *outputlist*				
parameters				
outputlist	a comma delimited list of outputs to clear			

This statement sets one or more outputs to the *off* state.

The output number can be 1-5 (dual axis mode) or 1-10 (1½ axis mode).

```
clrout 2;          // turns off the second output on the module
```

```
clrout 1, 3;      // turns off the first and third outputs
```

Pulse Output	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
pulse *output* **for** *n*				
parameters				
output	the output to pulse 1-5 (dual axis mode) 1-10 (1½ axis mode)			
n	the time to pulse the output, an expression as milliseconds			

This statement causes the specified *output* to pulse for the specified duration. If the output is already *on* when this statement executes, the output state is unchanged – however it will be turned *off* after the specified time.

If another statement changes the state of the output to off before the allotted duration, the generation of the pulse is aborted.

The generated pulse is accurate within ½ of a millisecond.

```
// turns on the 2nd output on the module for 500ms
pulse 2 for 500;
```

PLS Define	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
pls *output* **using** *reference definitions*				
parameters				
output	the output (1-5) to control via a PLS			
reference	the encoder count scaled reference variable to compare to: **fposc** Feedback position of axis msb **mposc1 - mposc5** Master position counters #1 through #5 **mposc** Master position counter **smodc** Slave position (modulo) **smark** Slave marked position **tmc1 tmc2** Temporary master counters #1 & #2 **tsc1 tsc2** Temporary slave counters #1 & #2 **sdc** Slave decrement counter **fposc1** Feedback position of axis 1 (fposcA)			

	fposc2 Feedback position of axis 2 (fposcB) **tmodc** Temporary master counter mod mmc **sfposc** Secondary feedback position of axis
definitions	a comma-separated list of up to 16 PLS definitions: **on** x **to** y Turn output on when the reference is within the bounds specified by x through y (may be expressions)

The first statement defines or redefines a PLS (software-based programmable limit switch) associated with a given output. A definition over-writes the previous definition for an output (if one was defined already).

⚠ When a PLS is defined/re-defined it will be disabled and will not compute the state for the output. To enable a PLS after it is defined/re-defined, a *pls on* statement must be issued:

```
// define a PLS for output #1
//   output will be on when fposc is within 10-200 or 400-430
pls 1 using fposc on 10 to 200, on 400 to 430;

// enable the PLS for output #1
pls 1 on;
```

⚠ When using open loop stepper tposc is not available for PLS thus issue the command 'set simulated feedback on' to have tposc copied to fposc, on each control loop, allowing the use of this command.

PLS Enable/Disable	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
pls *output state*				
parameters				
output	the output to control via a PLS 1-5 (dual axis mode) 1-10 (1½ axis mode)			
state	**on** or **off**			

This statement enables ("on") or disables ("off") a PLS for an output.

On - Enables the pls functionality initialized for a particular output with the PLS Define statement.

Off – Disables the pls functionality initialized for a particular output with the PLS Define statement.

⚠ If the output is on when a PLS is disabled, it will remain on – unless the user re-enables the PLS (to re-compute the PLS output), or they *clrout* the output.

Wait For Input	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`wait for` *`transition`* **`of`** *`input`* { **`or`** *`condition`* }				
parameters				
`transition`	**`rise`** or **`fall`**			
`input`	the general purpose input to wait upon 1-5 (dual axis mode) 1-10 (1½ axis mode)			
`condition`	an optional exit condition			

This statement waits for the specified *transition* of the specified general purpose *input* to occur.

The MSB will not continue execution until the transition occurs – unless there was a *condition* specified and the condition evaluated to *true*.

```
// delay execution of MSB until input1 transitions from off to on
wait for rise of 1;
```

⚠ When this statement is used with the optional exit condition and the statement is part of a BG MSB, it is possible to miss transitions of the general purpose input. Therefore, the optional exit condition form should be used with care in BG MSBs.

Generate Pulses	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`generate output` *`output`* **`rate`** *`freq`*				
parameters				
`output`	1-10			
`freq`	the frequency (in Hz) to generate pulses; rounded to an integer			

This statement begins or ends generation of pulses using a specific output. If pulses are being generated on an output, then *setout*, *clrout* and *pulse output* commands given to the same output have the following behavior:

setout	no pulse generation occurs; the output will be active
clrout	pulse generation occurs for non-zero *freq*s
pulse output	no pulse generation occurs until the pulse output completes

When a frequency of 0 is specified, no pulse generation occurs. This effectively turns the output back into a general-purpose output.

The minimum frequency that can be generated is 1 Hz. The *maximum* frequency that can be generated is well over 500 kHz.

The accuracy of the generated signal varies by frequency (lower frequencies are more accurate). The following table summarizes the accuracy for several frequencies:

<100 Hz	+/- 0.001 Hz
500 Hz	+/- 0.005 Hz
1 kHz	+/- 0.02 Hz
2 kHz	+/- 0.1 Hz
5 kHz	+/- 0.5 Hz
10 kHz	+/- 2 Hz
20 kHz	+/- 8 Hz
50 kHz	+/- 50 Hz
100 kHz	+/- 200 Hz
250 kHz	+/- 1.5 kHz
500 kHz	+/- 5 kHz

⚠ Due to a hardware limitation, this statement is only usable with outputs 3, 4, 5 (Axis 1) and outputs 3, 4, 5 (Axis 2). The use of outputs other than those listed will be ignored.

⚠ The number of generated pulses cannot be controlled – only the frequency of the generated pulses.

Generate Steps	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`generate` *`n`* **`steps on`** *`pair`*				
parameters				
`n`	the number of steps to generate in 500 µsec			
`pair`	the step/direction pair to output steps on:			

	1. axis 1 step/direction pair (M3-40A/B/C outputs 3&4) 2. axis 2 step/direction pair (M3-40A/B/C outputs 3&4) 3. alternate step/direction pair (outputs 5 on each axis)

This statement generates step and direction pulses on the specified step and direction pair.

If the expression *n* evaluates to a negative number, then the direction will be negative.

All of the pulses will be emitted in the next 500µs loop period.

⚠ Any *setout*, *pulse* or *generate output* used in parallel with this command will cause erroneous step/dir pulses to be emitted. One should not use these commands in conjunction with *generate steps*.

⚠ This command when used with *cmode* set to *stepper* mode will command additional pulses out the step/dir outputs.

Counter read, write, offset	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				

`variable = ctr[n]`
`ctr[n] = expression`
`ctr[n] = offset`				

parameters				
`n`	the counter number (0 through 7)			
`variable`	the variable to store the current value of the counter to			
`expression`	a new value for the counter			
`offset`	an offset for the counter (subtracted from the current counter value)			

These specialized forms of the *assignment* statement give read/write/offset access to the axis counters.

On the M3-40A, -40B, and -40C, there are 8 counters/axis that accumulate off-to-on transitions of the following:

ctr[0] digital input 1
ctr[1] digital input 2
ctr[2] digital input 3
ctr[3] digital input 4
ctr[4] digital input 5
ctr[5] 'A' channel input (non-quadrature)
ctr[6] 'B' channel input (non-quadrature)
ctr[7] 'Z' channel input (non-quadrature)

The first form of the statement stores the current counter value in a variable.

The second form of the statement changes the current counter value.

The third form of the statement offsets the current counter value.

The first and third forms are often used together:

```
totalcounts = 0;
[top]
// wait until input #1 rises
wait for rise of 1;
// get the current counter value
x = ctr[7];
// accumulate
totalcounts = totalcounts + x;
// offset so no counts are missed
ctr[7] -= x;
goto top;
```

Alternate Stepper Output	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
generate alternate *mode*				
parameters				
mode	**on** generate stepper outputs on alternate pins **off** generate stepper outputs on standard pins			

On a M3-40A/B/C, (and when in stepper mode in the case of the M3-40A), the step and direction outputs are normally output on axis (TBx) pin pairs (15, 16).

These cards also allow a third-axis to be controlled by temporarily outputting step and direction pulses on TB1 pin 22 (step) and TB2 pin 22 (direction).

To output on this alternate pair, the command *generate alternate on* should be issued. To output on the standard pair, the command *generate alternate off* should be issued.

⚠ One needs to be careful as only the destination of the step and direction signals change – the axis still believes that motion is being commanded on the primary axis (and thus updates its idea of where the absolute stepper position is). Therefore, it is good practice to zero the target position (*zero target position*) before switching to or from this alternate mode:

```
// move my axis 30 revs
zero target position;
generate alternate off;
move at 5 for 30 using 10,10;
wait for in position;

// move axis #3 20 revs
zero target position;
generate alternate on;
move at 10 for 20 using 10,10;
wait for in position;
```

```
// move me again 10 revs
zero target position;
generate alternate off;
move at 5 for 10 using 10,10;
wait for in position;
```

4.8 Simple Motion

Summary:

move to *position* **{ using** *acc, dec* **}**
move at *maxvelocity* **to** *position* **{ using** *acc, dec* **}**
move trap to *position* **using** *rate*
move in *time* **to** *position* {**mode** *n* }
move for *displacement* **{ using** *acc, dec* **}**
move at *maxvelocity* **for** *displacement* **{ using** *acc, dec* **}**
move trap for *displacement* **using** *rate*
move in time for displacement {mode n }
wait for in position
new endposition *position* **using** *rate*
new endposition relative *displacement* **using** *rate*
slew begin
slew at *velocity* **in** *time*
slew for *displacement*
slew end

Move Absolute, Triangular	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
move to *position* **{ using** *acc, dec* **}**				
parameters				
position	absolute end position, user-units			
acc	acceleration rate, user-units/sec/sec			
dec	deceleration rate, user-units/sec/sec			

This statement generates a triangular move to the specified end *position*. If the parameters *acc* and *dec* are omitted, then the default rates are used.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

Note: The specified *position* may also be specified as **ZPULSE_POS** or **ZPULSE_NEG**, meaning the next encoder Z-pulse in the positive or negative directions, respectively.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should only be used with absolute move commands.

```
/* Move to the absolute position specified by the variable
drillpos using default acceleration and deceleration rates. */

move to drillpos;
```

```
/* Move in the positive direction to the Z pulse using default
acceleration and deceleration rates. */

move to ZPULSE_POS;
```

Move Absolute, Speed-limited	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
move at *maxvelocity* **to** *position* **{ using** *acc*, *dec* **}**				
parameters				
`maxvelocity`	unsigned maximum velocity, user-units/sec			
`position`	absolute end position, user-units			
`acc`	acceleration rate, user-units/sec/sec			
`dec`	deceleration rate, user-units/sec/sec			

This statement generates a trapezoidal move to the specified end *position*. If it is not possible to reach the specified maximum velocity *maxvelocity*, then a triangular move is generated. If the parameters *acc* and *dec* are omitted, then the default rates are used.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

Note: The specified *position* may also be specified as **ZPULSE_POS** or **ZPULSE_NEG**, meaning the next encoder Z-pulse in the positive or negative directions, respectively.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should only be used with absolute move commands.

```
/* Move to the absolute position specified by the variable
drillpos using default acceleration and deceleration rates and
the rapidrate variable for a max velocity. */

move at rapidrate to drillpos;
```

Move Absolute, Trapezoidal	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
move trap to *position* **using** *rate*				
parameters				
`position`	absolute end position, user-units			
`rate`	acceleration/deceleration rate, user-units/sec/sec			

This statement generates a 1/3-1/3-1/3 trapezoidal move (1/3 of the time accelerating, 1/3 constant velocity, 1/3 decelerating) to the specified end *position*. The acceleration and deceleration rate must be specified.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

Note: The specified *position* may also be specified as **ZPULSE_POS** or **ZPULSE_NEG**, meaning the next encoder Z-pulse in the positive or negative directions, respectively.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should only be used with absolute move commands.

```
/* Move to the absolute position specified by the variable
drillpos using the variable rapidacc to set acceleration and
deceleration rates. The velocity used will be based on
the calculation to achieve a trap move. */

move trap to drillpos using rapidacc;
```

Move Absolute, Time-limited	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
move in *time* **to** *position* {**mode** *n* }				
parameters				
`time`	time, sec			
`position`	absolute end position, user-units			
`n`	acc / dec ramp multiplier			

This statement generates a 1/3-1/3-1/3 trapezoidal move to the specified end *position* in the specified *time*. The optional *mode* feature decreases the amount of time spent on acceleration and deceleration. The *n* parameter must be a positive, non-zero integer. By increasing the value of *n*, the acceleration and deceleration times are equally reduced, allowing more time at constant speed.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

Note: The specified *position* may also be specified as **ZPULSE_POS** or **ZPULSE_NEG**, meaning the next encoder Z-pulse in the positive or negative directions, respectively.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should only be used with absolute move commands.

```
/* Move to the absolute position specified by the variable
drillpos setting the calculated velocity, accel and decel rates
to make a trapezoidal move in the time specified by the variable
movetime. */
```

```
move in movetime to drillpos;
```

Move Incremental, Triangular	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB

syntax

```
move for displacement { using acc, dec }
```

parameters

displacement	incremental position, user-units
acc	acceleration rate, user-units/sec/sec
dec	deceleration rate, user-units/sec/sec

This statement generates a triangular move for a specified *displacement*. If the parameters *acc* and *dec* are omitted, then the default rates are used.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should not be used with incremental move commands.

```
/* Move an incremental distance specified by the variable
spanmove using default acceleration and deceleration rates */

move for spanmove;
```

Move Incremental, Speed-limited	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB

syntax

```
move at maxvelocity for displacement { using acc, dec }
```

parameters

maxvelocity	unsigned maximum velocity, user-units/sec
displacement	incremental position, user-units
acc	acceleration rate, user-units/sec/sec
dec	deceleration rate, user-units/sec/sec

This statement generates a trapezoidal move for a specified *displacement*. If it is not possible to reach the specified maximum velocity *maxvelocity*, then a triangular move is generated. If the parameters *acc* and *dec* are omitted, then the default rates are used.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should not be used with incremental move commands.

```
/* Move an incremental distance specified by the variable
spanmove using default acceleration and deceleration rates and
using the variable slowspeed as a max velocity. */

move at slowspeed for spanmove;
```

Move Incremental, Trapezoidal	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
move trap for *displacement* **using** *rate*				
parameters				
displacement	incremental position, user-units			
rate	acceleration/deceleration rate, user-units/sec/sec			

This statement generates a 1/3-1/3-1/3 trapezoidal move (1/3 of the time accelerating, 1/3 constant velocity, 1/3 decelerating) for a specified *displacement*. The acceleration and deceleration *rate* must be specified.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should not be used with incremental move commands.

```
/* Move the incremental distance specified by the variable offset
using the variable rapidacc to set acceleration and deceleration
rates. The velocity used will be based on the calculation to
achieve a trap move. */

move trap for offset using rapidacc;
```

Move Incremental, Time-limited	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
move in *time* **for** *displacement* {**mode** n }				
parameters				

time	time, sec
displacement	incremental position, user-units
n	acc / dec ramp multiplier

This statement generates a 1/3-1/3-1/3 trapezoidal move for a specified *displacement* in the specified *time*. The optional *mode* feature decreases the amount of time spent on acceleration and deceleration. The *n* parameter must be a positive, non-zero integer. By increasing the value of *n*, the acceleration and deceleration times are equally reduced, allowing more time at constant speed.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should not be used with incremental move commands.

```
/* Move the incremental distance specified by the variable offset
setting the calculated velocity, accel and decel rates to make a
trapezoidal move in the time specified by the variable
movetime. */

move in movetime for offset;
```

Wait For In Position	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
wait for in position				

This statement temporarily stops the execution of the active MSB until the target generator has reached its final value and the position error, perr is within the programmed in-position window.

```
// Move speed-limited
move at slowspeed for spanmove;

// Wait till motor is within the programmed in-position window
wait for in position;

// Turn on output 1 for 1 second
pulse 1 for 1000 ms;
```

Set New End Position	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
new endposition *position* **using** *rate* **new endposition relative** *displacement* **using** *rate*				

This statement modifies the end position for the active *move* command. If there is no active *move*, then this statement is effectively ignored. The first form of this statement changes the end position to a new *absolute* position. The second form of this statement changes the end position *relative to the current position*. Using a *displacement* of 0 effectively stops motion *here* without generating a fault (unlike the *stop* command). Both statements require a *rate* to be specified. This *rate* is used as the acceleration/deceleration rate for the modified profile.

The *newvel* variable may be set to a nonzero value in order to specify a velocity. A trapezoidal move will be done whenever possible, if the end position does not allow for that then a triangular move will result. S-curve is not supported when using *newvel* although you may start out with an S-curve move and it will change to a trapezoidal or triangular with the new target and if *newvel* is nonzero, velocity.

⚠ Linear acceleration and deceleration is used (as programmed in the axis *acc* and *dec* properties) unless the property *jerk_a_req/jerk_d_req* is set to a non-zero value in which case an S-curve type profile is generated.

⚠ **ZPULSE_POS** or **ZPULSE_NEG** should not be used with this motion command.

Example 1: After din1 is activated change the end position to -3 mm.

```
/* This example demonstrates how a move can be modified
on-the-fly by using the new endposition command */

[top]
zero feedback position;

// start moving to 25 mm
move at 5 to 25;

// if din1 is activated during the move, change the end
// position of the move to -3 mm
wait for rise of 1;
new endposition -3 using 10;

wait for in position;
delay 3000;
goto top;
```

Example 2: The move will be terminated 3mm after din1 is activated. Speed is only changed when it is time to decel to the new end position.

```
/* This example demonstrates how a move can be modified
on-the-fly by using the new endposition command.  */

[top]
zero feedback position;

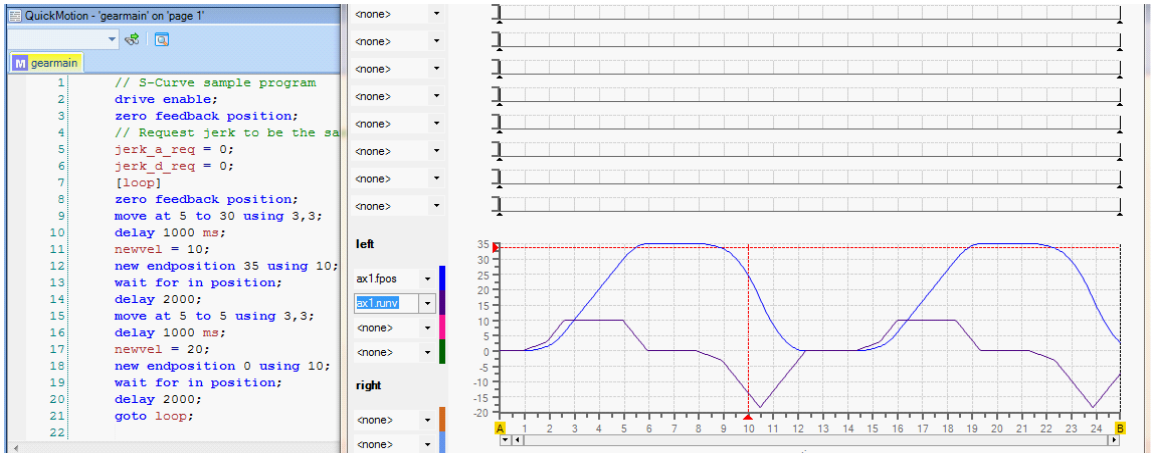
move at 5 to 25;
wait for rise of 1;

new endposition relative 3 using 10;
wait for in position;

delay 3000;
```

```
goto top;
```

Example 3: Change target from 30 to 35, acceleration from 3 to 10 and velocity from 5 to 10 during the acceleration phase of the move.



Slew (begin)	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB

syntax

slew begin

This statement changes the operating mode of the axis to *slewing*.

```
slew begin;        // change from position mode to slew mode
```

Slew At	☐ Positioning	☑ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB

syntax

slew at *velocity* **in** *time*

parameters

velocity	new slew velocity, user-units/sec
time	time, sec

This statement alters the current slew *velocity*. The velocity is changed smoothly over the specified *time*. For an immediate speed change, specify 0.0 for *time*.

```
// change from position mode to slew mode
slew begin;
```

```
// change from current speed to feedrate in 0.5 seconds
slew at feedrate in 0.5;
```

Slew For	☐ Positioning	☑ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
slew for *displacement*				
parameters				
displacement	ending relative slew position, user-units			

This statement alters the current slew velocity over time (to a slew velocity of 0.0) such that some *displacement* is consumed. If the current slew velocity is 0.0, then this statement is ignored.

The *displacement* should be unsigned, as the sign of the current slew velocity is used to sign the *displacement*.

```
// change from position mode to slew mode
slew begin;

// change from current speed to feedrate in 2 seconds
slew at feedrate in 2;

// delay execution of MSB until input3 transitions from off to on
wait for rise of 3;

// slew to a stop in the distance specified by the variable
// registrationoffset
slew for registrationoffset;
```

Slew (end)	☐ Positioning	☑ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
slew end				

This statement changes the operating mode of the axis to positioning. A zero-speed slew (in 0.0 *time*) is first generated if the axis is currently slewing at a non-zero velocity.

```
// change from position mode to slew mode
slew begin;

// change from current speed to slowjog in 0.5 seconds
slew at slowjog in 0.5;

// delay execution of MSB until input1 transitions from on to off
wait for fall of 1;
```

```
// stop motion and return to position mode
slew end;
```

4.9 Gearing

Summary:

gear at *numerator : denominator*
gear at *numerator : denominator* ***in*** *counts*
gear at *numerator : denominator* **in** *counts* **after** *acounts*
gear for *slavecounts* **in** *mastercounts*
gear for *slavecounts* **in** *mastercounts* **after** *acounts*
offset slave by *slavecounts* **in** *time*
wait master *counts*
wait slave *counts*
wait *source* **within** *start , end*
wait *source* **outside** *start , end*
zero *masslv* **counters**

Gear At	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
gear at *numerator* : *denominator*				
parameters				
`numerator`	new gear ratio numerator			
`denominator`	new gear ratio denominator			

This statement *instantaneously* changes the gear ratio of the slaved axis to the specified values.

Gear At In	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
gear at *numerator* : *denominator* **in** *counts*				
gear at *numerator* : *denominator* **in** *counts* **after** *acounts*				
parameters				
`numerator`	new gear ratio numerator			
`denominator`	new gear ratio denominator			
`counts`	counts of the master encoder			
`acounts`	counts of the master encoder to "wait for" before applying the gear/at...in...			

This statement changes the gear ratio of the slaved axis to the specified values over some number of master *counts*. An optional *after* condition can be applied to delay applciation of the gear/at/in.

Gear For In	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`gear for` *`slavecounts`* **`in`** *`mastercounts`*				
`gear for` *`slavecounts`* **`in`** *`mastercounts`* **`after`** *`acounts`*				
parameters				
`slavecounts`	counts of the axis encoder			
`mastercounts`	counts of the master encoder			
`acounts`	counts of the master encoder to "wait for" before applying the gear/for...in...			

This statement temporarily modifies the gear ratio of the slave axis such that a *slavecounts* correction (offset) occurs over a master-feedback displacement of *mastercounts*. The *slavecounts* correction may be positive or negative. An optional after condition can be applied to delay application of the gear/for/in.

Offset Slave Position	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`offset slave by` *`slavecounts`* **`in`** *`time`*				
parameters				
`slavecounts`	counts of the axis encoder			
`time`	time, sec			

This statement offsets the position (and therefore phase) of the axis such that a *slavecounts* correction (the offset) occurs over a period of *time*. The *slavecounts* correction may be positive or negative.

Wait for Counts of Master	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
`wait master` *`counts`*				
parameters				
`counts`	counts of the master			

This statement waits until the specified number of master *encoder* counts has been generated.

Wait for Counts of Slave	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				

wait slave *counts*	
parameters	
counts	counts of the axis (slave)

This statement waits until the specified number of axis (slave, target-position) counts has been generated.

Wait Within	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
wait *source* **within** *start* , *end*				
parameters				
source	**master1, master2, master3, master4** or **slave**			
start	a modulo starting bound			
end	a modulo ending bound			

This statement waits for the *modulo position* (either *mposc1-4* or *sposc*) to lie within the specified bounds.

Wait Outside	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
wait *source* **outside** *start* , *end*				
parameters				
source	**master1, master2, master3, master4** or **slave**			
start	a modulo starting bound			
end	a modulo ending bound			

This statement waits for the *modulo position* (either *mposc1-4* or *sposc*) to lie outside the specified bounds.

Clear Temporary Gearing Counters	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
zero *masslv* **counters**				
parameters				
masslv	**master** clears tmc1 and tmc2 **slave** clears tsc1 and tsc2			

This statement atomically clears the temporary master or slave counters.

4.10 Position Capture & Registration

Summary:

set capture *transition* **of input** *input* { **gate input** *gateinput gatestate* }
set capwin range *start, end* **using** *reference* { *arm* }
wait capture { **if limit of** *limit* **goto** *limitlabel* }

(Also reference the EtherCAT Applications Guide for additional details pertaining to network drive control).

Set Capture	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set capture *transition* **of input** *input* { **gate input** *gateinput gatestate* }				
parameters				
transition	*rise*, *fall* or *edge* (any)			
input	the input# (1-10, representing all the inputs on the M3-40A card)			
gateinput	the input# (1-10, representing all the inputs on the M3-40A card)			
gatestate	**on** or **off**			

This statement initializes the parameters to be used for all captures on this axis, specifying the input (*capInput*) to use and the optional gated input. If gating is specified, then the specified gating input (*capGate*) must be at the specified gating state (*capGateState*).

The following variables are computed and available after a successful capture:

capposc	capture position in encoder counts
cappos	capture position in user units
capTriggered	flag set to 1 when capture occurs

Note: *capposc* and *cappos* are only valid when *capTriggered* is a 1. Once armed *capposc*/*cappos* will reflect the value latched when the capture input goes active but is not necessarily within the defined capture window. *capTriggered* verifies the capture window against the latched *capposc*/*cappos*, prior to setting.

If more than one running MSB on an M3-40A card arms the *same* input for capture, unexpected capture results may occur.

Only one input may be armed for capture at a time *per axis*. If another input is presently armed when this command is issued, the other input is effectively *disarmed*.

Set Capture Window	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
set capwin range *start, end* **using** *reference* { *arm* }				

parameters	
`start`	Start window position to compare against *reference*. *Reference* >= *start*.
`end`	End window position to compare against *reference*. If equals *start* then no window exists and capture will occur based on input. *Reference* <= *end*.
`reference`	the encoder count scaled reference variable to compare to: **fposc** feedback position **mposc1 - mposc5** master position counters #1 through #5 **mposc** master position counter **smodc** slave position (modulo) **smark** slave marked position **tmc1 tmc2** temporary master counters #1 & #2 **tsc1 tsc2** temporary slave counters #1 & #2 **sdc** slave decrement counter **fposc1** feedback position of axis 1 (fposcA) **fposc2** feedback position of axis 2 (fposcB) **tmodc** temporary master counter mod mmc **sfposc** secondary feedback position of axis
`arm`	If included will arm the capture, if not arm will need to be done by a Wait or On command.

This statement initializes a window to be monitored for valid captures to occur, anything outside this window is considered invalid and ignored. If the capture occurs outside this window it will automatically be re-armed within the loop period (default 800 uS). If 'arm' is specified this statement will automatically arm the capture prior to completing this instruction. The *capwinStart* variable is the start of range and the *capwinEnd* variable is the end of range, inclusive.

⚠ When using open loop stepper tposc is not available thus issue the command 'set simulated feedback on' to have tposc copied to fposc, on each control loop, allowing the use of this command.

Wait Capture	☐ Positioning	☐ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB

syntax

```
wait capture { if limit of limit goto limitlabel }
```

parameters	
`limit`	optional master encoder count limit
`limitlabel`	optional label to branch to if `limit` is reached

This statement waits for the capture and arms the capture input. If the capture occurs the next statement in the MSB is executed. A maximum limit of counts prior to exiting (*capLimit*/*capLimitflag*) can be set. This limit references the 'reference' set by 'set capwin' and the sign must be adjusted accordingly. The *capWait* variable will be set to 1 while the 'wait capture' is active, 0 if not.

If a limit (*capLimit*) is specified, then the statement will branch to the specified goto *limitlabel* after that number of master encoder counts has passed.

4.11 S-Curve

S-Curve support is optionally available for the move commands, from a stopped position. When using timed commands the distance, acceleration, and velocity will be calculated for the given time and then translated to an S-Curve move. The time will not be the same as the non S-Curve move but all other parameters will be, including position. Variables of interest are:

'runv' - velocity fed to the PID algorithm internal use only, read only.

'jerk_a/jerk_d' - acceleration/deceleration actual jerk, read only .

'jerk_a_req/jerk_d_req' – requested acceleration/deceleration jerk in units/sec3, read/write. Set to 1 for automatic calculation.

'sign' – nonzero for S-Curve move, 1 for CCW rotation and -1 for CW rotation, read only.

The minimum jerk that can be used is calculated by the formula (a_{max} * a_{max})/v_{max}), applied independently to the requested acceleration and deceleration jerk. The maximum velocity is the same as the non S-Curve move and defined by the expression:

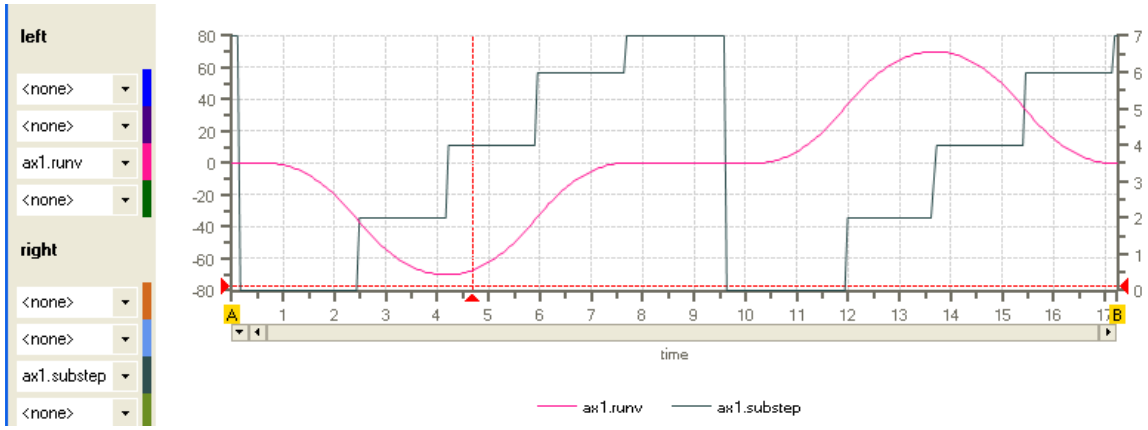
```
sign = 1;
if (delta < 0) {
   sign = -1;
   delta = -delta;
}
a_max = sign * acceleration;
d_max = -sign * deceleration;
V_max = sqrt(2.0 * a_max * d_max * sign * delta / (d_max - a_max));
```

If 'jerk_a_req/jerk_d_req' is 0 then a normal move will be attempted. If only one is set then 'jerk_a/jerk_d' will be set equal. If the requested jerk is greater than the minimum then it will be used. The variables 'jerk_a/jerk_d' are the actual jerk used for the move. Also note that S-Curve uses twice the acceleration and deceleration specified by the non S-Curve move request.

Below shows a sample S-Curve where the jerk is set to 1, thereby having the motion card calculate the optimum jerk and the velocity set to a large number so that the motion card will calculate the maximum velocity possible for the move. Note that the step graph is the segment (substep), 1 to 7, of the S-Curve, with 0 being segment 1. If for some reason the proper velocity or distance can not be attained by the parameters given, the non S-Curve curve move will be used. The end position can not be changed and a slew stop will do the non S-Curve stop. In the example below the motion card will calculate the maximum velocity and optimum jerk. Note there is little or no segment 4 (constant velocity). Also linear segments 2 and 6 are 0.

```
1  // S-Curve sample program
2  drive enable;
3  zero feedback position;
4  // Request jerk to be the same for acc/dec.
5  jerk_a_req = 1;
6  jerk_d_req = 1;
7  [loop]
8  move at 1000 for 246 using 20, 20;    // turn CCW
9  wait for in position;
10 delay 1000;
11 move at 1000 for -246 using 20, 20;   // turn CW
12 wait for in position;
13 delay 2000;
14 goto loop;
```

Resulting motion S-Curve using QuickScope:



⚠ 800uS is the default loop period. If 500uS is desired use the 'set loopperiod .0005' command prior to drive enable.

4.12 Linear and Circular Interpolation (Vectors)

Reference the EtherCAT Applications Guide for details.

5 Chapter 5: Camming and Data Tables

Camming tables in QuickMotion are two-dimensional arrays of floating-point data. There are 6 tables available for use, numbered 0 through 5, each having up to 2000 rows and always 2 columns. These columns are named "x" and "y". Although their primary use is to hold data for *spline*- and *CAM*-based motion, they can be used to hold arbitrary data such as positions for recipe-based motion. Although limited to 6 tables, these tables can also be swapped out dynamically and refreshed with new data when loaded from the controller file system.

Spline tables use the "x" column as time and the "y" column as a *relative* position. *CAM* tables use the "x" column as a *relative master* position and the "y" column as a *relative slave* position.

Since *spline* and *CAM* tables use *relative* position data, the first point pair in these tables must be 0.0, 0.0 (time/master-position of 0, position/slave-position of 0). The exception to this is with CAM tables where the y component can be non-zero in newer firmware revisions, thereby establishing an offset. In addition, for any tables used for *spline* and *CAM* operations, all "x" values must be increasing, that is: a given row's "x" must be greater than the previous row's "x". Also, the minimum number of rows (pairs) in these tables is 3.

⚠ It is recommended that CAM tables and instructions be used whenever possible. Significant enhancements have been made to camming which have currently not been carried forward to splines. Some of this consists of the ability to start on non-zero y column values, ability to start anywhere within a table, and forward and reverse table traversing.

Points in a *spline* or *CAM* table are also referred to as *knots,* as they represent critical loci that must be passed through when interpolation occurs.

For example, in the following *spline* table:

0.0	0.0
1.5	2.0
2.0	2.5
3.0	3.0
4.0	2.0
5.0	0.0

there are 6 knots. Since this is a *spline* table, the last 5 knots are interpreted as follows:

At time = 1.5 seconds, the position of the axis should be 2.0 user-units beyond where the axis started this spline move.
At time = 2.0 seconds, the position of the axis should be 2.5 user-units beyond where the axis started this spline move.
At time = 3.0 seconds, the position of the axis should be 3.0 user-units beyond where the axis started this spline move.
At time = 4.0 seconds, the position of the axis should be 2.0 user-units beyond where the axis started this spline move.
At time = 5.0 seconds, the position of the axis should be back where the axis started this spline move.

The position of the axis between these "knots" is determined by the interpolation method specified by the QM code when the table is *started*.

The three available interpolation methods in QM for *spline* (and *CAM* tables) are:

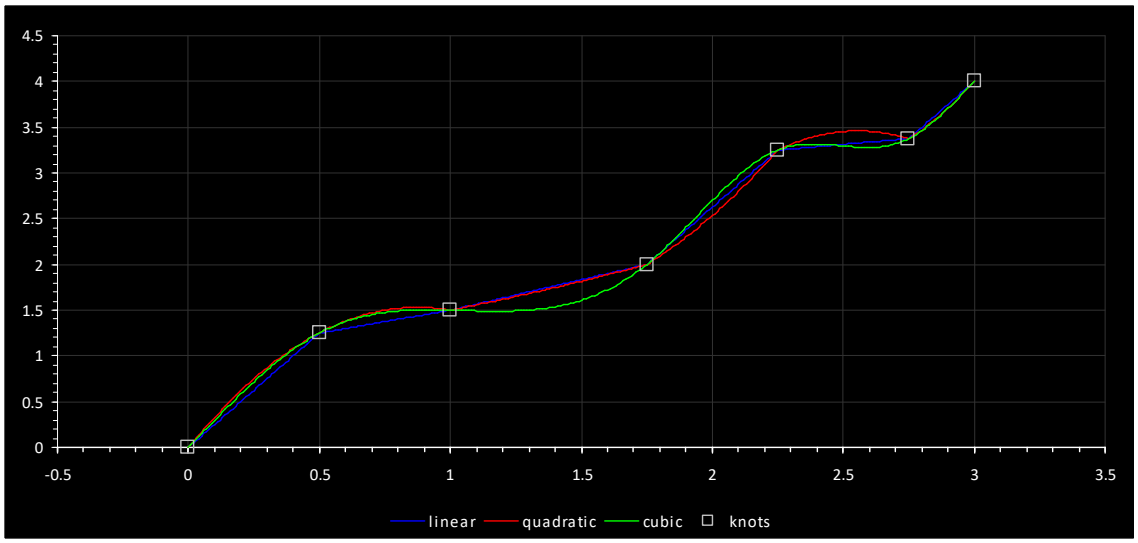
linear a straight-line joins each knot

quadratic a piecewise 2nd degree polynomial is fitted between this knot and the next; the first derivative of the first point is forced to 0.

cubic a piecewise 3rd degree polynomial is fitted between this knot and the next two knots; the first and second derivatives of the first point is forced to 0.

The following graph shows different interpolation methods using the following table of knots:

0.000	0.000
0.500	1.250
1.000	1.500
1.750	2.000
2.250	3.250
2.750	3.375
3.000	4.000



CAM tables are interpreted similar to their time-based *spline* counterparts. For example, in the following *CAM* table:

0.0	0.0
2.5	1.0
4.0	-1.0
5.0	0.0

the last 3 knots are interpreted as follows:

At a relative master position of 2.5 user-units, this (slave) axis should be 1.0 user-units beyond where it started.
At a relative master position of 4.0 user-units, this (slave) axis should be 1.0 user-units before where it started.
At a relative master position of 5.0 user-units, this (slave) axis should be where it started.

The master position is kept in the QM variable, `mpos` and is scaled to user-units by dividing by the axis parameter *mppr*. No other scaling occurs (i.e. *uun* and *uud* are not utilized). A raw (counts) variable is also available in *mposc*.

⚠ Unlike splines, Cam tables may start on a non-zero relative position (y). This position is used as an offset.

⚠ 'activeCAM_row' may be set to any desired row upon which mpos will be initialized to that which is the 'x' value of that row, allowing the table to start in that position.

⚠ 'invertmaster' variable is by default set to 0, meaning the cam table is traversed moving from row 0 to N. If 'invertmaster' is set to 1 the cam table position will begin at the end of the table and traverse N to 0. 'activeCAM_row' determines the start position, initialized by the precompute command either to the end or start of the table based upon 'invertmaster'. Prior to a 'table start' command 'activeCAM_row' can be changed to a different start position. 'invertmaster', when set causes mpos to decrement on positive master pulses, thus the reverse traversing of the table.

⚠ 'camming_invertend' variable is by default set to 0, meaning follow the logic described above for 'invertmaster'. If you wish to invert the logic of the 'invertmaster', with regards to camming table positioning only, set this flag. The 'invertmaster' variable will still control whether mpos is added or subtracted from based upon the master but 'caming_invertend', if set, allows you to start at the other end of the camming table. The direction you traverse the camming table is important since if you are at the start of the table and go slightly negative you will hold position but if you go past the end of the table the command will be considered completed.

invertmaster	camming_invertend	
0	0	master difference added to mpos, assume moving from beginning of camming table to end (thus go beyond end COMPLETE).
0	1	master difference added to mpos, assume moving from end of camming table to beginning (thus go beyond beginning COMPLETE).
1	0	master difference subtracted from mpos, assume moving from end of camming table to beginning (thus go beyond beginning COMPLETE).
1	1	master difference subtracted from mpos, assume moving from beginning of camming table to end (thus go beyond end COMPLETE).

5.1 Loading Tables

Summary:

table *n* **clear**
table *n* **addpair** *xexpression , yexpression*
table *n* **addseries** *pairs*
table *n* **copy from** *rowOffset1* **to table** *m rowOffset2 numRows*
table *n* **loadoffset** *rowOffsetFile, numPairs,rowOffsetTable*
table *n* **loadseries** *source fileNumber*

In order to use a table, it must be loaded with point pairs. There are several QM statements which facilitate loading of tables. These statements allow tables to be loaded either directly from within program code, thus static data, or dynamically from binary table files which reside on the controller file system. The commands that effect table loading are:

Table Clear	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
table *n* **clear**				
parameters				
n	the table to clear			

Clears a table of all of its points, thus setting the number of data points to 0, within a table.

```
table 1 clear;
```

There can be no active motion command when this statement is issued.

Table Add Pair	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
table *n* **addpair** *xexpression , yexpression*				
parameters				
n	the table to add a point pair to			
xexpression	an expression which when evaluated will be utilized as the value in the "x" column			
yexpression	an expression which when evaluated will be utilized as the value in the "y" column			

This statement adds a point pair to a table. This statement is used when the table is computed at MSB runtime since the pair is computed by two expressions.

```
table 2 addpair 3.75 + ztime, q + zoffset;
```

⚠ There can be no active motion command when this statement is issued.

⚠ An error will occur if there are already 2000 rows in the table.

Table Add Series	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
table *n* **addseries** *pairs*				
parameters				
n	the table to add a point pair to			
pairs	a series of one or more pairs (in the form of x,y), colon-delimited			

This statement adds constant point pairs to a table.

```
// add 4 point pairs to table 1

table 1 addseries 0.0,0.0 : 1.0,1.5 : 2.0,1.75 : 3.0,2.0;
```

⚠ There can be no active motion command when this statement is issued.

⚠ An error will occur if adding these pairs will result in a table with more than 2000 rows.

Table Copy	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
table *n* **copy from** *rowOffset1* **to table** *m rowOffset2 numRows*				
parameters				
n	The table which is source of the copy.			
rowOffset1	The source table row offset, 0 is no offset.			
m	The table which is destination of the copy.			
rowOffset2	The destination table row offset, 0 is no offset, -1 is append.			
numRows	The number of rows to copy, 0 is all.			

This statement allows for one table to be copy or appended to another table. The destination table does not need to exist. The offsets can be used to merge table data.

```
table 1 copy from 0 to table 2 0 0;    // Copy all of table 1 to 2
```

Table Loadoffset	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
table *n* **loadoffset** *rowOffsetFile, numPairs,rowOffsetTable*				
parameters				
n	The table to set the offset information on.			
rowOffsetFile	The file row offset to begin transfer on, 0 is no offset.			
numPairs	The number of cam file pairs to transfer, 0 is all.			
rowOffsetTable	The cam table row offset to begin storing file at, 0 is start.			

This statement works in conjunction with the 'loadseries' command, setting the offsets to be used. The offsets can be used to merge table data. This command only initializes parameters for 'loadseries' and does not directly effect the table.

```
table 1 loadoffset 0 0 0; // Default, transfer all from start .
```

Table Loadseries	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
table n **loadseries** *source fileNumber*				
parameters				
n	The table to load the cam file into.			
source	The location on disk where file will be found, 'flash' or 'ram' (/_system/Datatables or /RAMDISK/Datatables).			
fileNumber	The file to transfer, 'camtable#.tbl', where # is any valid positive number. A variable may be referenced as well.			

This statement requests a cam file to be transferred from the controllers file system. The file is transferred to the MSB for local storage and must be precomputed prior to operation. The 'loadoffset' parameters are referenced for this command as to where within the file and table to begin the transfer.

```
table 1 loadseries ram 1; // Load file 'camtable1.tbl'
```

The file format of 'camtable#.tbl' consists of a binary file of 32 bit float pairs with a file record structure as:

float rows[NUMROWS][2];

Where NUMROWS is the number of cam file pair entries, with the first starting at 0, 0. The same rules as the 'addseries' command exists. Each float is stored in little endian format with X being the first float.

As an example of a 3 row table with the values of:

0, 0
250, 25.67

500, 50.48

The binary data within a file would consist of 24 bytes, 4 bytes per entry in little endian and IEEE-754 floating point format. Below is byte representation of the required file, in hex:

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x7A 0x43 0x29 0x5C 0xCD 0x41
0x00 0x00 0xFA 0x43 0x85 0xEB 0x49 0x42

IEEE-754 conversion calculator is available at:

http://babbage.cs.qc.cuny.edu/IEEE-754/Decimal.html

For example, 50.48, is 0x4249EB85, reversed when stored in the file since in little endian format (low byte first).

5.2 Using Tables for Spline/CAM

Summary:

table *n* **continue**
table *n* **precompute**
table *n* **start** *imethod tscale , rpscale , repeatcount*
table *n* **start** *imethod* **cam** *mpscale , spscale , repeatcount*
stop table

Table Continue	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
table *n* **continue**				
parameters				
n	The table to continue that was previously stopped.			

Continues a cam table that was stopped by the 'stop table' command. Note that this command should only be used if the master position stopped at the beginning of the next table row position otherwise any row that is currently being executed during a 'stop table' (immediate stop) will be re-executed. The master position allows for a slewed stop. Upon execution of 'stop table', with the servo not moving, would then save all the camming information so that you can exit camming, jog into position, and then continue with the same camming table from where left off with the 'table *n* continue' command.

```
table 1 continue;
```

⚠There can be no active motion command when this statement is issued.

⚠ In many cases this command is no longer needed since the 'activeCAM_row' can be set prior to 'table start'. Only supported in camming mode, not spline.

Table Pre-compute for Spline/CAM	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☐ FG MSB
syntax				
table *n* **precompute**				
parameters				
n	the table to pre-compute			

This statement readies a table for use by a *spline/CAM* motion. After points have been added to a table, there are a series of computations that need to occur before the table can be utilized for spline and CAM motion operations. This statement causes those computations to occur.

There is no need to issue this command if a table is being utilized simply for data (i.e. for *tbln*, *tblx* or *tbly* operations).

Failure to *precompute* a table before *starting* the table will cause a *hard fault*.

⚠ It takes roughly 250ms to *precompute* a 1000 row table.

⚠ The table must contain at least 3 points and all 'x' column values must be *increasing* or an error will occur. The first point in the table must be 0.0,0.0 otherwise an error will occur.

⚠ In camming mode 'invertmaster' set to 0 will cause 'precompute' to initialize the table to move from start to end, if set then from end to start, with positive master position motion.

```
// prepare the table for CAM use

table 1 precompute;
```

Table Start Spline Motion	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB

syntax

table *n* **start** *imethod tscale , rpscale , repeatcount*

parameters

n	the table to utilize for motion
imethod	**linear** uses linear interpolation **quadratic** uses 2nd order interpolation **cubic** uses 3rd order interpolation
tscale	time scale factor: the values in the "x" (time) column in the table are effectively divided by this number
rpscale	relative position scale factor: the values in the "y" (relative position) column in the table are effectively multiplied by this number
repeatcount	the number of times to "run through" the table (0=forever)

This statement starts *spline* motion using the specified table.

The current (target) position is used as the starting relative-position for the motion.

The table must be *ready* for use (i.e. a *table precompute* operation has been successfully completed on the table).

⚠ "In position" will be *true* only when the table has completed all required repeats. If 0 (forever) is specified for the `repeatcount`, then "In position" will never be true unless a `stop table` is issued.

⚠ The scale factor *tscale* must evaluate to a value greater than 0 otherwise an error will occur.

⚠ The scale factor *rpscale* must evaluate to a value other than 0 otherwise an error will occur.

⚠ The scale factor *rpscale* is not affected by uun or uud. The generated position is also unaffected by uun or uud.

```
[top]

zero feedback position;

table 1 start quadratic 1.0, 1.0, 0;  // run through the table
forever, 'table stop' command will cause the background motion
command to stop.
```

Table Start CAM Motion	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB

syntax

table *n* **start** *imethod* **cam** *mpscale* , *spscale* , *repeatcount*

parameters

n	the table to utilize for motion
imethod	**linear** uses linear interpolation **quadratic** uses 2nd order interpolation **cubic** uses 3rd order interpolation
mpscale	master-position scale factor: the values in the "x" (master-position) column in the table are effectively divided by this number
spscale	relative slave-position scale factor: the values in the "y" (relative slave position) column in the table are effectively multiplied by this number
repeatcount	the number of times to "run through" the table (0=forever)

This statement starts *CAM* motion using the specified table.

The master position (*mpos*, *mposc*) is not cleared when this statement is executed, the *activeCAM_row* will be used to offset into the cam table and that position will become *mpos/mposc*. *'table precompute'* will set the *activeCAM_row* to 0.

The current (target) position is used as the starting relative slave-position for the motion.

The table must be ready for use (i.e. a *table precompute* operation has been successfully completed on the table).

⚠ If the master position "*backs up*" past 0 (its initial position) and the "*repeats left to do*" counter is *greater than* 1, then the "*repeats left to do*" counter is decremented and the master-position wraps. If the repeatcount was specified as 0 (forever), then the master-position will always wrap.

⚠ "In position" will be *true* only when the table has completed all required repeats. If 0 (forever) is specified for the *repeatcount*, then "In position" will never be true unless a *stop table* is issued.

⚠ The scale factor `mpscale` must evaluate to a value greater than 0 otherwise an error will occur.

⚠ The scale factor `spscale` must evaluate to a value other than 0 otherwise an error will occur. The scale factor `rpscale` is not affected by uun or uud. The generated position is also unaffected by `uun` or `uud`.

```
[top]

zero feedback position;

table 1 start quadratic cam 1.0, 1.0, 1;
```

Table Stop	☑ Positioning	☐ Slewing	☐ Tracking	☑ BG MSB ☑ FG MSB
syntax				
stop table				

This statement stops *spline* or *CAM* motion. If in *CAM* motion then the current table state is saved in case a 'table continue' command is executed.

If there is no active `table start` (*spline*/*CAM*) motion, then this command is effectively ignored and no fault occurs.

Unlike other `stop` statements, `stop table` will never generate a *hard fault*.

```
wait until mpos > 9;

stop table;
```

5.3 Accessing Table Data

As mentioned earlier, tables may also be used to store/retrieve data. In QM, there are special array access operators called `tblx[]`, `tbly[]` and `tbln[]` that allow the user to retrieve information from a table.

`tblx[]` and `tbly[]` retrieve the "x" value (`tblx`) or "y" value (`tbly`) from a given row in a table. `tbln[]` retrieves the total number of rows in a table.

Their syntax is as follows:

```
tblx[table#, row]

tbly[table#, row]

tbln[table#]
```

⚠ Any attempt to read a value outside the bounds of the table will result in a value of 0.

They can be used in any QM expression, as show in the example QM code below:

```
// use table 1 for "move in time" pairs

// x will hold the move time (although it can hold anything we want)

// y holds an absolute position

// note that 'x' values in the table don't have to be

//   in increasing form as they do for spline/cam

//   since we are using the table just as data

// also note that there is no 'precompute' as the table is

//   just being used for data and not spline/cam

table 1 clear;

table 1 addseries 1.0,1.0 : 0.5,1.5 : 2.0,3.75 : 1.0,6.0;

[top]

zero feedback position;

// set index to 0 (indexes into tables are 0-based)

i = 0;
```

```
// grab how many pairs are in table 1

n = tbln[1];

[loop]

// done?

if i >= n goto top;

// grab data

move in tblx[1,i] to tbly[1,i];

wait for in position;

delay 1000;

// increment index

i = i + 1;

goto loop;
```

5.3.1 Diagnosing Table Issues

When table data is loaded by an MSB it can be difficult to determine if it is correct from a diagnostic viewpoint. Diagnostic variables exist that can be monitored to allow a user to walk through the table to visually or programmatically verify data from QuickBuilder. A Quickbuilder Debug Window can be used to view the Diagnostic Variables listed below:

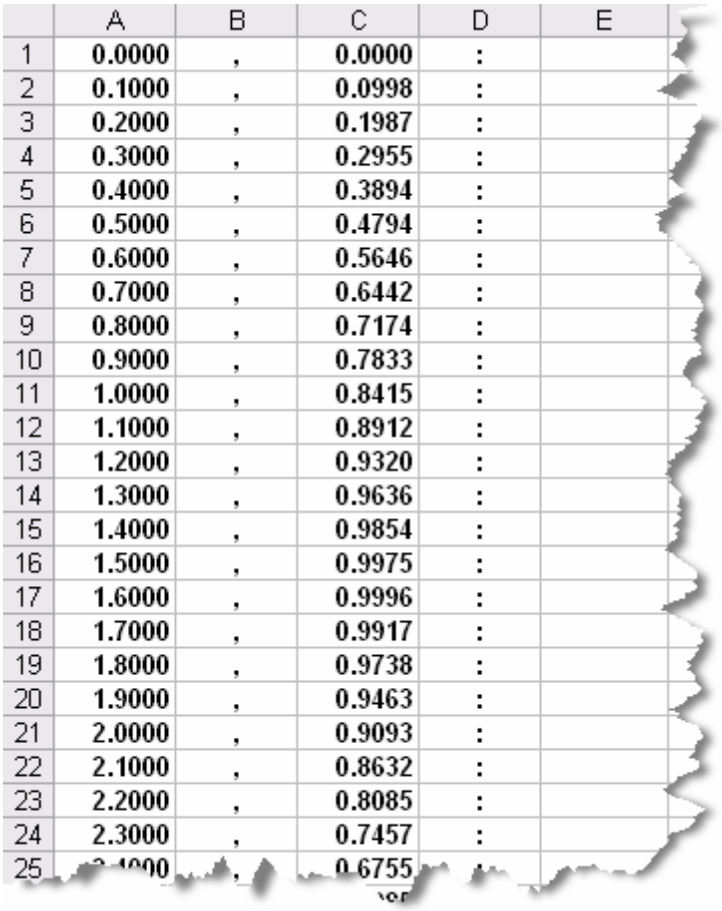
Diagnostic Variables	Description	Type
debugTable	Cam table to view, from 0 to 5, representing table 1 to 6 since 0 based.	read-write
debugTableRows	Number of rows presently in the selected cam table, debugTable.	read-only
debugTableRow	Current row number to view in the selected cam table, debugTable.	read-write
debugTableX	X value for selected debugTableRow.	read-only
debugTableY	Y value for selected debugTableRow.	read-only

⚠ The above variable are only available from a QuickBuilder Debug Window, when executing an MSB use the tblx, tbly, and tbln commands discussed in the previous section.

5.4 Microsoft Excel as Table Data

It is relatively simple to use data from Microsoft Excel as table data.

One can easily create four columns with *x-data*, a column containing a comma (","), *y-data* and lastly a column containing a colon (":") as follows:



	A	B	C	D	E
1	0.0000	,	0.0000	:	
2	0.1000	,	0.0998	:	
3	0.2000	,	0.1987	:	
4	0.3000	,	0.2955	:	
5	0.4000	,	0.3894	:	
6	0.5000	,	0.4794	:	
7	0.6000	,	0.5646	:	
8	0.7000	,	0.6442	:	
9	0.8000	,	0.7174	:	
10	0.9000	,	0.7833	:	
11	1.0000	,	0.8415	:	
12	1.1000	,	0.8912	:	
13	1.2000	,	0.9320	:	
14	1.3000	,	0.9636	:	
15	1.4000	,	0.9854	:	
16	1.5000	,	0.9975	:	
17	1.6000	,	0.9996	:	
18	1.7000	,	0.9917	:	
19	1.8000	,	0.9738	:	
20	1.9000	,	0.9463	:	
21	2.0000	,	0.9093	:	
22	2.1000	,	0.8632	:	
23	2.2000	,	0.8085	:	
24	2.3000	,	0.7457	:	
25	2.4000	,	0.6755	:	

Since the QuickBuilder editor allows free-form lines, the data can simply be copied and pasted into QM code in an MSB such as:

```
table 1 clear;

table 1 addseries

// paste Excel cells here
;
```

⚠ The last colon (":") in the last row will need to be removed using this method.

⚠ Refer to the loadseries command to dynamically load tables from a binary file stored on the controller's file system.

5.5 Virtual Master

At times a virtual master is required. This can be done in one of two ways:

1. Use the `move master` command to generate background pulses on the selected access based on timer loop tick counts. This allows the same access to operate normally, as an axis.

2. Setup a simulated axis, which runs as though it was receiving real encoder input, although it is not. The master position can then be published so others can track to it. Just about all motion commands are valid during a simulation, including s-curve.

Once a method of generating a master position is determined it can then be published across the backplane of the controller using variant 36827 (see Virtual Master Broadcasting).

Move Master Position	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
move master at *rate* **for** *limit* { **using** *ramp* }				
move master at *rate* **forever** *limit* { **using** *ramp* }				
parameters				
rate	The pulse rate of the encoder in pulses added per position loop period (800us default on the M3-40A).			
limit	The total number of pulses to generate.			
ramp	Optional ramp added or subtracted from rate at the position loop period (800us on the M4-40A).			

This statement virtually "moves" the master encoder by changing its "position" at the specified *rate* for a certain number of generated pulses (first form, specified by *limit*). If the *limit* is set to 0 then just the *rate* will change, dynamically, using any specified *ramp*, from the current rate.

To generate a continuous stream of pulses, use the second form. *move_master_ramp* and *move_master_rate* variables can be referenced to check current settings of virtual master.

Reference **`set master`** *`mode { using global }`* `for additional information.`

Example:

```
// stop the virtual axis
move master at 0 forever using 1;
// set the virtual access global
set master virtual using global;   // this will place the master information in dual ported memory for
broadcast to slave axis
```

```
// start the virtual axis
move master at 100000 forever using 1;
```

One of the drawbacks of the 'move master at' command is that it is based on the loop period and not user units. This can make things difficult to program. The benefit is the axis is available for motion. If an axis can be reserved and dedicated as a master then a simulated feedback can be used. In essence the axis becomes a virtual axis, responding to commands as it would on a real axis:

```
set simulated feedback on; // this will cause fposc to = tposc after each loop period, drive must not be enabled
```

5.5.1 Broadcasting (M3-40 only)

When the 'set master virtual using global' command is given, mposc delta counts (mposc - last_mposc) are placed in a dual ported memory for broadcast by the controller, across the backplane. This allows muliple axis to follow a master. Up to 4 masters are currently support with a broadcast update rate of 4ms. Prior to broadcasting, the controller must be initialized with the proper master/slave information

A special variant table located at 36827 supplies the interface to control virtual broadcasting. In Quickbuilder define a table of type 'int' with an override of 36827.



Once created the variable table columns are defined as follows where masterNum is 0 to 3 (up to 4 masters):

MasterArray[masterNum][0] - 0 disables broadcasting, non-zero enables.

MasterArray[masterNum][1] - Master axis number, 1 to 32.

MasterArray[masterNum][2] - Slave axis bit positions, up to 32 supported (16 2 axis cards). Which slaves to replicate master information to.

Example:

```
MasterArray[0][0] = 0;   // disable any running master

MasterArray[0][1] = 1;  // Axis number 1 will be a master

MasterArray[0][2] = 14;  // Axis 2, 3, and 4 are slaves referencing master:  0x0000000E
```

5.6 Segmented Moves and Examples

Summary:

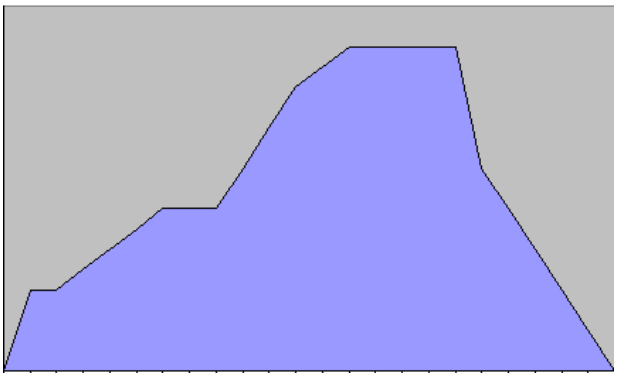
segmove table clear
segmove table accdec to vel using rate
segmove table accdec to vel for displacement
segmove table slew until position
segmove table stop at position using rate
segmove table start relative

Topics:

- Concept
- Commands
- Examples

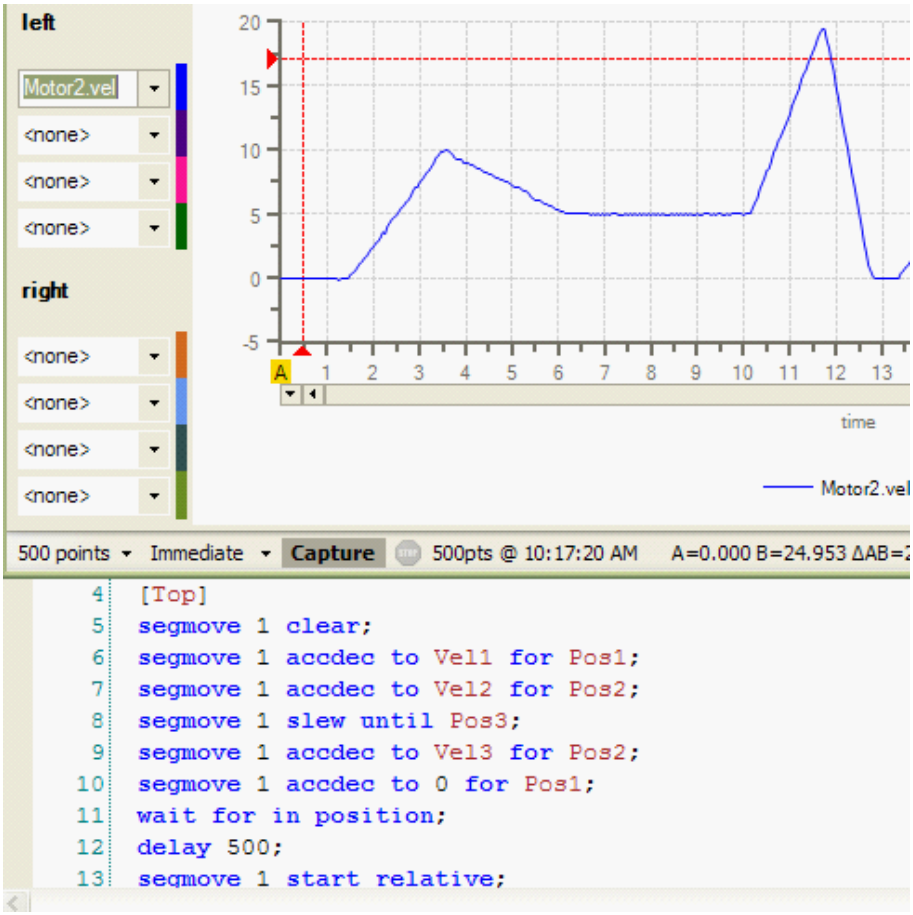
5.6.1 Concept

A segmented move is a precompiled move with multiple distances, acceleration, and velocities tied together. Below is the velocity profile of an example segmented move.



Up to 16 Segmented Move "tables" can be defined with up to 20 segments each residing within them. Once a segment table has been defined and then started, you can redefine that same table while it runs without affecting the segment table in progress.

Below is an example of a segmented move with 5 segments using table 1. You define each acceleration or deceleration ramp and each constant velocity ramp as a separate segment.



5.6.2 Commands

Creating and running a table is easy and uses the following procedure:

1. Clear the Table.
2. Create up to 20 acc/dec and constant velocity segments.
3. Start the Table.

Clear Segmented Move Table	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
segmove *table* **clear**				
parameters				
table	what table to clear: 1 to 16			

This command clears any existing table information
Example: **segmove** 1 **clear;**

Add Segmented Move to Table at rate	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB

				☑ FG MSB
syntax				
segmove *table* **accdec to** *vel* **using** *rate*				
parameters				
table	Which table to add to: 1 to 16			
vel	Velocity, user-units/sec			
rate	Acceleration/deceleration rate, user-units/sec/sec			

This command adds an acc/dec segment from the current velocity to the new <vel> at the specified <rate>.
Example: `segmove 1 accdec to Vel1 using Acc1;`

Add Segmented Move to Table over displacement	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
segmove *table* **accdec to** *vel* **for** *displacement*				
parameters				
table	Which table to add to: 1 to 16			
vel	Velocity, user-units/sec			
displacement	Incremental position, user-units			

This command add an acc/dec segment from the current velocity to the new <vel> over some <displacement>. Note this is an incremental acc/dec segment.
Example: `segmove 1 accdec to Vel2 for Pos2;`

Add Segmented Move to Table (slew)	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
segmove *table* **slew until** *position*				
parameters				
table	Which table to add to: 1 to 16			
position	Velocity, user-units/sec			

This command adds a constant velocity segment until reaching some specified <position>. This is an absolute position from the start of the profile. Prior segments in table must represent movement before this command is accepted, otherwise a fault will occur as table is built
Example:

```
segmove 1 slew until Pos3;
```

Add Segmented Move to Table (stop)	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB

				☑ FG MSB
syntax				
segmove *table* **stop at** *position* **using** *rate*				
parameters				
table	Which table to add to: 1 to 16			
position	Position to stop at, user-units			
rate	Acceleration/deceleration rate, user-units/sec/sec			

This command stops motion at the specified *position,* with a given *rate*. This will cause motion to stop at an absolute position at a specified deceleration rate. Prior segments in table must represent movement before this command is accepted, otherwise a fault will occur as table is built.
Example:

```
segmove 1 stop at Position using Accel;
```

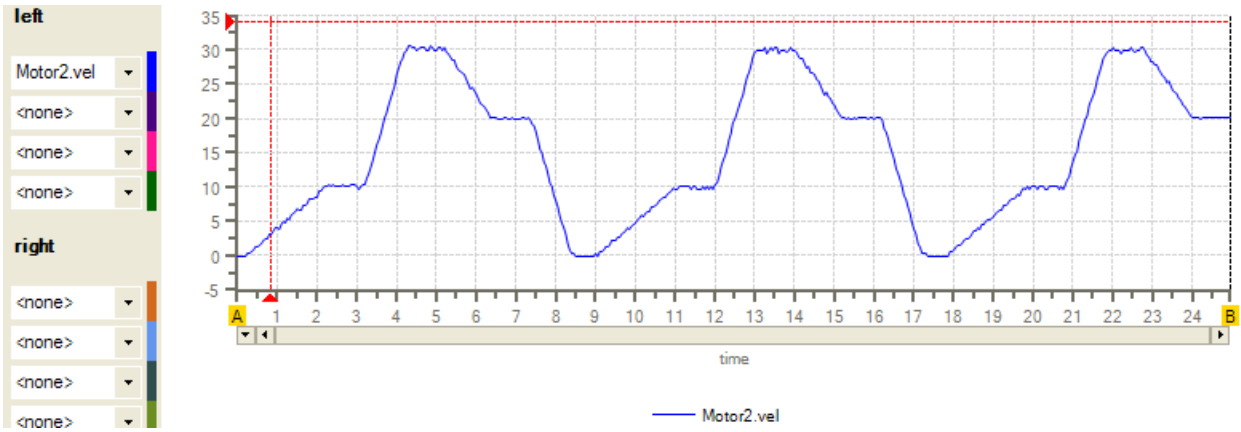
Add Segmented Move to Table (relative)	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
segmove *table* **start relative**				
parameters				
table	Which table to add to: 1 to 16			

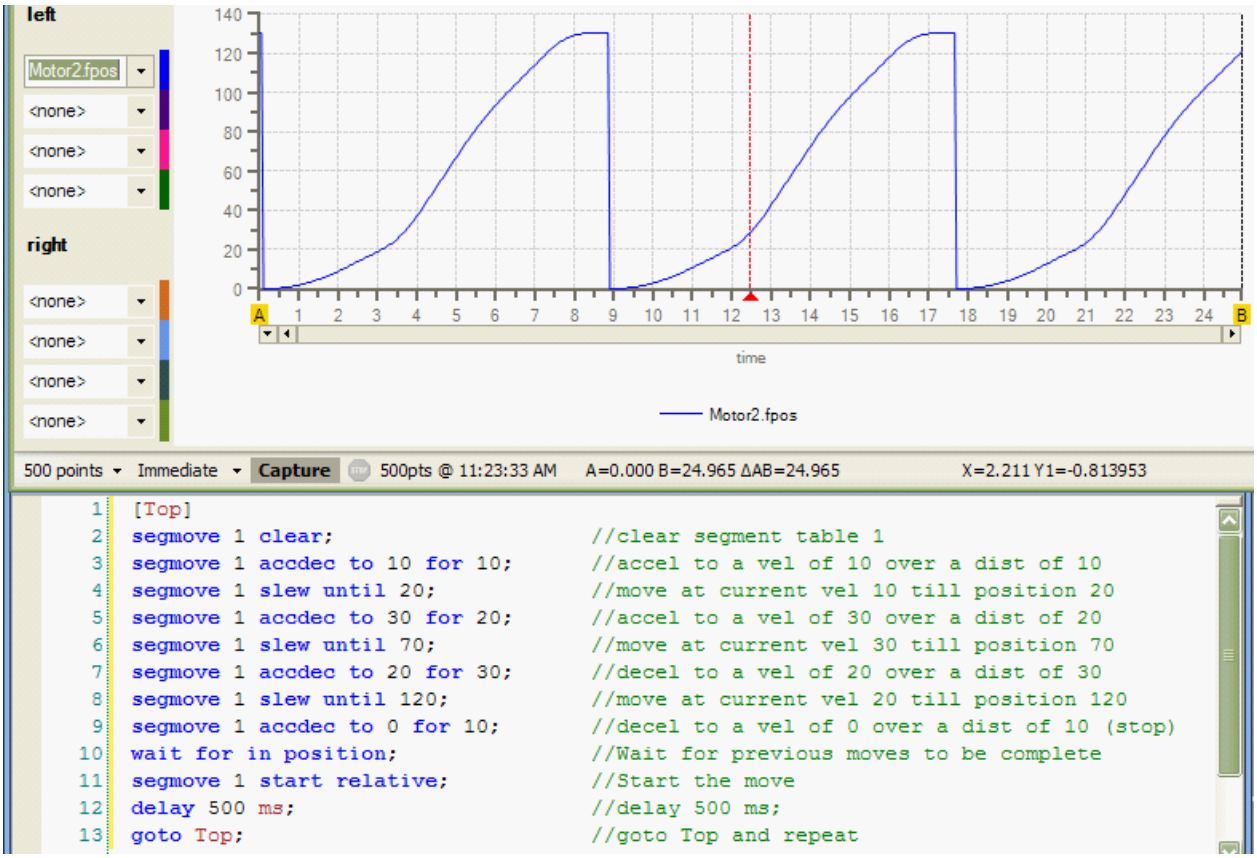
This command starts a relative segmented move – a "zero feedback position" occurs automatically upon executing this command.
Example:

```
segmove 1 start relative;
```

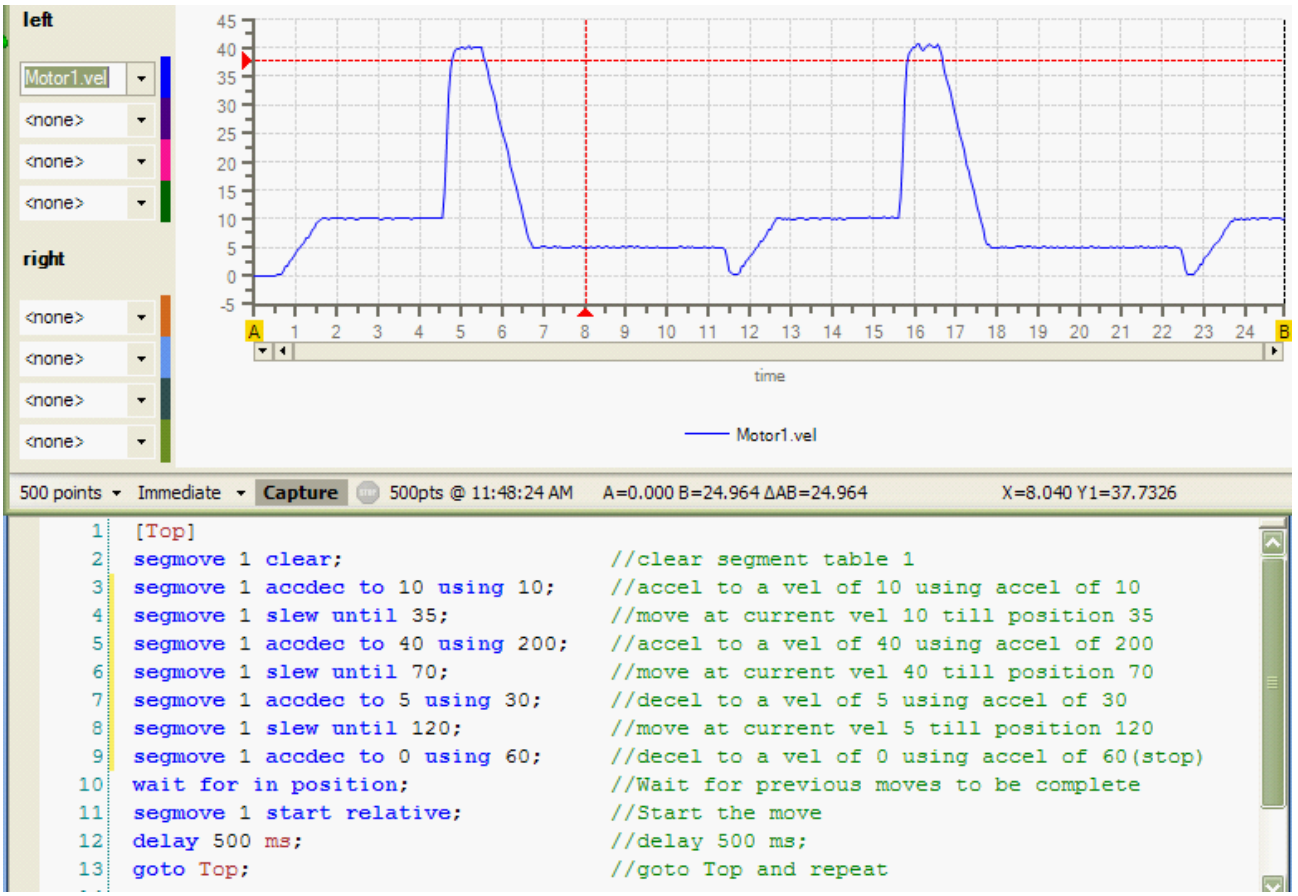
5.6.3 Examples

The following pages include screen shots of move profiles and the code used to create them.

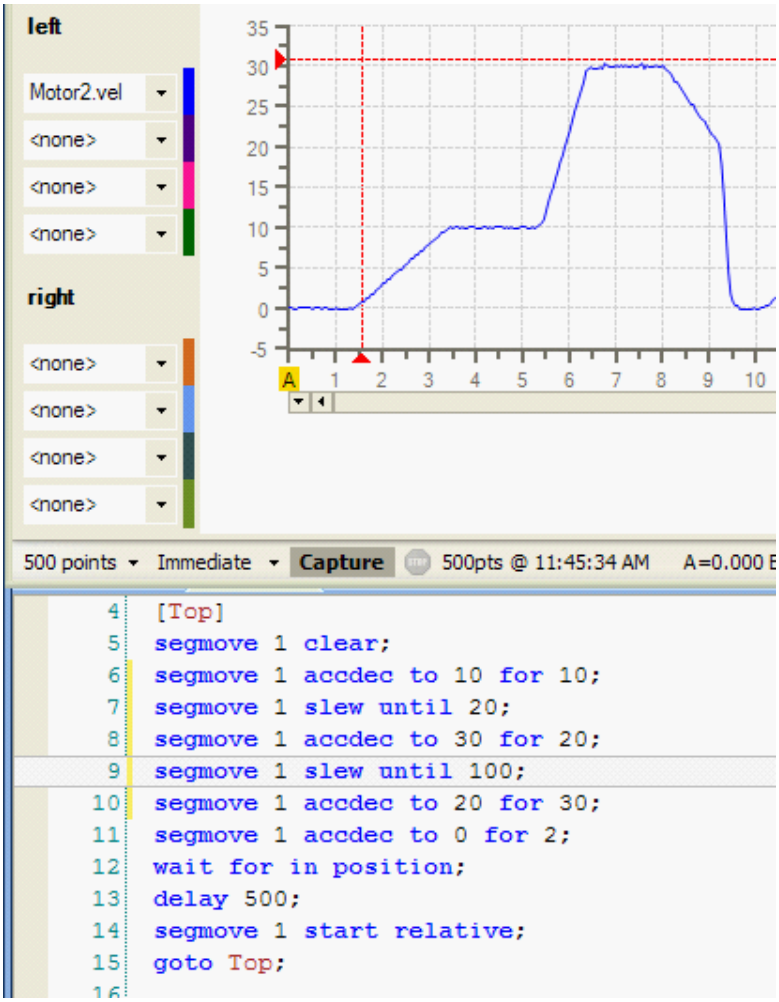




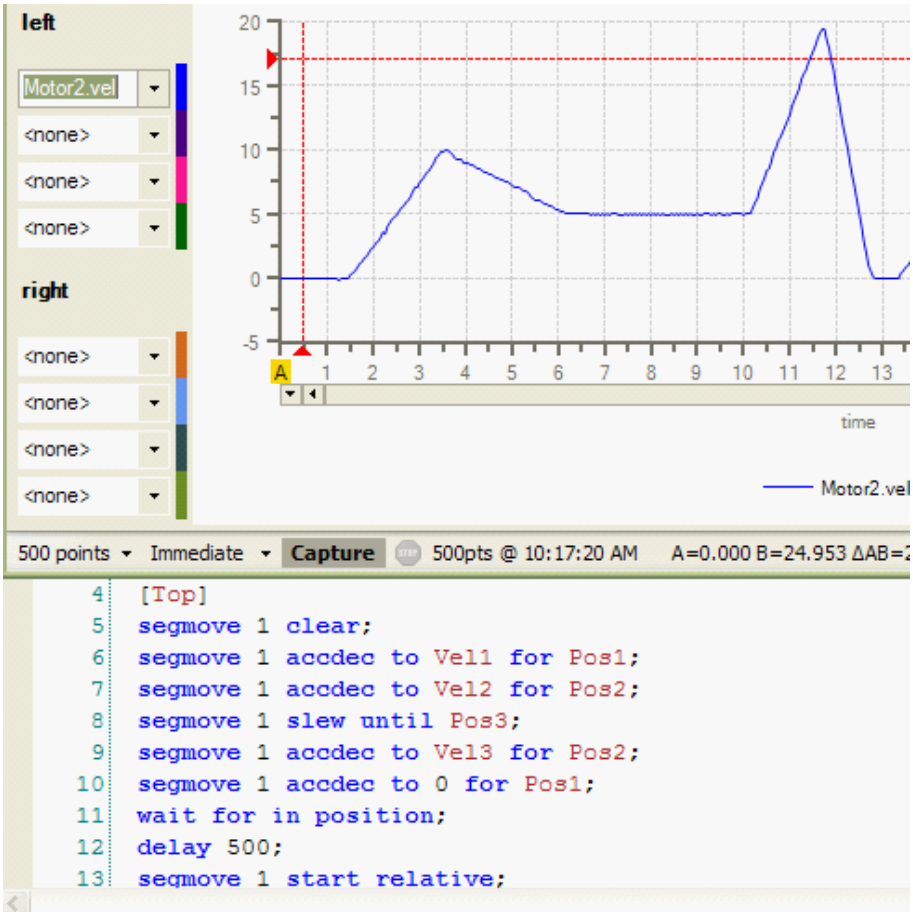

```
[Top]
segmove 1 clear;                    //clear segment table 1
segmove 1 accdec to 10 for 10;      //accel to a vel of 10 over a dist of
10
segmove 1 slew until 20;            //move at current vel 10 till position
20
segmove 1 accdec to 30 for 20;      //accel to a vel of 30 over a dist of
20
segmove 1 slew until 70;            //move at current vel 30 till position
70
segmove 1 accdec to 20 for 30;      //decel to a vel of 20 over a dist of
30
segmove 1 slew until 120;          //move at current vel 20 till position
120
segmove 1 accdec to 0 for 10;      //decel to a vel of 0 over a dist of
10(stop)
wait for in position;               //Wait for previous moves to be
complete
segmove 1 start relative;           //Start the move
delay 500 ms;                       //delay 500 ms;
goto Top;                           //goto Top and repeat
```




```
[Top]
segmove 1 clear;                    //clear segment table 1
segmove 1 accdec to 10 using 10;    //accel to a vel of 10 using accel of
10
segmove 1 slew until 35;            //move at current vel 10 till position
35
segmove 1 accdec to 40 using 200;   //accel to a vel of 40 using accel of
200
segmove 1 slew until 70;            //move at current vel 40 till position
70
segmove 1 accdec to 5 using 30;     //decel to a vel of 5 using accel of 30
segmove 1 slew until 120;           //move at current vel 5 till position
120
segmove 1 accdec to 0 using 60;     //decel to a vel of 0 using accel of
60(stop)
wait for in position;               //Wait for previous moves to be
complete
segmove 1 start relative;           //Start the move
delay 500 ms;                       //delay 500 ms;
goto Top;                           //goto Top and repeat
```




```
[Top]
segmove 1 clear;
segmove 1 accdec to 10 for 10;
segmove 1 slew until 20;
segmove 1 accdec to 30 for 20;
segmove 1 slew until 100;
segmove 1 accdec to 20 for 30;
segmove 1 accdec to 0 for 2;
wait for in position;
delay 500;
segmove 1 start relative;
goto Top;
```

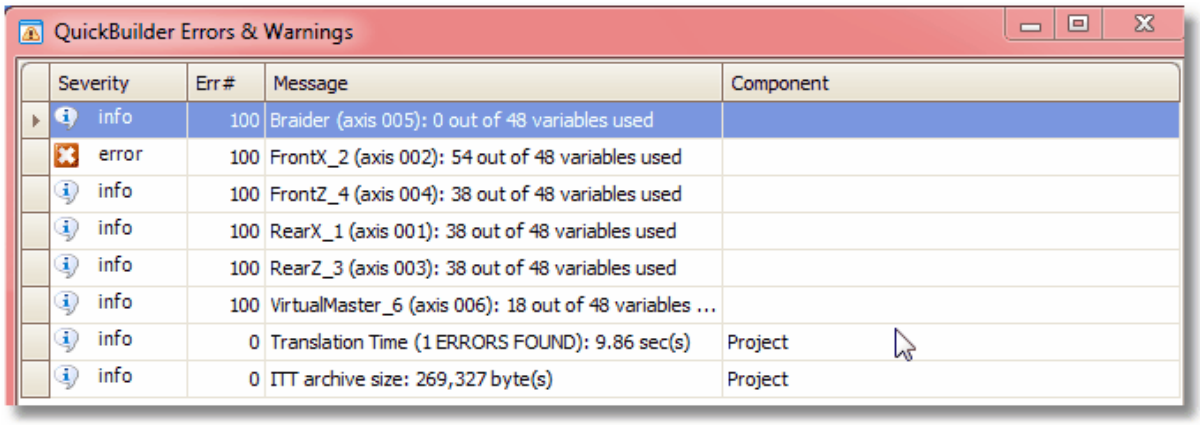



```
[Top]
segmove 1 clear;
segmove 1 accdec to Vel1 for Pos1;
segmove 1 accdec to Vel2 for Pos2;
segmove 1 slew until Pos3;
segmove 1 accdec to Vel1 for Pos2;
segmove 1 accdec to 0 for Pos1;
wait for in position;
delay 500;
segmove 1 start relative;
```

6 Chapter 6: Motion Variables

6.1 QuickMotion User-defined Variables

When entering code into an MSB, variables are automatically defined as they are typed in. A total of 48 user variables per axis are allowed. Each variable is automatically created as a double-precision floating point variable. When you translate a QuickBuilder program the number of variables used on each axis will be displayed, should too many be referenced an error will be flagged:



Variables used in an MSB are automatically assigned to each axis that uses that particular MSB.

For example in the following example two MSB variables are used: *speed* and *dist*.

```
// move at assigned speed for assigned distance
[top]
move at speed for dist;
wait for in position;
delay 500;
goto top;
end;
```

In this sample project, this is the only MSB that QuickStep is calling for the axis named *ax1*, the *speed* and *dist* variables are automatically added to the *ax1* object at the time of translation in QuickBuilder. Note that they will not be shown in the resource tree under an axis object until the project is translated, because it is only at this point that QuickBuilder knows which axes are using which MSBs.



Because a single MSB can be used by more than one axis, the actual variable name has the axis name pre-pended to it so that it can be uniquely accessed by QuickStep (QS4). So in the above example, the variable *dist* used on

ax1 has a name of *ax1.dist* that is used to access it from QuickStep. If that same MSB were to be used on *ax2* as well the *dist* variable *ax2.dist* would be used in QuickStep.

Note that axis prefixes are only used at the QuickStep level and not within MSBs. The MSB cannot directly access a variable from an MSB running on a different axis. If this information is needed, it can be obtained by first assigning the desired MSB variable to a QuickStep variable. This QuickStep variable can then be assigned to a variable in the other MSB. The methodology for reading and writing variables between QuickStep and an MSB is shown below.

QS4 Example code:

```
// QS4 Sample code showing how to update variables between
// QuickStep and MSBs

// set the MSB variable x for Axis1 DrillPosition
// where, DrillPosition is a QuickStep variable
Axis1.x = DrillPosition;

// set the MSB variable speed for Axis1 to 5
Axis1.Speed = 5;

// set the QuickStep variable AxOneTarget
// to the MSB variable Axis1.Target
AxOneTarget = Axis1.Target
```

Axis1 MSB Example code:

```
// MSB Sample code showing how to use updated variables
// between QuickStep and MSBs

Halfspd = Speed/2;

/* Make a trap move to the DrillPosition specified in the QS4 step at
half speed */
move at Halfspd to x;

wait for in position;    // Wait for move to complete
pulse 1 for 1000;        // Turn the drill output on for 1 sec

/* Move to Target at the speed specified above. */

move at Speed to Target;
```

6.2 QuickMotion Pre-defined Variables

In addition to user-defined variables, there are approximately 100 pre-defined variables for an axis in the QuickMotion language.

Many of these variables correspond to properties in the QS4 world.

The pre-defined variables are organized on the following pages into tables by function. The functional groups are:

- **Status Variables** – These are read-only variables that give information as to the status of a given axis, such as fault code, in position, over-travel reached, etc.
- **Control Variables** – These are a mix of read-only and read-write variables used to set general control conditions for the axis and how it interfaces with the drive. Some of these can only be adjusted before the axis is enabled.
- **Tuning Variables** – These variables are all read-write and they are used to adjust the control loop characteristics. These values can be adjusted while the axis is running either by using the tuning wizard in QuickBuilder, or by directly changing the value of the variable.
- **Feedback Variables** – These variables are a mix of read only and read-write that set the characteristics of the encoder feedback. This is where the counts per revolution and the user unit conversions are set.
- **IO and Register Variables** – These variables are used to read the status of the Axis I/O; change the status of outputs, and assign special functionality to an I/O point such as input to be used for positive over travel. Additionally general purpose global registers are available.
- **Tracking Variables** - This is a large set of variables used to set up electronic gearing and registration type applications. These variables greatly simplify these types of applications from a programming perspective, plus they dramatically improve performance.
- **Capture Variables** - These variables are used for registration/capture routines.
- **Diagnostic Variables** - These variables are useful in monitoring low level functionality internal to QuickMotion.
- **Quickstep Variables** - These variables are used when programming in Quickstep rather than QuickBuilder.
- **Fault Variables** - These variables are used to analyze axis fault conditions.
- **Setup Variables** - These variables are used to initialize and/or check certain parameters read during initialization and setup.
- **RFID Variables** - These variables are used with Turck RFID readers. Reference the EtherCAT Applications Guide.
- **Vector Variables** - These variables are used for vector moves during the implementation of 2D, 3D Linear Interpolation, and 2D Circular Interpolation (EtherCAT only).

Status Variables	Description	Type
axisnum	EtherCAT Only: Contains the axis number an MSB is running on, starting at 1.	INT32, read-write
activeCAM_row	Active cam row presently executing in cam table.	INT32, read-only
camRequest	1 requesting cam file from controller disk, 0 idle, else error code.	INT32, read-only
capStatus	Capture status, bit 8 (axis 1), bit 9 (axis 2). 1 = active.	INT16, read-only
enabled	Holds the state of *drive enable*. When used with EtherCAT this reflects the state of power on the drive (Voltage Enabled) and ready to run. Will be set after a 'drive enable' instruction if executes without error.	BYTE8, read-only
fault1 fault2 (not used) fault3 (not used) fault4 (not used)	Fault status words, reference Chapter 8.	BYTE8, read-only
faulted	Set to *true* when a fault has occurred.	BYTE8, read-only
inpos	Holds the state of *in position*. *In Position* is *true* when the target generator is *inactive* and when the position error (*perr*) is within bounds set by *inposw*. When used with EtherCAT this reflects that the drive is "in position" that was commanded.	BYTE8, read-only
overpos	Set to *true* when target position (t*pos*) >= *poslim* or when the associated hardware positive overtravel limit is *active*.	BYTE8, read-only
overneg	Set to *true* when target position (*tpos*) <= *neglim* or when the associated hardware negative overtravel limit is *active*.	BYTE8, read-only
overtrq	Set to *true* when the commanded torque *trqc* has been clamped to the torque limit (either *tmax* or *tlim*). Not used for EtherCAT.	BYTE8, read-only
pstate	Current axis motion state (M3-40): **enum** PSTATE { IDLE, // *ready to run* RUNNING, // *processing sub-steps* COMPLETE, // *done running, awaiting IDLE* STOP, // *stop* SLEWSTOP, // *slewed stop* SLEWING, // *slewing*	BYTE8, read-only

Status Variables	Description	Type

```
    PRESPLINE,    // pre 'SPLINE' move
    PRECAM,       // pre 'CAM' move
    CONT_CAM,     // continue 'CAM' move that was stopped
    INSPLINE,     // in 'SPLINE' move
    INCAM,        // in 'CAM' move
    TABLESTOP,    // stop table
    TRACKING,     // geared
    PRETRACKING   // initialization for TRACKING (geared)
mode
};
```

EtherCAT (IncentiveECAT and M3-41):

```
enum PSTATE {
    IDLE, // ready to run
    RUNNING,      // processing sub-steps
    COMPLETE,     // done running, awaiting IDLE
    STOP,         // stop
    SLEWSTOP,     // slewed stop
    SLEWING,      // slewing
    PRESPLINE,    // pre 'SPLINE' move
    PRECAM,       // pre 'CAM' move
    CONT_CAM,     // continue 'CAM' move that was stopped
    INSPLINE,     // in 'SPLINE' move
    INCAM,        // in 'CAM' move
    TABLESTOP,    // stop table
    TRACKING,     // geared
    PRETRACKING   // initialization for TRACKING (geared)
mode
    EXIT_TRACKING,
    ECAT_COMPLETE_PENDING,
    ECAT_PROFILE_POS_INIT,
    ECAT_PROFILE_POS_STARTING1,
    ECAT_PROFILE_POS_STARTING1A,
    ECAT_PROFILE_POS_STARTING2,          // 20
    ECAT_PROFILE_POS_RUNNING,
    ECAT_PROFILE_POS_WAIT_INPOS,
    ECAT_PROFILE_VEL_INIT,
    ECAT_PROFILE_VEL_WAIT_DELAY1,
    ECAT_PROFILE_VEL_WAIT_DELAY2,
    ECAT_PROFILE_VEL_WAIT_DELAY3,
    ECAT_PROFILE_VEL_WAIT,
    ECAT_PROFILE_TORQUE_INIT,
    ECAT_PROFILE_INIT_CSP,
    ECAT_PROFILE_INIT_INTERPOLATED,
    ECAT_PROFILE_WAIT_CSP1,
    ECAT_PROFILE_WAIT_CSP2,
    ECAT_MODE_WAIT_CSP,                  // 33
    ECAT_MODE_WAIT_INTERPOLATED,
    ECAT_MODE_WAIT_PROFILE, // This just idles, awaiting
                            // for the Profile request to be processed.
                ECAT_PROFILE_INIT_QSTOP,
    ECAT_PROFILE_WAIT_QSTOP1,
```

Status Variables	Description	Type

	ECAT_PROFILE_WAIT_QSTOP2,
	ECAT_PROFILE_WAIT_QSTOP,
	ECAT_PROFILE_AT_VEL, // 40
	ECAT_PROFILE_AT_TORQUE,
	ECAT_HOMING_INIT,
	ECAT_HOMING_STARTING1,
	ECAT_HOMING_STARTING1A,
	ECAT_HOMING_STARTING2, // 45
	ECAT_HOMING_RUNNING,
	ECAT_HOMING_WAIT_INPOS_KOLLMORGEN,
	ECAT_HOMING_WAIT_INPOS,
	ECAT_HOMING_WAIT_INPOS_IAI,
	ECAT_PROFILE_POS_WAIT_ABORTING, // 50
	ECAT_PROFILE_VEL_STOP,
	ECAT_PROFILE_VEL_STOPPING,
	ECAT_HOMING_ABORT_START,
	ECAT_HOMING_ABORT_WAIT, // 54
	ECAT_OFFLINE = -1,
	ECAT_USER_OFFLINE = -2 // User requested offline
	};	

tracking_pstate	EtherCAT only, used mainly for vector moves on axis that is following main axis (Arc and linear interpolation). Same states as 'pstate'. Contains the current execution state of the drive during a 'move on a gear' operation. 'pstate' must be in TRACKING mode for this property to be valid.	INT32, read only
time	A settable, accurate time counter (sec). This is a floating point variable with precision of the control loop period (800uS default, M3-40. 1mS, M3-41/IncentiveECAT). The value will increment by the control loop period value each interrupt cycle. time = time + period;	DOUBLE, read-write
touchProbeStatus	EtherCAT Only: Maps directly to drive object 0x60b9.	INT32, read only
zpulse*	Set to *true* when the Z-pulse has been seen. *Note: Currently this does not work properly and only detects the first Z-pulse. Workaround is to watch for a change in ztheta as an indication of Z-pulse. Not used on EtherCAT, only M3-40.	BYTE8, read-only

Control Variables	Description	Type
acc	Default acceleration rate for absolute and incremental motion. Scaled in user-units/sec/sec.	FLOAT32, read-write

cmode	Control mode – controls the structure of the position/velocity loops.
	
	M3-40:
	
	Torque (0) – Control loop outputs a torque command (velocity loop is active).
	
	Velocity (1) – Control loop outputs a velocity command to the drive (velocity loop is inactive).
	
	Stepper (2) – Control loop outputs step and direction pulses to the drive (velocity loop is inactive).
	
	Open loop (16) – Or this with *Torque* mode for open loop, direct dac control. Write directly to trqc, -10 to 10 (float) representing volts.
	
	M3-41/IncentiveECAT:
	
	`$CYCLIC_SYNC_POSITION_MODE 0 (default)`
	`$PROFILE_VELOCITY_MODE 1`
	`$INTERPOLATED_MODE 3`
	`$PROFILE_POSITION_MODE 4`
	`$PROFILE_TORQUE_MODE 5 (not supported)`
	`$CYCLIC_SYNC_TORQUE_MODE 6 (not supported)`
	`$CYCLIC_SYNC_VELOCITY_MODE 7 (not supported)`
	`$HOMING_MODE 8`
	`$VELOCITY_MODE 9`
	
	This variable cannot be changed while the axis is *enabled*
		BYTE8, read-write

dec	Default deceleration rate for absolute and incremental motion. Scaled in user-units/sec/sec.	FLOAT32, read-write
encoder_mode	EtherCAT Only: Enable Absolute Positioning mode by setting the encoder_mode axis property variable to a 1, prior to executing the 'drive enable' command. This will set the current position to that read from the drive, assuming battery backup, rather than 0. A value of 0 is for an incremental encoder. This variable can also be automatically set by using the axis property pull-down menu option: absolute.	INT32, read-write
gtimebase	A global timebase variable that affects both axes. This variable in conjunction with the per-axis *timebase* sets the	FLOAT32, read-write

Control Variables	Description	Type
	effective per-axis natural time base of the target generator. This parameter should only be set through a reference to the first axis in an MSB or from a QS4 program. Should not be used on M3-41/IncentiveECAT as it will increment the time between control loops used in calculations but not change the control loop period itself. Defaults to 1.0 since the control loop time is: loop time = Tick Period * gtimebase;	
homing_speed1	EtherCAT only: Some homing modes require multiple speeds, with this one being the speed to the switch (object 0x6099.1).	FLOAT32, read-write
homing_speed2	EtherCAT only: Some homing modes require multiple speeds, with this one being the speed to the index (object 0x6099.2).	FLOAT32, read-write
inpos_t	EtherCAT only: Settling time for in position prior to notifying application in milliseconds. This value is written to the drive object 0x6068.0 if it is available.	INT32, read-write
inpos_w	EtherCAT only: In Homing and Profile Position mode this is the in position window that is acceptible prior to acknowledging motor is in position. Object 6067.0 is written to if available. In CSP and other modes the perr must be less than inpos_w in order for inpos to be set as 1. This variable maps to inposw which is available on M3-40 boards.	FLOAT32, read-write
invel_t	EtherCAT only: Time required, in milliseconds, to be at the target velocity, Profile Velocity mode. Maps to object 0x606E.0.	INT32, read-write
invel_w	EtherCAT only: In Profile Velocity mode this is the window velocity is to be maintained in. inpos_w X Target Velocity. Object 606D.0 is written to if available.	FLOAT32, read-write
jerk_a_req	Requested acceleration jerk (default 0), units/sec^3. Jerk (S-curve generation) for absolute and incremental motion (scaled in user-units/sec/sec/sec). If set to zero (0.0), then S-curve generation is disabled. Set to -1 for automatic calculation based on move.	FLOAT32, read-write
jerk_d_req	Requested deceleration jerk (default 0), units/sec^3. Jerk (S-curve generation) for absolute and incremental motion (scaled in user-units/sec/sec/sec). If set to zero (0.0), then S-curve generation is disabled. Set to -1 for automatic calculation based on move.	FLOAT32, read-write
jerk_a	Actual acceleration jerk used, units/sec^3.	FLOAT32, read-only

Control Variables	Description	Type
jerk_d	Actual deceleration jerk used, units/sec^3.	FLOAT32, read-only
newvel	New velocity is used in conjunction with the 'new endposition' command to request a different velocity than is current. If 0 then is ignored.	FLOAT32, read-write
sppr	Steps/rev to output when in stepper mode (when *cmode* is *Stepper*). Not used on M3-41/IncentiveECAT.	INT3, read-write
stoprate	Rate at which to do a slewed stop (uu/sec/sec)	FLOAT32, read-write
theta	Motor angle. In EtherCAT this value is incremented by the position change (+/-) each control loop tick. UINT32.	INT16, read-only
time	Incremented by loop period each interrupt.	DOUBLE, read-write
timebase	Used to override the natural time base of the target generator. When set to 1.0 (the default value), the target generator's "time" is un-scaled. When set to a value between 0.0 and 1.0, the target generator's "time" is slowed-down, effectively generating lower velocities. When set to 0.0, motion stops. *Changing the timebase only effects commanded motions, it does not alter other commands such as delay. Do not change on EtherCAT systems, should stay as default of 1.0.*	FLOAT32, read-write
tlim	M3-40: Torque limit (Nm) – torque command limit. EtherCAT: Does not apply to most EtherCAT systems. Used on some drives in PROFILE TORQUE MODE and by IAI, reference the EtherCAT Applications Guide.	FLOAT32, read-write
tmax	M3-40: Scale factor – maximum torque (Nm) that is generated at the motor when the control loop commands 10V to the drive. This is set using the *property inspector* and cannot be changed in QM code. Consult the connected motor and drive specifications to properly set this value. This property is valid when *cmode* is *Torque*. EtherCAT:	FLOAT32, read-write

Control Variables	Description	Type
	Used on most drives to limit the torque while in CSP mode. Reference the EtherCAT Applications Guide for specifics.	
vmax	M3-40: Scale factor – velocity generated when the control loop commands 10V to the drive. Scaled in RPM (rotational) or linear-units/min (linear). This is set using the *property inspector* and cannot be changed in QM code. Consult the connected motor and drive specifications to properly set this value. This property is valid when *cmode* is *Velocity*. EtherCAT: Used to limit velocity in CSP mode to a maximum. Also to set the maximum velocity for Velocity and Position modes and homing speeds in Homing modes.	FLOAT32, read-write
ztheta	M3-40: Motor angle of Z. EtherCAT: Not used.	INT16, read-only

Tuning Variables	Description	Type
_highBW	Internal use only. Not used with EtherCAT.	BYTE8, read-write
_inertia	Internal use only (tuning inertia). Not used with EtherCAT.	DOUBLE, read-write
_wn	Internal use only (tuning wn). Not used with EtherCAT.	DOUBLE, read-write
_zeta	Internal use only (tuning zeta). Not used with EtherCAT.	DOUBLE, read-write
aff	Velocity-loop acceleration feed-forward gain. Not used with EtherCAT.	FLOAT32, read-write

Tuning Variables	Description	Type
	Scaled as Nm/(rev/sec)/sec or Nm/(linear-unit/sec)/sec of commanded velocity.	
kd	Velocity-loop derivative gain (D). Not used with EtherCAT. Scaled as Nm-sec/(rev/sec)or Nm-sec/(linear-unit/sec) of velocity error.	FLOAT32, read-write
kfilt	A compensation value for the Kalman-filter used in the velocity estimator. Not used for EtherCAT.	FLOAT32, read-write
kgain	A compensation value for the Kalman-filter used in the velocity estimator. Not used for EtherCAT.	FLOAT32, read-write
ki	Velocity-loop integral gain (I). Not used with EtherCAT. Scaled as Nm/(rev/sec)/sec or Nm/(linear-unit/sec)/sec of velocity error.	FLOAT32, read-write
kv	Velocity-loop proportional gain (P). Not used with EtherCAT Scaled as Nm/(rev/sec) or Nm/(linear-unit/sec).	FLOAT32, read-write
kvf	Velocity-loop factor (0.0-1.0). Not used with EtherCAT. When set to 1.0, the velocity loop is a classic PID structure. When set to 0.0, the velocity loop is a classic PDF structure. When set to a value in between, the velocity loop is a combination of both.	FLOAT32, read-write
nonvolatile	Writing a 1 will cause tuning parameters to originate from nonvolatile serial flash instead of the defaults used in the property window when the program was first created. 0 clears this feature (one per axis). Not used for EtherCAT	BYTE8, read-write
pdead	Position-loop dead-band (user-units). if perr <= pdead then perr = 0.	FLOAT32, read-write
pff	Position-loop velocity feed-forward gain (0.0 – 1.0). Not used on EtherCAT.	FLOAT32, read-write
ppg	Position-loop proportional gain. Not used with EtherCAT. Scaled as 1000/min.	FLOAT32, read-write
vff	Velocity-loop velocity feed-forward gain.	FLOAT32, read-write

Tuning Variables	Description	Type
	Scaled as Nm/(rev/sec) or Nm/(linear-unit/sec) of commanded velocity. EtherCAT: Used on some drives to set velocity feed forward value when initiating CSP mode. Not all drives support it, typically object 0x60B1.	

Feedback Variables	Description	Type
camming_invertend	Inverts logic of invertmaster with regards to camming table start position and assumed direction traversing. By default 0, follow invertmaster, 1 to do opposite of invertmaster (!invertmaster).	BYTE8, read-write
encoderZ	Z encoder input. Not used on EtherCAT	INT64, read-only
encoderZ3	Combination of Axis 1 and Axis 2 Z inputs A/B. Not used on EtherCAT.	INT64, read-only
fpos	The feedback position scaled in user-units.	DOUBLE, read-only
fposc	The feedback position scaled in encoder counts.	INT64, read-only
gratio	Present gear ratio.	DOUBLE, read-only
inposw	*In Position* window. Controls when the axis is deemed *in position*. Scaled in user-units. Typically EtherCAT object 0x6067 when supported. Also used for EtherCAT settling to make sure perr is smaller than inposw when the drive has finished move.	DOUBLE, read-write
invertcmd	M3-40: Whether to invert the sign of the command output: 0 = no inversion 1 = invert dac voltage = - dac voltage EtherCAT: Not used.	BYTE8, read-write
invertfeed	Whether to invert the way the feedback encoder counts: 0 = count normally 1 = count inverted	BYTE8, read-write
invertmaster	Whether to invert the way the master encoder counts: 0 = count normally 1 = count inverted	BYTE8, read-write

Feedback Variables	Description	Type
mppr	Master encoder counts per revolution (or per *linear unit* for linear feedback devices).	INT32, read-write
neglim	Negative over-travel limit, scaled in user-units. overneg = tpos <= axistc->neglim;	DOUBLE, read-write
perr	The position error (scaled in user-units).	DOUBLE, read-only
perrlimit	The limit before a *following-error* fault is generated. The variable is scaled in user-units. 0 = disable *following-error* fault check	DOUBLE, read-write
ppr	Feedback encoder counts per revolution (or per *linear unit* for linear feedback devices).	INT32, read-write
poslim	Positive over-travel limit, scaled in user-units. overpos = tpos >= axistc->poslim;	DOUBLE, read-write
runv	Calculated run velocity fed to PID algorithm. Used internally as next tick time velocity.	FLOAT32, read-write
settling	EtherCAT only: Time in seconds to allow things to settle when commanding a new move while already in motion and will be changing directions.	DOUBLE, read-write
sign	Nonzero for SCurve move, 1 for CCW, -1 for CW. EtherCAT also uses it to show direction of all moves.	BYTE8, read-only
stepsout	Stepper pulses output. Not used on EtherCAT.	INT64, read-only
substep	Segmented move current step on. Trapazoidal is 0 to 2, S-Curve is 0 to 6. Used for diagnostics.	BYTE8, read-only
sfmod	The secondary position modulus. Used to control when *sfposc* wraps around to 0.	INT64, read-write
sfpos	Secondary feedback position (in revolutions). sfpos = sfposc * (1/ppr)	DOUBLE, read-only
sfposc	A secondary feedback position (scaled in counts). A separately maintained feedback position similar to *fposc* with the exception that the position will "wrap" (modulo) at 0 and at *sfmod* (unless *sfmod* is set to 0). sfposc = fposc % sfmod;	INT64, read-only

Feedback Variables	Description	Type
tpos	The target position scaled in user-units.	DOUBLE, read-only
tr	EtherCAT Only: Translation ratio. Result of uun/uud used by the program.	DOUBLE, read-only
tposc	The target position scaled in encoder counts.	INT64, read-only
trqc	M3-40: The commanded torque value (Nm). Note that if in torque mode and cmode open loop bit set (bit 5) then this becomes DAC analog output -10 to 10V, floating point. EtherCAT: Used only on velocity drives as torque compensation. Presently not used by any drives, reserved for future use.	FLOAT32, read-only
uud	User-units conversion factor (*denominator*). Motion commands are divided by this value (after multiplying by uun) to scale user-units to revolutions (or *linear unit* or linear feedback devices).	INT32, read-write
uun	User-units conversion factor (*numerator*). Motion commands are multiplied by this value (then divided by uud) to scale user-units to revolutions (or *linear unit* or linear feedback devices).	INT32, read-write
units_ratio	EtherCAT Only: User-units conversion factor (*uun/uud as a double*). If uun is set to 0 then units_ratio will be used for more precise ratio settings then uun/uud.	DOUBLE, read-write
vcmd	Commanded velocity (in rev/sec or linear-units/sec). EtherCAT: Used to set target velocity for drives and modes that require. Presently just MOTIONLINX uses this variable, otherwise set to default of 1.0.	M3-40 - read-only EtherCAT - read-write
vel	Feedback velocity (in rev/sec or linear-units/sec).	FLOAT32, read-only

Feedback Variables	Description	Type
verr	Velocity error (in rev/sec or linear-units/sec). Not used with EtherCAT.	FLOAT32, read-only
zfpos	Holds the last feedback position before it was modified by a "*zero feedback position*" or "*zero following error*" statement. Not used with EtherCAT.	DOUBLE, read-only
ZPULSE_POS	The next Z-pulse location in the positive direction; user-units. Not used with EtherCAT.	DOUBLE, read-only
ZPULSE_NEG	The next Z-pulse location in the negative direction; user-units. Not used with EtherCAT.	DOUBLE, read-only
ztpos	Holds the last target position before it was modified by a "*zero feedback position*" or "*zero following error*" statement. Not used with EtherCAT.	DOUBLE, read-only

IO and Register Variables	Description	Type
ctr0-ctr7	Axis counters (64-bit). These variables count off-to-on transitions of the eight axis-related inputs (5 digital inputs, A, B and Z). **variable** **input (M3-40A/B/C)** ctr0 din1 ctr1 din2 ctr2 din3 ctr3 din4 ctr4 din5 ctr5 A-encoder channel ctr6 B-encoder channel ctr7 Z-encoder channel EtherCAT: See EtherCAT Applications Guide.	INT64, read-write
din1 – din5	The state of digital inputs 1 through 5. EtherCAT: See EtherCAT Applications Guide.	BYTE8, read-only
din6 – din10	The state of digital inputs 6 through 10. *Valid only when the module is in 1½ axis mode* EtherCAT: See EtherCAT Applications Guide.	BYTE8, read-only
dout1 – dout5	The state of digital outputs 1 through 5. EtherCAT: See EtherCAT Applications Guide.	BYTE8, read-only
dout6 – dout10	The state of digital outputs 6 through 10. *Valid only when the module is in 1½ axis mode* EtherCAT: See EtherCAT Applications Guide.	BYTE8, read-only

IO and Register Variables	Description	Type
dins	The state of digital inputs 1 through 5 (or 10 if in 1 ½ axis mode) as a single integer. *Each input has its own binary value starting with 1 for din1, 2 for din2, 4 for din3, 8 for din4, etc. As an example, if din3 and din5 were both on, dins would equal 20.* EtherCAT: See EtherCAT Applications Guide.	INT32, read-only
douts	The state of digital outputs 1 through 5 (or 10 if in 1 ½ axis mode) as a single integer. *Each output has its own binary value starting with 1 for dout1, 2 for dout2, 4 for dout3, 8 for dout4, etc. As an example, if dout3 and dout5 were both on, douts would equal 20.* EtherCAT: See EtherCAT Applications Guide.	INT32, read-only
driveenable	The digital output number to use for "drive enable." Positive input number for true state=high Negative number for true state=low 0 = use no output *When an output is assigned for use as drive enable, all set/clear operations to that output are ignored.*	BYTE8, read-write
global_flag1 to global_flag5	EtherCAT only: 5 general use global flag registers directly accessible by all MSB's and QuickBuilder program.	DOUBLE, read-write
global_inputs	M3-41 Only: Local I/O is present on the 5300 M3-41 EtherCAT module. This module has 6 inputs which is global to all MSBs. The MSB property 'global_inputs' is used to read the 6 inputs, with the first bit being the first input.	INT32, read-only
global_outputs	M3-41 Only: Local I/O is present on the 5300 M3-41 EtherCAT module. This module has 2 outputs which are global to all MSBs. The outputs are referenced as 9 and 10 when using the 'setout'/'clrout' instructions. The MSB property 'global_outputs' can be used in addition to 'setout'/'clrout' for read/write operations of the local outputs.	INT32, read-write
global_reg1 to global_reg32	EtherCAT only: 32 general use global registers directly accessible by all MSB's and QuickBuilder program.	DOUBLE, read-write
overposin	The digital input number to use for positive over-travel. Positive input number for true state=high Negative number for true state=low 0 = disable positive over-travel checking	BYTE8, read-write
overnegin	The digital input number to use for negative over-travel. Positive input number for true state=high Negative number for true state=low	BYTE8, read-write

IO and Register Variables	Description	Type
	0 = disable negative over-travel checking	
running	The digital output number to use for "MSB active" (running). 0 = use no output *When an output is assigned for use as "MSB active" (running), all set/clear operations to that output are ignored.*	BYTE8, read-write

Tracking Variables	Description	Type
antibackup	Whether or not to allow the slave to generate geared pulses in response to negative displacements of the master 0 = allow generated pulses in all cases 1 = accumulate negative displacements of the master and generate geared slave pulses when accumulated total > 0	BYTE8, read-write

master_feedback	EtherCAT Only:
	Any axis can track another axis by simply dropping into tracking mode. To reference which axis to track, use the variable 'master_feedback', which by default is 1. Set this variable to the axis you wish to track, set whether to reference fpos or tpos of the master axis, and then drop into tracking mode.
	
	// ******** TRACKING MODE **********
	[SetTracking]
	zero master counters;
	master_feedback = 1;
	// Set we will track axis 1
	// Set the feedback mode first so when enter tracking it is
	// referencing correct master. 'set master feedback' references
	// the other axis fposc or feedback position.
	// 'set master target1' references the other axis tpos position.
	set master feedback;
	set mode tracking; // Enter tracking mode
	
	M3-41:
	Another option, only available on the M3-41 hardware module, is the provision for directly connecting up to 3 local quadrature encoders. These encoders can be used as master references by the EtherCAT axis. To reference these encoders, the master_feedback variable is set to 1001, 1002, or 1003, for each of the respective encoder inputs. Once master_feedback is referencing a local encoder, its present value will appear in 'mpos' and 'mposc' MSB variables. In addition, all local encoder counts can be accessed using the 'ctr' array, index 5 to 7 from an MSB, or ctr5, ctr6, and ctr7 from QuickBuilder.
	
	Master_feedback = 1001, ctr[5]/ctr5, P1 connector pins:
		INT32, read-write

Tracking Variables	Description	Type
	P1-19 A0+ P1-20 A0- P1-21 B0- P1-22 B0+ Master_feedback = 1002, ctr[6]/ctr6, P1 connector pins: P1-23 A1+ P1-24 A1- P1-25 B1- P1-26 B1+ Master_feedback = 1003, ctr[7]/ctr7, P1 connector pins: P1-27 A2+ P1-28 A2- P1-29 B2- P1-30 B2+	
mcinv	The bit-oriented variable controls when *mposc1-5* are cleared. Bit 0, the least-significant bit, controls *mposc1*. Bit 1, the next significant bit controls *mposc2*, etc. Bit 16 to bit 20 controls whether mposc# is cleared upon entering *tracking* mode. If set cleared. Bit 21 to bit 22 controls whether tsc1/tsc2 is cleared upon entering *tracking* mode. If set cleared.	INT32, read-write
mdelta1	Master position delta, counts This variable holds the displacement that occurred in the master encoder *between* position captures. Essentially this is the last value of *mposc1* before *mposc1* is cleared due to a position capture. Cleared upon entering *tracking* mode.	INT32, read-only
mdelta2	Master position delta, counts This variable holds the displacement the occurred in the master encoder *between* position captures. Essentially this is the last value of *mposc2* before *mposc2* is cleared due to a position capture. Cleared upon entering *tracking* mode	INT32, read-only

Tracking Variables	Description	Type
mdelta3	Master position delta, counts This variable holds the displacement the occurred in the master encoder *between* position captures. Essentially this is the last value of *mposc3* before *mposc3* is cleared due to a position capture. Cleared upon entering *tracking* mode	INT32, read-only
mdelta4	Master position delta, counts This variable holds the displacement the occurred in the master encoder *between* position captures. Essentially this is the last value of *mposc4* before *mposc4* is cleared due to a position capture. Cleared upon entering *tracking* mode	INT32, read-only
mdelta5	Master position delta, counts This variable holds the displacement the occurred in the master encoder *between* position captures. Essentially this is the last value of *mposc5* before *mposc5* is cleared due to a position capture. Cleared upon entering *tracking* mode	INT32, read-only
mmc	Master position modulus (0=functionality disabled) This variable is used as a *modulus* for the variables *mposc1-5 and tmod*. Whenever updated, *mmc* is applied by formula: *mposc1* = *mposc1* **mod** *mmc* *mposc2* = *mposc2* **mod** *mmc* *mposc3* = *mposc3* **mod** *mmc* *mposc4* = *mposc4* **mod** *mmc* *mposc5* = *mposc5* **mod** *mmc* *tmodc* = *tmodc* **mod** *mmc*	INT32, read-write
mposc	Master position, counts This variable is cleared when the mode is changed to *tracking*. This variable is unaffected by *mmc* or changes to *mposc1-5*. This counter rolls over at 65536 times the value of *mppr* when in *tracking* mode.	INT64, read-write
mposc1	Master position, counts (modulo by *mmc*)	INT32, read-only

Tracking Variables	Description	Type
	This variable is cleared when the mode is changed to *tracking*. This variable is cleared when input #1 makes an *off-to-on* transition unless the 0-bit in *mcinv* is set in which case this variable is cleared when the input makes an *on-to-off* transition. Bit 16 in *mcinv* set will disable clearing upon entering *tracking* mode.	
mposc2	Master position, counts (modulo by *mmc*) This variable is cleared when the mode is changed to *tracking*. This variable is cleared when input #2 makes an *off-to-on* transition unless the 1-bit in *mcinv* is set in which case this variable is cleared when the input makes an *on-to-off* transition. Bit 17 in *mcinv* set will disable clearing upon entering *tracking* mode.	INT32, read-only
mposc3	Master position, counts (modulo by *mmc*) This variable is cleared when the mode is changed to *tracking*. This variable is cleared when input #3 makes an *off-to-on* transition unless the 2-bit in *mcinv* is set in which case this variable is cleared when the input makes an *on-to-off* transition. Bit 18 in *mcinv* set will disable clearing upon entering *tracking* mode.	INT32, read-only
mposc4	Master position, counts (modulo by *mmc*) This variable is cleared when the mode is changed to *tracking*. This variable is cleared when input #4 makes an *off-to-on* transition unless the 3-bit in *mcinv* is set in which case this variable is cleared when the input makes an *on-to-off* transition. Bit 19 in *mcinv* set will disable clearing upon entering *tracking* mode.	INT32, read-only
mposc5	Master position, counts (modulo by *mmc*) This variable is cleared when the mode is changed to *tracking*. This variable is cleared when input #5 makes an *off-to-on* transition unless the 4-bit in *mcinv* is set in which case this variable is cleared when the input makes an *on-to-off* transition. Bit 20 in *mcinv* set will disable clearing upon entering *tracking* mode	INTE32, read-only
move_master_counts	Move master counts target if not forever.	INT32, read-only
move_master_rate_target	Move master rate target setting (virtual master).	INT32, read-only
move_master_ramp	Move master ramp setting (virtual master)	INT32, read-only

Tracking Variables	Description	Type
move_master_rate	Move master rate setting (virtual master)	INT32, read-only
mpgai	Master position during *gear…at…in*, counts This variable holds the number of consumed master position counts during the last *gear…at…in* statement	INT32, read-only
mpgfi	Master position during *gear…for…in*, counts This variable holds the number of consumed master position counts during the last *gear…for…in* statement	INT32, read-only
sdc	Slave decrement counter This counter decrements for every output slave count whereas *tsc1* and *tsc2* increment.	INT32, read-write
spgai	Slave position during *gear…at…in*, counts This variable holds the number of consumed slave position counts during the last *gear…at…in* statement.	INT32, read-only
spgfi	Slave position during *gear…for…in*, counts This variable holds the number of consumed slave position counts during the last *gear…for…in* statement.	INT32, read-only
smodc	Slave position counter. This variable is cleared when the mode is changed to *tracking*.	INT32, read-only
smod	Slave position modulus This variable is used as a *modulus* for the variable smodc. Whenever *smodc* is updated, *smod* is applied by formula: `smodc = smodc mod smod`	INT32, read-write
smark	Slave modulo marker position, counts When an input transistions in conjunction with the bits specified in *smarkrise* and *smarkfall*, this variable is computed by formula: `smark = sphase - smodc`	INT32, read-only
smarkrise	This bit-oriented variable controls when *smark* is calculated. When the input corresponding to a set bit in *smarkrise* makes an off-to-on transition, *smark* is calculated. Bit 0, the least significant bit, represents input #1, etc.	INT32, read-write

Tracking Variables	Description	Type
smarkfall	This bit-oriented variable controls when *smark* is calculated. When the input corresponding to a set bit in *smarkfall* makes an on-to-off transition, *smark* is calculated. Bit 0, the least significant bit, represents input #1, etc.	INT32, read-write
sphase	Slave marker position phase, counts Used to offset *smark*.	INT32, read-only
tmc1	Temporary master position counter 1 This variable is auxiliary, settable counter that tracks master position. These variables can be zeroed atomically by *zero master counters*. Cleared when changing mode to TRACKING.	INT32, read-write
tmc2	Temporary master position counter 2 This variable is auxiliary, settable counter that tracks master position. These variables can be zeroed atomically by *zero master counters*. Cleared when changing mode to TRACKING.	INT32, read-write
tmodc	Temporary master position counter mod mmc. User cleared only.	INT32, read-write
tsc1	Temporary slave position, counter 1 This variable is auxiliary, settable counter that tracks slave position.	INT32, read-write
tsc1rise	This bit-oriented variable controls when *tsc1* is cleared. When the input corresponding to a set bit in *tsc1rise* makes an off-to-on transition, *tsc1* is cleared. Bit 0, the least significant bit, represents input #1, etc.	INT32, read-write
tsc1fall	This bit-oriented variable controls when *tsc1* is cleared. When the input corresponding to a set bit in *tsc1fall* makes an on-to-off transition, *tsc1* is cleared. Bit 0, the least significant bit, represents input #1, etc.	INT32, read-write
tsc2	Temporary slave position, counter 2 This variable is auxiliary, settable counter that tracks slave position.	INT32, read-write
tsc2rise	This bit-oriented variable controls when *tsc2* is cleared. When the input corresponding to a set bit in *tsc2rise* makes an off-to-on transition, *tsc2* is cleared. Bit 0, the least significant bit, represents input #1, etc.	INT32, read-write

Tracking Variables	Description	Type
tsc2fall	This bit-oriented variable controls when *tsc2* is cleared. When the input corresponding to a set bit in *tsc2fall* makes an on-to-off transition, *tsc2* is cleared. Bit 0, the least significant bit, represents input #1, etc.	INT32, read-write
tracking_pstate	EtherCAT Only: See Status Variables.	INT32, read_only
tracking_sign	EtherCAT Only: If the axis is tracking this variable is set to 1 or -1 depending on the direction of rotation.	INT32, read_only

tracking_status	EtherCAT Only: If an axis is tracking this variable represents the current state of motion:
	
	enum QS2STATE {
	NOT_INITIALIZED,
	STOPPED_READY,
	WAIT_START,
	ACCELERATING,
	AT_MAX_SPEED,
	DECEL_TO_NEW_MAX_SPEED,
	DECEL_TO_STOP,
	SOFT_STOP,
	REG_MOVE,
	HOME,
	NOT_USED1,
	FOLLOWING,
	NOT_USED2,
	COMMAND_ACCEPTED
	};
		BYTE8, read-only

tracking_tpos	EtherCAT Only: If an axis is tracking then this is the target position scaled in user-units.	DOUBLE, read-only
tracking_tposc	EtherCAT Only: If an axis is tracking then this is the target position scaled in encoder counts.	INT64, read-only
vmdelta	Virtual master delta counts.	INT32, read-only

Capture Variables	Description	Type
capArmed	Capture armed, non zero. If capture is armed and this variable is cleared any capture will be ignored (equivalent to disabling capture).	BYTE8, read-write
capEdge	Edge to monitor for capture as set by the 'set capture' instruction. 2 – any edge, 1 – rising edge, 0 – falling edge.	INT16, read-only
capGate	Capture input used to gate the trigger input, if -1 then always gated.	INT16, read-write

capGateState	Gate active as on or off, 0 it waiting for gate to be high, 1 if waiting for gate to be low.	INT16, read-only
capInput	Capture input to be used as a trigger.	INT16, read-only
capLimit	Capture limit value	INT64, read_write
capLimitflag	Set if capture limit occured	BYTE8, read-only
capMod	EtherCAT only: Defaults to 1 and applies a modulus to the captured value.	INT32, read-write
capOffset	Amount to move after a capture occurs, fposc + capOffset = new end position. If 0 then move continues as was originally instructed.	INT64, read-write
cappos	Capture position in user units. capposc * 1/ppr.	DOUBLE, read-only
capposc	Capture position in counts. This will be fposc + encoder offset if not in Tracking mode. If Tracking then sfposc + encoder offset. Latched when defined capture input goes active.	INT64, read-only
capStatus	Capture status, bit 8 (axis 1), bit 9 (axis 2). 1 = active.	INT16, read-only
capTriggered	Capture occurred flag, non zero. capposc/cappos contains the latched position when occurred.	BYTE8, read-write
capWait	If 1, an MSB is waiting on a 'wait capture' instruction, else 0.	BYTE8, read-only
capwaitBranch	Capture MSB offset branch value.	INT16, read-only
capwinEnd	End of capture range as set by the 'set capwin' instruction. If same as capwinStart then no window is active.	INT64, read-write
capwinStart	Start of capture range as set by the 'set capwin' instruction. If same as capwinEnd then no window is active.	INT64, read-write
capwinType	Capture window type (0-17): **fposc** (0) feedback position **mposc1 - mposc5** (1-6) master position counters #1 through #5 **mposc** (7) master position counter **smodc** (8) slave position (modulo) **smark** (9) slave marked position **tmc1 tmc2** (10/11) temporary master counters #1 & #2 **tsc1 tsc2** (12/13) temporary slave counters #1 & #2 **sdc** (14) slave decrement counter **fposc1** (15) feedback position of axis 1 (fposcA) **fposc2** (16) feedback position of axis 2 (fposcB) **tmodc** (17) temporary master counter mod mmc **sfposc** (18) secondary feedback position of axis EtherCAT: Reference EtherCAT Application Guide.	INT16, read-only
msource	Master source setting: 0x01 – feedback1 0x02 – feedback2	BYTE8, read-only

	0x03 - feedbackz 0x04 – target1 0x05 – target2 0x06 – common 0x07 - virtual Bit OR of above: 0x80 - global EtherCAT: Reference EtherCAT Application Guide.	

Diagnostic Variables	Description	Type
activeBG_MSBs	Number of active background MSB's running on axis.	BYTE8, read-only
activeFG_MSBs	Number of active foreground MSB's running on axis.	BYTE8, read-only
debugTable	Cam table to view, from 0 to 5	INT32, read-write
debugTableRows	Number of rows presently in the selected cam table, debugTable.	INT32, read-only
debugTableRow	Current row number to view in the selected cam table, debugTable.	INT32, read-write
debugTableX	X value for selected debugTableRow.	FLOAT32, read-only
debugTableY	Y value for selected debugTableRow.	FLOAT32, read-only
last_ALStatusCode	EtherCAT Only: Last AL Status code read from drive.	INT32, read-write
last_errorCode	EtherCAT Only: Last error code read from drive, typically object 0x603F.	INT32, read-write
last_errorRegister	EtherCAT Only: Last error register from drive, typically 0x1001.	INT32, read-write
last_errorType	EtherCAT Only: Used internally by M3-41/IncentiveECAT.	INT32, read-write
lastOverall	Last full loop time of all axis in uS. EtherCAT: Not used.	FLOAT32, read-only
loopperiod	Periodic motion loop interrupt time in uS.	FLOAT32, read-only
looprate	Number of motion loop interrupts/second.	FLOAT32, read-only

minLoopTime	Minimum actual individual axis loop execution time (uS) reached. EtherCAT: Not used.	FLOAT32, read-only
maxLoopTime	Maximum actual individual axis loop execution time (uS) reached. EtherCAT: Not used.	FLOAT32, read-only
minOverall	Minimum actual axis loop execution time (uS) reached for all axis. EtherCAT: Not used.	FLOAT32, read-only
maxOverall	Maximum actual axis loop execution time (uS) reached for all axis. EtherCAT: Not used.	FLOAT32, read-only
overflowFlag	Motion loop overflow flag, set to 1 if loop time exceeded while in the loop (800uS default, reference 'set loopperiod') EtherCAT: Not used.	INT32, read-only
scanning	EtherCAT Only: Variable set according to EtherCAT scan state. 0 = not scanning, 1= initializing, 2 = operational and scanning.	INT32, read-only
wStatus	EtherCAT Only: Status Word read from the drive, object 0x6041.	INT16, read-only
wControlWord	EtherCAT Only: Control Word used to issue commands to the drive, object 0x6040.	INT16, read-only

Setup Variables	Description	Type
axisptr	EtherCAT Only: Properties such as fpos, tpos, etc., are local to an axis and not shared with other axis MSBs. This limitation can be overridden by using the 'axisptr' property of an MSB. This property controls what axis the MSB will retrieve its property value on a read and write operation. It is typically set to the value of 'axisnum', which is the axis number of that axis executing the MSB. Setting this axis number to any other value will override what axis the property is retrieved from.	INT32, read-write
dim_factor_num	EtherCAT Only: Reserved for later use with velocity drives.	INT32, read-write
dim_factor_denom	EtherCAT Only: Reserved for later use with velocity drives.	INT32, read-write
dwSlaveID	EtherCAT Only: Slave ID for this axis.	INT32, read-only
eCAT_driveType	EtherCAT Only: The type of drive and axis number that an MSB is executing can be referenced programmatically via the 'eCAT_driveType' and	INT32, read-only

Setup Variables	Description	Type
	'axisnum' property variables. 'axisnum' contains the axis number, where 1 is the first. 'eCAT_driveType' is defined as follows: $DRIVE_COPLEY 2 $DRIVE_YASKAWA 3 $DRIVE_ELMO (not supported) 4 $DRIVE_KOLLMORGEN 5 $DRIVE_SANYO_DENKI 6 $DRIVE_EMERSON 7 $DRIVE_AMC 8 $DRIVE_VIRTUAL 9 $DRIVE_IAI_ACON_MODE3 11 $DRIVE_ABB_MICROFLEX 12 $DRIVE_ABB_MITSUBISHI 13 $DRIVE_ABB_PANASONIC 14 $DRIVE_ABB_LINMOT 15 $DRIVE_MOTIONLINX 17 $DRIVE_WAGO_ENCODER_631 18 $DRIVE_WAGO_ENCODER_637 19	
eCAT_manufID	EtherCAT Only: EtherCAT ESI manufacturer ID.	INT32, read-only
eCAT_productCode	EtherCAT Only: EtherCAT ESI product code.	INT32, read-only
qs_decel_time	EtherCAT Only: Reserved for future VFD quick stop deceleration time.	INT32, read-write
vel_accel_time	EtherCAT Only: Reserved for future VFD velocity acceleration time.	INT32, read-write
vel_decel_time	EtherCAT Only: Reserved for future VFD velocity deceleration time.	INT32, read-write
vmin	EtherCAT Only: Reserved for future VFD velocity minimum.	FLOAT32, read-write

RFID Variables	Description (EtherCAT Only)	Type
RFID_channel	Selects the RFID channel to be operated on by the properties that follow. Entries of 1 to RFID_totalChannels are the valid selections, with 0 disabling access. All properties should be initialized to their proper values before setting the RFID_channel to a non-zero value.	INT32, read-write
RFID_totalChannels	Represents the total number of RFID channels available in the system.	INT32, read-only
RFID_state	Represents the current state of the RFID interface logic state machine as it executes any requests issued by the RFID_control property variable. Possible values are as follows: RFID_OFF 0 RFID_IDLE 1 RFID_READING_1 2	INT32, read-write

RFID Variables	Description (EtherCAT Only)	Type
	RFID_READING_2 3 RFID_READING_WAIT_DONE 4 RFID_WRITING_1 10 RFID_WRITING_2 11 RFID_WRITING_3 12 RFID_WRITING_4 13 RFID_WRITE_DONE 15 RFID_WRITE_WAITTAG 20 RFID_READ_WAITTAG 21 RFID_WAITNOTAG 22 RFID_ERROR 30	
RFID_error	Turck specific error where bits 7 to 0 represent the category and bits 15 to 8 are the description. Any time the RFID_error property is non-zero an error is present. To clear the error the RFID reader must be reset using the RFID_control property RESET bit. See EtherCAT Applications Guide for error codes..	INT32, read-only
RFID_address	This address is forwarded to the RFID reader to determine where in its memory block to begin accessing data for read and write operations. A value of 0 is the first address. RFID_address is auto-incremented after any read or write by the amount in RFID_bytesTransferred therefore set it back to the desired start location after each read or write.	INT32, read-write
RFID_index	The index is used to select which RFID_data_readl/RFID_data_readh or RFID_data_writel/RFID_data_writeh array item is to be operated on. Where 0 is the first item, up to 31 (32 array items for 256 bytes total possible). int RFID_data_readl[32]; int RFID_data_readh[32]; int RFID_data_writel[32]; int RFID_data_writeh[32];	INT32, read-write
RFID_data_readl	The first 32 bit integer or 4 bytes of data transferred from the RFID tag. This property is an array of 32 deep, indexed by the RFID_index property. int RFID_data_readl[32];	INT32, read-write
RFID_data_readh	Read/write, the second 32 bit integer or 4 bytes of data transferred from the RFID tag. This property is an array of 32 deep, indexed by the RFID_index property in parallel to RFID_data_readl. int RFID_data_readh[32];	INT32, read-write
RFID_data_writel	Read/write, the first 32 bit integer or 4 bytes of data transferred to the RFID tag. This property is an array of 32 deep, indexed	INT32, read-write

RFID Variables	Description (EtherCAT Only)	Type
	by the RFID_index property. int RFID_data_writel[32];	
RFID_data_writeh	The second 32 bit integer or 4 bytes of data transferred to the RFID tag. This property is an array of 32 deep, indexed by the RFID_index property in parallel to RFID_data_writel. int RFID_data_writeh[32];	INT32, read-write
RFID_status	32 bit integer with only the first 8 bits reflecting the status as returned by the Turck RFID reader. • Done – (Bit 7) Slice is ready to receive command. This bit will be off until previous command bit is turned off. • Busy – (Bit 6) Slice is currently processing command. This is normally on when transceiver is waiting for a tag to be presented. • Error – (Bit 5) Slice has encountered an error during last command. Refer to Error_Cat and Error_Desc for details. This bit is not always set so check RFID_error for nonzero. • Trans_Conn – (Bit 4) Transceiver is correctly connected and communicating with the slice. • Trans_On – (Bit 3) Transceiver has been turned on by slice. • TP – (Bit 2) Tag present; Tag is present in transceiver field. LED on transceiver will blink rapidly. • TFR – (Bit1) Tag Fully Read; Tag has been present in transceiver field long enough so that entire tag memory has been stored in buffer. This bit does not need to be on to indicate a command has been completed.	INT32, read-only
RFID_control	32 bit integer which is used to request RFID transactions to occur, read and writing different aspects of the tag and transceiver. Some of the bits are defined by Turck but have been enhanced by CTC for additional features.	INT32, read-write
RFID_controlActive	Represents the value actually being transferred to the Turck RFID controller at any moment. During operation bits are set/cleared automatically by the M3-41 module, especially when transferring multiple blocks of data. Useful for diagnostic purposes.	INT32, read-only
RFID_count	Represents the value actually being transferred to the Turck RFID controller as the needed byte count. This property is automatically set based upon the RFID_bytesTotal required. Useful for diagnostic purposes.	INT32, read-only
RFID_bytesTotal	This property must be set to the total number of bytes to be transferred to/from the RFID reader. If to the reader RFID_data_writel/h array is used, if from the reader RFID_data_readl/h is used. For a single transfer this is typically set to 8. This property is also used in conjunction with the 'host read' and 'host write' commands when transfers are done	INT32, read-write

RFID Variables	Description (EtherCAT Only)	Type
	with the RFID_data_readl and RFID_data_writel properties. These properties can be used to transfer strings to/from Quickbuilder variants.	
RFID_bytesTransferred	This property represents the number of bytes that have been transferred during a read/write RFID operation as well as 'host read' and 'host write'. If a 'host read' is used to read a QuickBuilder variant string this property will represent the length of the string after the access.	INT32, read-write
RFID_tagIDl	This property is automatically set when a TAG ID read operation is performed. This property represents the lower 4 bytes.	INT32, read-write
RFID_tagIDh	This property is automatically set when a TAG ID read operation is performed. This property represents the upper 4 bytes.	INT32, read-write
RFID_lasttagIDl	This property is automatically set when a TAG ID read operation is performed and a unique id is found, different than that previously read. This property represents the lower 4 bytes. The RFID_lasttagIDl/h is checked against the latest tag id read to ensure no duplicates are found, if that option is enabled.	INT32, read-write
RFID_lasttagIDh	This property is automatically set when a TAG ID read operation is performed and a unique id is found, different than that previously read. This property represents the upper 4 bytes. The RFID_lasttagIDl/h is checked against the latest tag id read to ensure no duplicates are found, if that option is enabled.	INT32, read-write

Vector Variables	Description (EtherCAT Only)	Type
accX	Calculated acceleration along the X axis of the last vector move.	DOUBLE, read-only
accY	Calculated acceleration along the Y axis of the last vector move.	DOUBLE, read-only
accZ	Calculated acceleration along the Z axis of the last vector move.	DOUBLE, read-only
accVector	Circular Interpolation: Acceleration in radians/second^2 that is being used for the calculated profile.	DOUBLE, read-only
angle	Circular Interpolation: Initialized to 0 and records the calculated angle as it sweeps. 2D and 3D Linear Interpolation: Calculated angle of the last vector move.	FLOAT32, read-only
angleStart	Circular Interpolation: The angle at which motion should start where 0 is vertical on the Y axis, minus angle moves left, and positive angle moves right. The 'angleSweep' variable is added to this angle.	DOUBLE, read-write

Vector Variables	Description (EtherCAT Only)	Type
angleSweep	Circular Interpolation: The desired amount of angular motion that is to occur relative to the radius center point. A positive angleSweep rotates clockwise, negative, counter clockwise.	DOUBLE, read-write
axisY	The axis number, from 1 to N, which will be the Y axis, commanded from the X axis. The Y axis must be set for either 2D or 3D interpolation to occur.	INT32, read-write
axisZ	3D Linear Interpolation: The axis number, from 1 to N, which will be the Z axis, commanded from the X axis.	INT32, read-write
decVector	Circular Interpolation: Deceleration in radians/second2 that is being used for the calculated profile.	DOUBLE, read-only
decX	Calculated deceleration along the X axis of the last vector move.	DOUBLE, read-only
decY	Calculated deceleration along the Y axis of the last vector move.	DOUBLE, read-only
decZ	Calculated deceleration along the Z axis of the last vector move.	DOUBLE, read-only
magnitude	2D and 3D Linear Interpolation: Calculated size of the last vector move.	DOUBLE, read-only
radius	Circular Interpolation: The radius in user units of the arc to be drawn. A negative radius flips the arc.	DOUBLE, read-write
vectorY	Circular Interpolation: The calculated center of the arc for the Y axis will be stored here for diagnostic reference, in machine units. Make sure you update vectorY after a circular move if the next move is linear interpolation. 3D Linear Interpolation: The desired Y position on an X/Y/Z grid in user units, based upon revolutions. Note that this value is overwritten after a circular interpolated move for diagnostic purposes. 2D Linear Interpolation: The desired Y position on an X/Y grid in user units, based upon revolutions.	DOUBLE, read-write
vectorZ	Circular Interpolation: The calculated center of the arc for the X axis will be stored here for diagnostic reference, in machine units. 3D Linear Interpolation: The desired Z position on an X/Y/Z grid in user units, based upon revolutions. Note that this value is overwritten after a circular interpolated move for diagnostic purposes.	DOUBLE, read-write

Vector Variables	Description (EtherCAT Only)	Type
velVector	Circular Interpolation: Velocity in radians/second that is being used for the calculated profile.	DOUBLE, read-only
`velX`	Calculated velocity along the X axis of the last vector move.	DOUBLE, read-only
`velY`	Calculated velocity along the Y axis of the last vector move.	DOUBLE, read-only
`velZ`	Calculated velocity along the Z axis of the last vector move.	DOUBLE, read-only

6.3 Host Register Access

The Host Read/Write commands are used to directly access all the main controller's registers, including variant storage. These registers consist of, but are not limited to:

- Analog I/O
- Digital I/O
- Data tables
- Volatile and non-volatile Variant scalar, vector and tables
- Generic integer registers
- Non-volatile register
- Communications

Reference the Quickstep Register Guide for a summary of available registers:

http://www.ctc-control.com/customer/techinfo/docs/5300_951/951-530006.pdf

Summary:

host read *variable, register {, row, column}*
host write *variable, register {, row, column}*

Host Read	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
host read *variable, register {, row, column }*				
parameters				
variable	Local user variable or axis property to have 'register' stored to it.			
register	Main controller QuickBuilder register as defined in the Model 5300 Quick Reference Register Guide. May be constant or variable access.			
row	Optional row used only for variant register table access. May be constant or variable access.			
column	Optional column used only for variant register table access. May be constant or variable access.			

This statement pauses execution of the MSB while the contents of a QuickBuilder register is retrieved from the main processor. The register value is then stored into the local 'variable' or axis 'property'. The data type will automatically be converted to that of the local storage. Both integer based registers and variant vectors and tables are supported. When reading a variant, one cell at a time in the table (if any) is read. If no row or column is specified, 0 is assumed.

```
// Read the controller tick timer referencing a variable
// and store to 'userVar'
reg = 13002;
host read userVar, reg;
// Read the controller tick timer using constant register
// number and store to 'userVar'
host read userVar, 13002;
```

Host Write	☑ Positioning	☑ Slewing	☑ Tracking	☑ BG MSB ☑ FG MSB
syntax				
host write *variable, register {, row, column }*				
parameters				
variable	Local user variable or axis property to store to controller 'register'.			
register	Main controller QuickBuilder register as defined in the Model 5300 Quick Reference Register Guide. May be constant or variable access.			
row	Optional row used only for variant register table access. May be constant or variable access.			
column	Optional column used only for variant register table access. May be constant or variable access.			

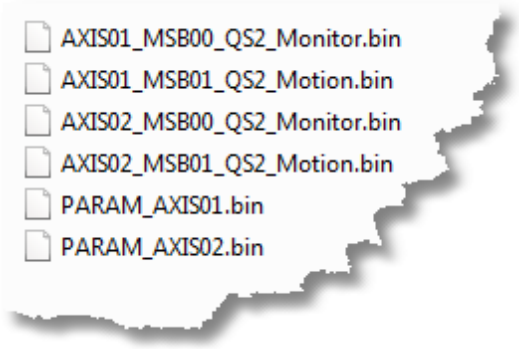
This statement pauses execution of the MSB while the contents of a local 'variable' or axis 'property' is written to a QuickBuilder register on the main processor. The data type will automatically be converted to that of the QuickBuilder register, thus double will be converted to integer, etc. Both integer based registers and variant vectors and tables are supported. When writing a variant, one cell at a time in the table (if any) is written. If no row or column is specified, 0 is assumed.

```
// Clear the controller tick timer, register 13002,
// referencing the contents of 'userVar'.
userVar = 0;
reg = 13002;
host write userVar, reg;
// Clear the controller tick timer, register 13002,
// using a constant value.
host write 0, 13002;
```

7 Chapter 7: Quickstep Support

QuickMotion has been designed for high integration with the QuickBuilder language and include such features as program interaction and user units. A legacy product, Quickstep, uses a register interface for motion control. This interface is not as tightly coupled but there is a large existing code base thus an MSB emulation mode has been created which allows the M3-40 module to appear as a 2219 motion control card, used on the 2700 series controllers, or 5140, within the 5100/5200 controller family.

Register emulation is always available from a read only perspective. In order to fully support the emulation mode a special MSB has been created and must be loaded. This MSB is the output of a QuickBuilder project where initial parameters and any minor MSB customization can be made. To simplify initial use a fully compiled project is available that can be loaded into a 5300 controller for Quickstep program support, QS2MSB. This project is available from the download portion of CTCs' web site. In the example provided, 2 axis, are supported. To support more axis simply add the card to the QuickBuilder project as well as 'start axis' references, or if QuickBuilder is not available, you may simply copy and rename the files with the appropriate axis names. Note the files which are available after compiling QS2MSB, within the controller sub-directory:



These files consist of the binary output, generated by QuickBuilder, for MSB's (AXIS??_MSB??_msbname.bin) and their respective configuration parameters (PARAM_AXIS??.bin). They must be placed in the controller '/_system/Programs/Motion' subdirectory. Upon power up or reset the M3-40A module will automatically look in this directory and if the files are present then an auto-boot sequence will begin. Namely, the files will be loaded into the M3-40 card and automatically executed. If a servo has been tuned and parameters saved to the card, the PARAM file settings will be ignored and only the MSB binary file will be loaded.

Once loaded and running most legacy Quickstep motion applications will run, unchanged.

⚠ Note that the emulation will appear similar to a 2700, 2219 module. Any extended features available within the 5100/5200 controller 5140 module are not currently supported. The Register interface is fully available on all motion products. If only Registers are desired the above .bin files do not have to be loaded. The .bin files are used to simulate full Quickstep motion instructions thus if MSB's are used in a normal QuickBuilder program, they are not needed.

7.1 Registers

Most of the motion registers supported by the 2700/5100/5200 controller are available within the 5300/Incentive environment, regardless of whether the emulation mode is run or not. If emulation is not running then write operations are not supported. These registers consist of:

<table>
<tr><th colspan="2">Motion Registers Grouped by function then axis</th></tr>
<tr><td></td><td>The 5300 firmware is designed to access up to 16 axes. For the 14XXX register values below substitute the axis number for 'ax' to get the correct register. Axis #1 = 1; For example the position of axis #1 is stored in register 14001.</td></tr>
<tr><td>140ax</td><td>Position (counts), R only [QuickBuilder reference = fposc]</td></tr>
<tr><td>141ax</td><td>Error (counts), R only [QuickBuilder reference = perr * ppr * (uun/uud)]</td></tr>
<tr><td>142ax</td><td>Velocity (counts / sec), R only [QuickBuilder reference = vel * ppr * (uun/uud)]</td></tr>
<tr><td>143ax</td><td>Status, R only:
<table>
<tr><th>Status</th><th>Description</th></tr>
<tr><td>0</td><td>Axis not initialized</td></tr>
<tr><td>1</td><td>Stopped and ready</td></tr>
<tr><td>2</td><td>Motion imminent: waiting for start</td></tr>
<tr><td>3</td><td>Accelerating</td></tr>
<tr><td>4</td><td>At max speed.</td></tr>
<tr><td>5</td><td>Decelerating to new max speed</td></tr>
<tr><td>6</td><td>Decelerating to stop</td></tr>
<tr><td>7</td><td>Soft stop</td></tr>
<tr><td>8</td><td>Registration move (armed, not moving)</td></tr>
<tr><td>9</td><td>Home</td></tr>
<tr><td>10</td><td>Following (not used)</td></tr>
<tr><td>128-255</td><td>Error (not used)</td></tr>
</table></td></tr>
<tr><td>144ax</td><td>Integral Error (count-seconds), R only (not supported)</td></tr>
<tr><td>145ax</td><td>Velocity Feedforward [QuickBuilder reference =QS2_VAR_NEW_VEL_FEEDFORWARD]</td></tr>
<tr><td>146ax</td><td>Deceleration (counts/sec^2) [QuickBuilder reference = QS2_NEW_DECELERATION]</td></tr>
<tr><td>147ax</td><td>Dedicated Inputs, R only:
This is a bit map of the input signals
<table>
<tr><th>Bit Number</th><th>Description</th><th>Bit Number</th><th>Description</th></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>0 (lsb)</td><td>Reg. (not supported)</td><td>4</td><td>Rev EOT</td></tr>
<tr><td>1</td><td>Home</td><td>5</td><td>Fwd EOT</td></tr>
<tr><td>2</td><td>Start</td><td>6</td><td>Z/Index (not supported)</td></tr>
<tr><td>3</td><td>Kill</td><td>7</td><td>Not Used</td></tr>
</table>
</td></tr>
<tr><td>148ax</td><td>Acceleration Feedforward [QuickBuilder reference = QS2_VAR_NEW_ACC_FEEDFORWARD]</td></tr>
<tr><td>149ax</td><td>Analog Output, R/W - 32,767 = -10.000V; 32,767 = 10.000V, [QuickBuilder reference = rint(dac_mv * 3.2767)]</td></tr>
</table>

<table>
<tr><td colspan="2">Motion Registers Grouped by axis then function</td></tr>
<tr><td></td><td>For the 15xxx, 16xxx, 17xxx register values below substitute the axis number for 'bx' to get the correct register. Axis #1 = 0; For example the position of axis #1 is stored in register 15000.</td></tr>
<tr><td>15bx0</td><td>Position (counts), R only [QuickBuilder reference = fposc]</td></tr>
<tr><td>15bx1</td><td>Error (counts), R only [QuickBuilder reference = perr * ppr * (uun/uud)]</td></tr>
<tr><td>15bx2</td><td>Velocity (counts / sec), R only [QuickBuilder reference = vel * ppr * (uun/uud)]</td></tr>
<tr><td>15bx3</td><td>Status, R only:
<table>
<tr><th>Value</th><th>Description</th><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Axis not initialized</td><td>6</td><td>Decelerating to stop</td></tr>
<tr><td>1</td><td>Stopped and ready</td><td>7</td><td>Soft stop (not used)</td></tr>
<tr><td>2</td><td>Motion imminent: waiting for start</td><td>8</td><td>Registration move (armed, not moving)</td></tr>
<tr><td>3</td><td>Accelerating</td><td>9</td><td>Home</td></tr>
<tr><td>4</td><td>At MAX speed</td><td>10</td><td>Following (not used)</td></tr>
<tr><td>5</td><td>Decelerating to new MAX speed</td><td>128-255</td><td>Error (not used)</td></tr>
</table>
</td></tr>
<tr><td>15bx4</td><td>Integral Error (count-seconds), R only (not supported)</td></tr>
</table>

<table>
<tr><td>15bx5</td><td>Velocity Feedforward, <s>also used to specify the Output in Direct mode</s> [QuickBuilder reference = QS2_VAR_NEW_VEL_FEEDFORWARD], output in direct mode not supported, use Analog Output instead (15bx9).</td></tr>
<tr><td>15bx6</td><td>Deceleration (counts/sec^2), R only [QuickBuilder reference = QS2_NEW_DECELERATION]</td></tr>
<tr><td>15bx7</td><td>Dedicated Inputs, R only:
This is a bit map of the input signals
<table>
<tr><th>Bit Number</th><th>Description</th><th>Bit Number</th><th>Description</th></tr>
<tr><td>0 (lsb)</td><td>Reg. (not supported)</td><td>4</td><td>Rev EOT</td></tr>
<tr><td>1</td><td>Home</td><td>5</td><td>Fwd EOT</td></tr>
<tr><td>2</td><td>Start</td><td>6</td><td>Z/Index (not supported)</td></tr>
<tr><td>3</td><td>Kill</td><td>7</td><td>Not Used</td></tr>
</table></td></tr>
<tr><td>15bx8</td><td>Acceleration Feedforward [QuickBuilder reference = QS2_VAR_NEW_ACC_FEEDFORWARD]</td></tr>
<tr><td>15bx9</td><td>Analog Output, R/W -32,767 = -10.000V; 32,767 = 10.000V [QuickBuilder reference = rint(dac_mv * 3.2767)]. On 2219 this is read only and Velocity Feedforward is written to for Analog Output.</td></tr>
<tr><td>16bx0</td><td>Reg. Start, R/W – Position at which the registration will be enabled [QuickBuilder reference = QS2_CAP_WINSTART]</td></tr>
<tr><td>16bx1</td><td>Reg. Window, R/W – The range that the registration will be enabled [QuickBuilder reference = QS2_CAP_WINEND_REL]</td></tr>
<tr><td>16bx2</td><td>Reg. Position, R only – The position at which the registration was detected, when Reg status is 1 [QuickBuilder reference = capposc]</td></tr>
<tr><td>16bx3</td><td>Reg. Offset, R/W – The distance to be moved after the registration input [QuickBuilder reference = QS2_CAP_WINOFFSET]</td></tr>
<tr><td>16bx4</td><td>Reg. Status, R/W – 0 = Armed (write 0 to arm), 1 = Detected, can only set to 0 [QuickBuilder reference = QS2_REG_STATUS]</td></tr>
<tr><td>16bx5</td><td>Numerator, R/W – For following the master axis [QuickBuilder reference = QS2_VAR_MTN]</td></tr>
<tr><td>16bx6</td><td>Denominator, R/W – For following the master axis [QuickBuilder reference = QS2_VAR_MTD]</td></tr>
<tr><td>16bx7</td><td>Leader Position, R only – Only valid when following a master axis (not supported)</td></tr>
<tr><td>16bx8</td><td>Leader Velocity, R only – Only valid when following a master axis (not supported)</td></tr>
<tr><td>16bx9</td><td>Reserved</td></tr>
<tr><td>17bx0</td><td>Firmware Revision, R only</td></tr>
<tr><td>17bx1</td><td>Filter & Mode, R/W:</td></tr>
</table>

~~In Direct mode the Feedforward Velocity gain specifies the output value (0 to 32767) with a value of 32767 = 10V (sign depends on the Filter type).~~
[QuickBuilder reference = QS2_FILTER_MODE]

Description	*Value*
Lower 3 bits (0x07) Filter type	0 or 3 = PID 1 = + Direct (CW) 2 = - Direct (CCW) 4 = PAVff 5 = PAV 6 = Stepper 7 = Initialize Encoder Resolution
Bits 4 & 5 Accel/Decel Type	0 = Linear 1 = S Curve ~~2 = Parabolic 3 = Inverse Parabolic~~
Bit 7 (0x80)	0=Trajectory Following 1 (value 128) = Encoder Following

Note: Initialize Encoder Resolution, filter type 7 is only a temporary mode that can be applied anytime there is no motion. It is recommended this be done prior to initial motion. This can be used to override the default ppr, mppr, and sppr. Upon writing this value the following registers will be initialized as follows, thus set accordingly prior to execution:

ppr = QS2_VAR_NEW_VEL_FEEDFORWARD
mppr = QS2_VAR_NEW_ACC_FEEDFORWARD
sppr = QS2_NEW_DECELERATION

After changing the above variables set them back to their previous values and set the Filter Mode to the proper mode for motion desired.

17bx2

Input Polarity, R/W:

This is a bit map that controls the active level of the input signals,. When the bit is 0 then the input is active when it is On; if the bit is 0 then the input is active Off.

Bit Number	*Description*	*Bit Number*	*Description*
0 (lsb)	Reg. (not supported)	**4**	Rev EOT
1	Home	**5**	Fwd EOT
2	Start	**6**	Z/Index (not supported)
3	Kill	**7**	Not Used

17bx3

Home Direction, R/W [QuickBuilder reference = QS2_HOME]

Direction	*Description*
CCW	0 or –1 = Home & Index -2 = Home Only -3 = Index Only (not supported)

	CW | 1 = Home & Index 2 = Home Only 3 = Index Only (not supported)
17bx4	**Options**, R only (not supported)
17bx5	**Reserved**
17bx6	**Maximum Following Error, R/W** default = 30000 [QuickBuilder reference = perrlimit * ppr]]
17bx7	**Speed Limit, R/W** – overrides maximum velocity, default = 4194303 steps/sec (not supported)
17bx8	**Maximum Position, R/W** – Used as a Software EOT when it is larger than the Minimum Position (not supported)
17bx9	**Minimum Position, R/W** – Used as a Software EOT when it is smaller than the Maximum Position (not supported)

7.2 Quickstep Variables

Quickstep Variables	Description	Type
QS2_Status	Axis status as defined by register 143xx.	read-write
QS2_Cmd	Command to be processed by the MSB, emulates 2700 2219 module.	read-write
QS2_Overrides	Override commands that can be processed by the MSB during motion without a fault.	read-write
QS2_Holding	Holding command to be processed by the MSB.	read-write
QS2_Params	Parameter command to be processed by the MSB. Written once any 'VAR_NEW' variables are updated.	read-write
QS2_VAR_NEW_ACCELERATION	X value for selected debugTableRow.	read-write
QS2_VAR_NEW_MAX_SPEED	Y value for selected debugTableRow.	read-write
QS2_VAR_NEW_PROPORTIONAL	Requested new kd, processed by MSB as required. This is also written to by the QS2 profile statement.	read-write
QS2_VAR_NEW_INTEGRAL	Requested new ki, processed by MSB as required. This is also written to by the QS2 profile statement.	read-write
QS2_VAR_NEW_DIFFERENTIAL	Requested new kd, processed by MSB as required. This is also written to by the QS2 profile statement.	read-write
QS2_VAR_NEW_VEL_FEEDFORWARD	Processed by MSB application.	read-write
QS2_VAR_NEW_HOLDING_MODE	Processed by MSB application.	read-write
QS2_VAR_NEW_DECELERATION	Processed by MSB application.	read-write
QS2_VAR_NEW_FORCE_POSITION	Processed by MSB application.	read-write
QS2_VAR_NEW_FORCE_CUMULATIVE	Processed by MSB application. Not currently used.	read-write
QS2_CAP_WINSTART	Processed by MSB application.	read-write
QS2_CAP_WINEND_REL	Processed by MSB application.	read-write

Quickstep Variables	Description	Type
QS2_CAP_WINOFFSET	Processed by MSB application.	read-write
QS2_VAR_NEW_ACC_FEEDFORWARD	Processed by MSB application.	read-write
QS2_HOME	Processed by MSB application.	read-write
QS2_VAR_MTN	Processed by MSB application.	read-write
QS2_VAR_MTD	Processed by MSB application.	read-write
QS2_LAST_CMD	Last QS2_Cmd processed.	read-write
QS2_CMD_CNT	Number of QS2_Cmd's processed.	read-write
QS2_OVERRIDE_CNT	Number of override commands processed.	read-write
QS2_HOLDING_CNT	Number of holding commands processed.	read-write
QS2_PARAM_CNT	Number of parameter commands processed.	read-write
QS2_MSB_STATE	General scratch storage used by the MSB to write program execution state information.	read-write
QS2_FILTER_MODE	Reference register 17bx1 for mode settings.	read-write
QS2_TMP1	General scratch storage used by the MSB as needed (integer).	read-write
QS2_TMP2	General scratch storage used by the MSB as needed (integer).	read-write
QS2_TMP3	General scratch storage used by the MSB as needed (integer).	read-write
QS2_TMP4	General scratch storage used by the MSB as needed (integer)	read-write
QS2_REG_STATUS	Registration status, write a 0 to arm, else read a 1 if detected.	read-write

7.3 Input Mapping (M3-40 Only)

M3-40A inputs are monitored by the QS2MSB MSB program and when executing the DIN inputs are monitored similar to a 2219. The inputs are mapped as:

DIN1 - START
DIN2 - REGISTRATION INPUT
DIN3 - FWD LIMIT
DIN4 - REV LIMIT
DIN5 - HOME

8 Chapter 8: Fault Codes & MSB Debugging

Should an error occur several registers contain information which can be helpful in detecting what caused the problem. These registers exist for each axis:

Fault Variables	Description	Type
fault1 fault2 (not used) fault3 (not used) fault4 (not used)	Fault status words, reference Chapter 8.	read-only
faulted	0 = no fault, 1 = faulted.	read-only
faultFunction	Code 0 to N which represents internal function where fault occurred. Thus far defines as: FGTick - 1 runMSB - 2 processMSB - 3 process_motion_command - 4 DP_MGRTask_Axis1 - 5 DP_MGRTask_Axis2 - 6 processVFC - 7	read-only
faultMSB	The MSB number from 0 to 31 which has faulted.	read-only
faultMSBLine	The line number as referenced to source code MSB where the fault occurred. Note that the source must be in sync with what is executing for this to be correct.	read-only
faultMSBOffset	Absolute byte offset into MSB binary opcode where was executing when fault occurred. Internal use.	read-only
faultOpcode	MSB opcode that was being executed when fault occurred. Internal use.	read-only

8.1 Fault Codes

⚠ Note that firmware prior to V1.40 (M3-40A) use outdated fault codes. Changes were made to enhance diagnostic abilities.

Fault Codes	Description	Code Value
MF_NO_ERROR	No error.	0
MF_GENERICFAULT	Generic Motion Fault.	1
MF_INVALIDTIME	Negative or Zero 'time' specified in MOVE.	2
MF_INVALIDVEL	Negative or Zero 'velocity' specified in MOVE.	3
MF_INVALIDACC	Negative or Zero 'acc' specified in MOVE.	4
MF_INVALIDDEC	Negative or Zero 'dec' specified in MOVE.	5
MF_INVALIDRATE	Negative or Zero 'rate' specified in MOVE.	6
MF_ONLYINBG	QM command only allowed in BG MSB.	7
MF_MOTIONACTIVE	MOVE attempted while MOVE in progress.	8
MF_UNIMPLEMENTED	MOVE attempted while MOVE in progress.	9
MF_WRONGMODE	In wrong mode (positioning/tracking/slewing.	10
MF_FGMSBLIMIT	FG MSB limit reached.	11
MF_NOTINSLEW	Not in SLEW mode.	12
MF_FOLLOWERR	Following error limit reached.	13
MF_BADINPUTNO	Invalid input number specified.	14
MF_NOTENABLED	Not enabled.	15
MF_BADARGUMENT1	Bad argument1/parameter.	16
MF_INVALID_TBL_OP	Invalid 'table' operation.	17

Fault Codes	Description	Code Value
MF_NOTINTRACK	Not in 'tracking' mode.	18
MF_CANTCONSUME	Illegal state for 'consume'.	19
MF_SESGMOVE_ERROR	'Segmented Move' error.	20
MF_SESGMOVE_SIZE	'Segment size' error, too many.	21
MF_NOCAMFILE	Requested CAM file not found.	22
MF_REMOTE_READ	Read of controller register failed.	23
MF_REMOTE_WRITE	Write of controller register failed.	24
MF_NOMSBFILE	MSB file does not exist on flash disk.	25
MF_BADARGUMENT2	Bad argument2/parameter.	26
MF_BADARGUMENT3	Bad argument3/parameter.	27
MF_BADARGUMENT4	Bad argument4/parameter.	28

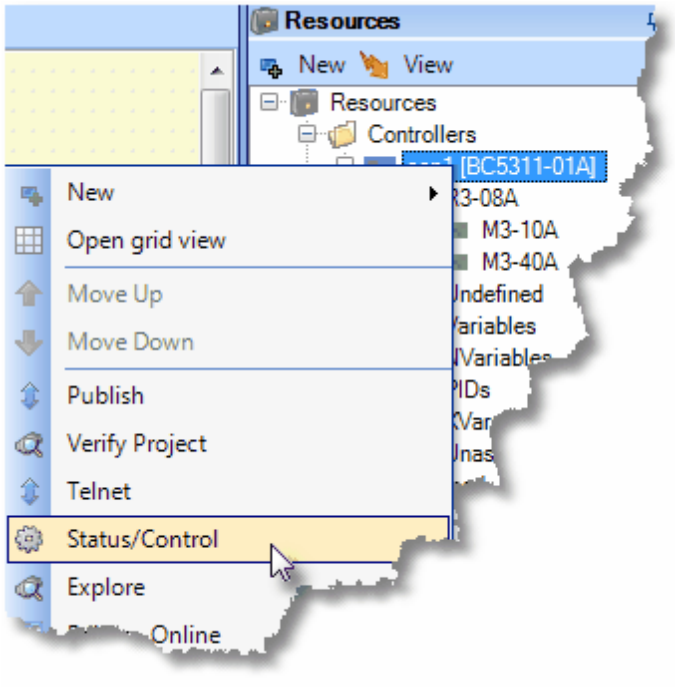
Additional EtherCAT Only Fault Codes	Description	Code Value
MF_SDO_READ	EtherCAT SDO read failed. Either network error or object not supported by device.	29
MF_SDO_WRITE	EtherCAT SDO write Failed. Either network error or object not supported by device.	30
MF_DRIVE_ERR_MSG	Generic drive error message.	31
MF_ECAT_OFFLINE	EtherCAT is offline	32
MF_DCSYNC	Execution of DCSync instruction failed. Drive may be offline or not support objects.	33
MF_INVALID_ECAT_MODE	Attempted to set a mode (cmode) the drive does not support. Typically CSP mode is used generically.	34

Additional EtherCAT Only Fault Codes	Description	Code Value
MF_STACK_ERROR	MSB instruction execution error, attempted to pop a variable off the stack and it was not there?	35
MF_READONLY	Trying to write to a read only variable.	36
MF_INVALID_TBL_OP_Y	Attempted to address more tables than available on Y axis.	37
MF_INVALID_TBL_OP_Z	Attempted to address more tables than available on Z axis.	38
MF_INVALID_TRACKING	Attempted an invalid instruction while in tracking mode. Typically beginning slew.	39
MF_TIMEOUT	IncentiveECAT API: Error occurring while waiting for in-position and have a timeout occur.	40
MF_ERROR_NO_SERVO	IncentiveECAT API: Error occurring while attempting to read or write to an axis that does not exist.	41
MF_ABORT_REQUESTED	IncentiveECAT API: Error occurring while executing an API command and the user application requested an ABORT occur, restarting the network.	42
MF_ERROR_OO_RANGE	Incentive ECAT API: Error occurring while attempting to online/offline an invalid axis #.	43
MF_ERROR_INIT_CONFIG	Incentive ECAT API: Attempting to online an offline axis and it failed while initializing the configuration.	44
MF_ERROR_NO_INIT	Incentive ECAT API: Attempting to online an offline axis and it failed while trying to place it in the EtherCAT INIT state.	45
MF_ERROR_NO_PREOP	Incentive ECAT API: Attempting to online an offline axis and it failed while trying to place it in the EtherCAT PRE-OP state.	46
MF_ERROR_NO_SAFEOP	Incentive ECAT API:	47

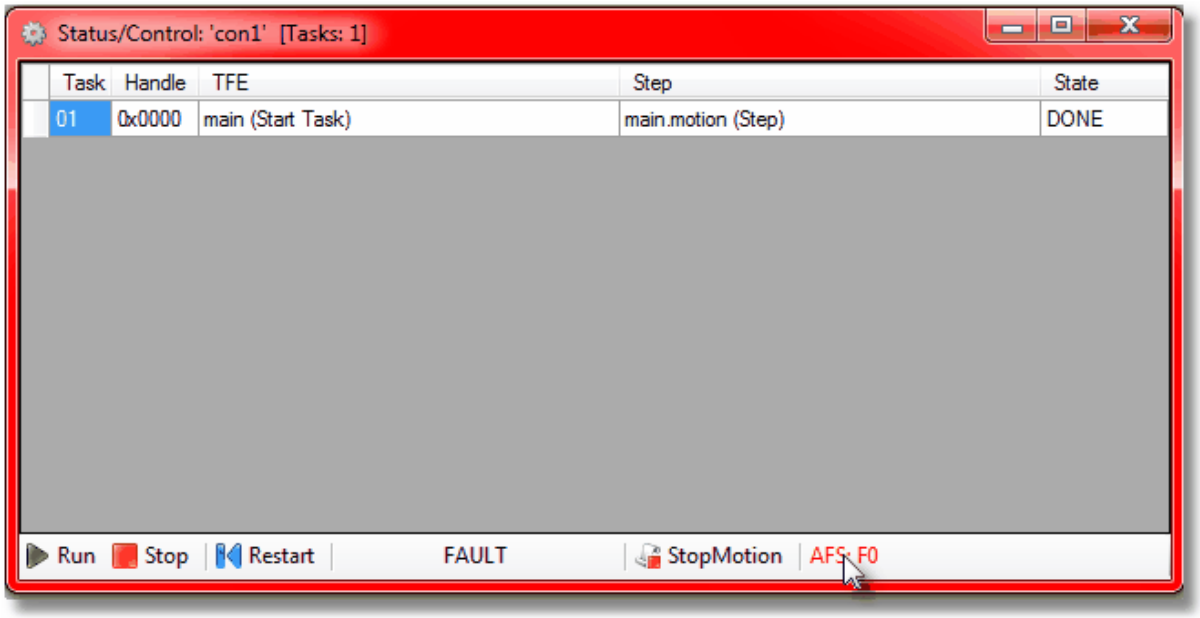
Additional EtherCAT Only Fault Codes	Description	Code Value
	Attempting to online an offline axis and it failed while trying to place it in the EtherCAT SAFEOP state.	
MF_ERROR_NO_OPERATIONAL	Incentive ECAT API: Attempting to online an offline axis and it failed while trying to place it in the EtherCAT OPERATIONAL state.	48
MF_ERROR_INIT_PDO	Incentive ECAT API: Attempting to online an offline axis and it failed while trying to initialize the PDO mapping.	49
MF_ERROR_REMAP_CONFIG	Incentive ECAT API: Attempting to online an offline axis and it failed while trying to remap the configuration.	50
MF_ERROR_DC_RESTORE	Incentive ECAT API: Attempting to online an offline axis and it failed while trying to restore propagation delay.	51
MF_ERROR_DC_SYNC01	Incentive ECAT API: Attempting to online an offline axis and it failed while trying to setup DC Sync (some drives require DC Sync operational before the drive is fully in the OPERATIONAL state.	52
MF_ERROR_ILLEGAL_VAL	Incentive ECAT API: Attempting to online an offline axis and it is no longer available in the drive table.	53
MF_ERROR_NOT_ONLINE	Incentive ECAT API: Attempting to online an offline axis and it is no longer available. Probably still offline.	54

8.2 MSB Status/Control Monitor Fault Processing

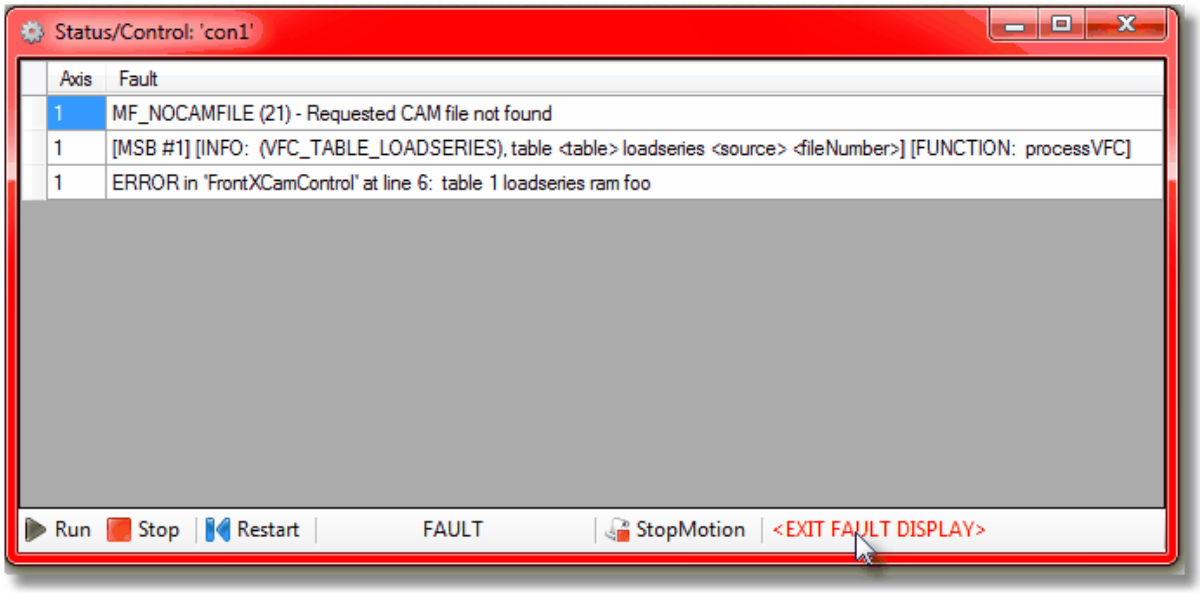
There are a number of features within QuickBuilder to enable the debugging of QuickMotion MSB's. This can be either during normal operation or should a fault occur. A fault is indicate by a flashing FLT LED on the controller CPU. To observe a QuickMotion fault the Status/Control monitor can be viewed:



Once the Status/Control window appears observe and click the AFS text. Note that each character represents an axis, with the first on the far left. In the example below a 0 means the axis is OK, F that there is a fault. Below shows a fault on axis 1 since it is 'F'.



Once clicked detailed information about the fault will be shown, if available:

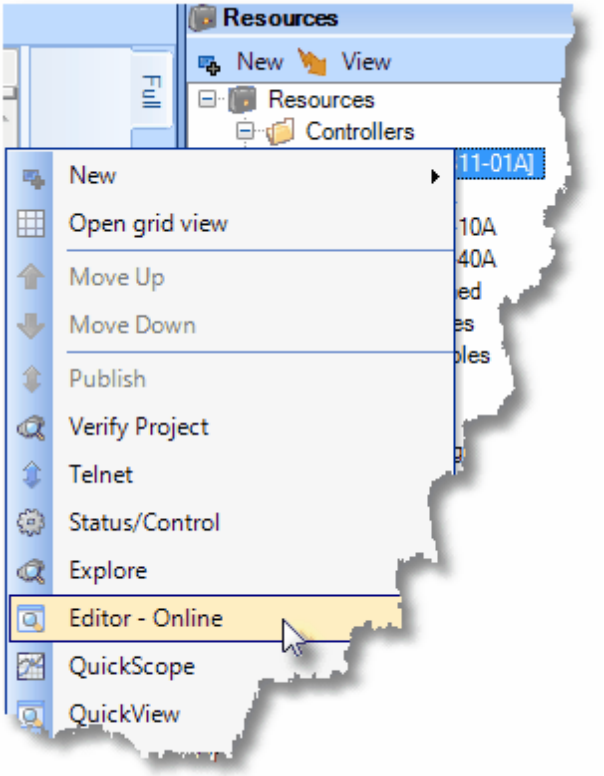


Note that the error occurred at line 6 of the source code of the FrontXCamControl MSB. In referencing that MSB we can see the line listed, 'table 1 loadseries ram foo' as being the problem. In this case there was no camtable209 file present within the controller flash disk.:



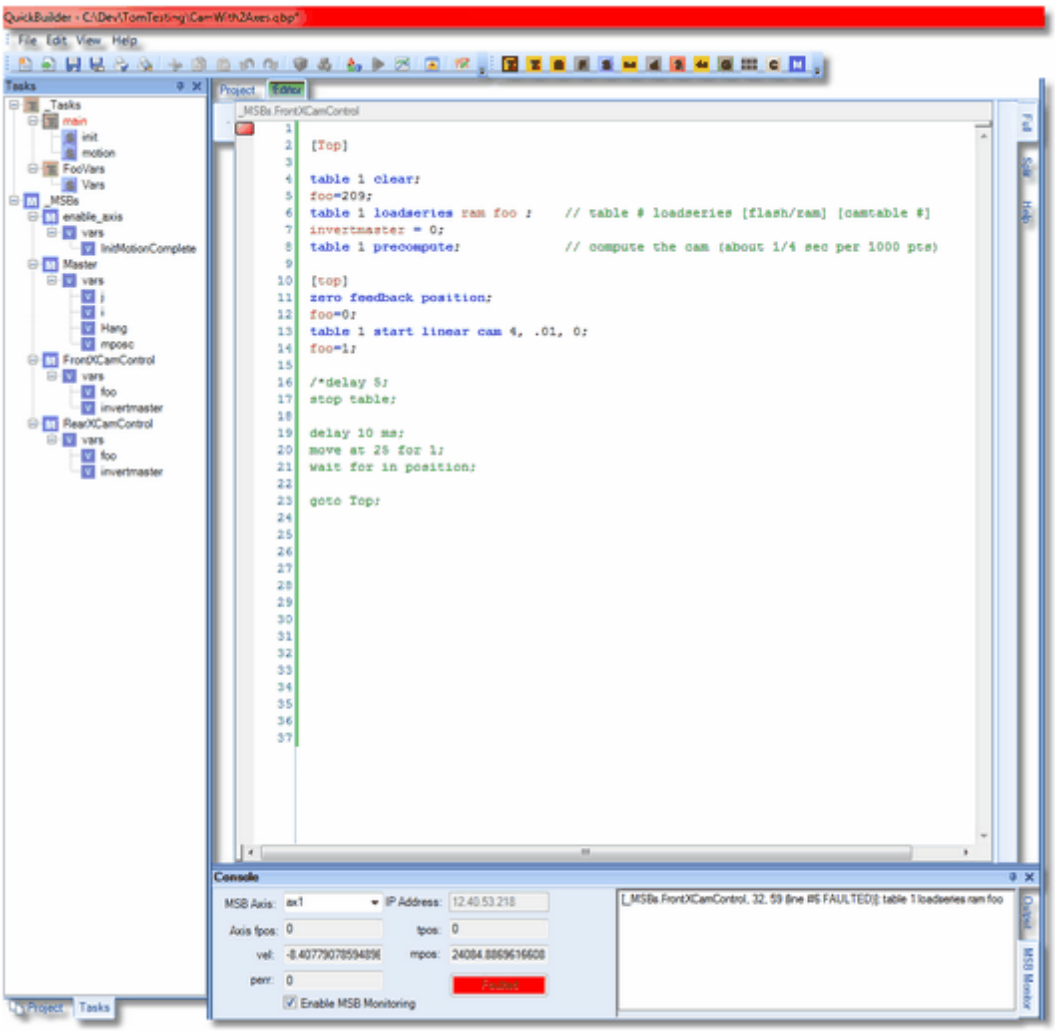
8.3 MSB Monitor

QuickBuilder offers a MSB Monitor when online in the Editor mode.



This monitor periodically (about every second) refreshes axis information for display. Current fpos, mpos, vel, tpos and perr are available as well as the instruction and state of MSB's that are executing. A pull down combo box lists all available axis, that selected is what will be automatically refreshed.

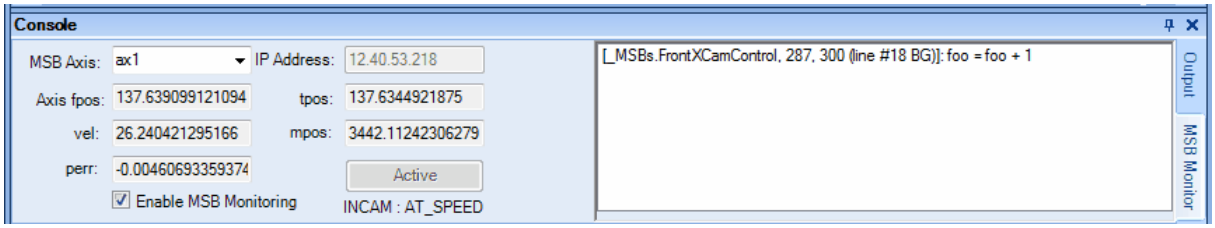
If the axis is faulted, using the example from the 'MSB Status/Control Monitor Fault Processing' section, the following will appear:



⚠ Note that the 'Enable MSB Monitoring' check box must be checked for monitoring to be active. Also the Editor tab should be green to indicate online debug mode.

⚠ Double clicking on the MSB line appearing in the list box will automatically make that code and line current in the Editor.

In situations where a fault had not occurred multiple MSB's would appear executing, as well as their line number and axis motion status:



9 Appendix: Sample Code

⚠**WARNING:** The following examples are offered for training purposes only and are not intended to perform any actual real-world application or function.

```
// ---------------  Pause Motion MSB  --------------------------

/* This MSB will pause motion by moving the timebase to 0
and then back to 1 based on switch 3 position.
Note: that changes to the timebase variable only impact
the actual motion commands other MSB commands such as
delay are not altered. */

[top]
      wait for rise of 3;           // wait for rising edge of input3
      timebase=0;                   // set the motion timebase to zero
      wait for fall of 3;           // wait for falling edge of input3
      timebase=1;                   // put timebase back to 100%
      goto top;                     // repeat

end;
// ---------------  Jog MSB  --------------------------

/* This MSB performs a simple jogging routine
The variable JogSpeed is passed to this MSB to set
the jog velocity.  If switch 1 is on a positive Jog
is activated, if switch 5 is on a negative Jog is activated;
if neither 1 or 5 are on zero speed is commanded, and the
motor stops. */

JogSpeed=1;                         // set a default jog speed

slew begin;                         // witch to slewing mode

[loop]
      // check the switches
      if !din1 && !din5 then speed=0;
      if din1 then speed = JogSpeed;
      if din5 then speed = -JogSpeed;

      slew at speed in 0.5;         // slew to speed in .5 sec
      delay 510;                    // wait 510 ms until at speed

      if !din2 goto loop;           // as long as input 2 is off loop

      slew end;                     // return to position mode

end;
```

```
// ---------------  Home via Z MSB  --------------------------


// add a move to switch code here if needed

foundz = 0;
// set zdir to 1 to search in the positive dir
// set zdir to -1 to search in the negative dir
zdir = -1;

// check if we know where the Z-pulse is
if zpulse goto knownz;

// dont know where z is, so...
// move positive for +/- 2 revs looking for it
zero feedback position;
move in 0.25 for zdir*2;

[searchloop]
// a z while moving?
// check the zpulse variable (1 if a z pulse has been seen)
if zpulse goto foundmid;
// done?
if !inpos goto searchloop;
// no z, stop and quit
stop;
end;

// found a z mid move, so stop
[foundmid]
new endposition relative 0 using 10000;
wait for in position;

// move to z
[knownz]

// find the Z that is closest
if zdir > 0 goto posz;

[negz]
move in 0.125 to ZPULSE_NEG;
goto exit;

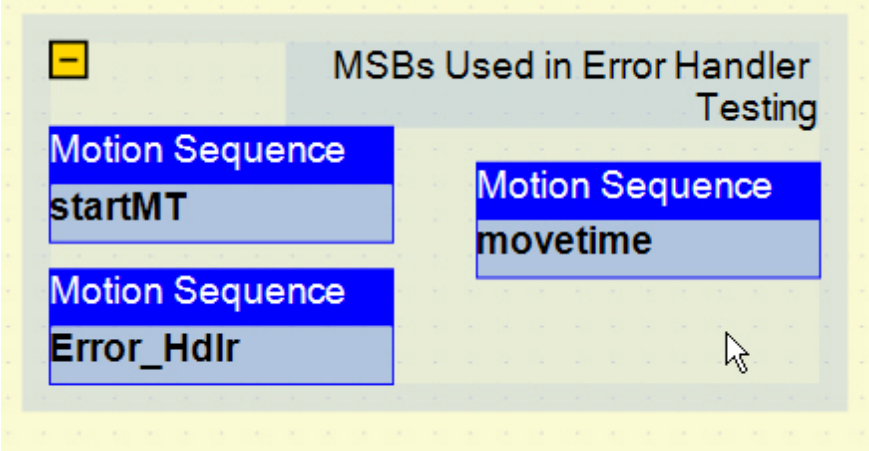
[posz]
move in 0.125 to ZPULSE_POS;
goto exit;

[exit]
wait for in position;
zero feedback position;
```

```
foundz = 1;
end;

// ---------------  Error Handler Example  -------------------------
```

The following MSBs illustrate how an error handler can be used on a motion axis. The code for each is given below with a brief explanation.



startMT MSB: **This MSB starts the asynchronous event handler. In this case Error_Hdlr will automatically be called whenever there is a hardfault.**

```
on hardfault start Error_Hdlr;      //start an asynchronous event to
                                    monitor for a hardfault error
start movetime FG;                        //now go do some motion
```

Error_Hdlr: **This MSB contains the error handler code. The comment at the end gives a listing of valid error codes**

```
// Error handler example MSB

// Check for a fault
if fault1 == 0 goto genfault; //Generic Motion Fault
if fault1 == 1 goto invtime;  //Negative or Zero 'time' specified in MOVE
if fault1 == 2 goto invvel;   //Negative or Zero 'velocity' specified in
MOVE
                                                  //(etc)continue on to trap
all errors if you want

goto unknown;                                     // If you get here there's no
known fault code

[genfault]                                        //routine for general fault goes
here
//(put code here)
```

```
[invtime]
// Invalid Time Fault:  Trigger this error by setting move time to 0 in the
movetime MSB
      setout 5;                    //turn on output to signal error
      delay 5000;                  // wait 5 sec
      clrout 5;                    // turn off output 5
      delay 1000;                  // wait 1 sec
      reset;                             // reset all faults

      delay 1000;                  // wait 1 sec
      start movetime FG;      //re-start the MSB.  Hint if you don't change
                                 the movetime
      //you'll end up right back here in six seconds.

[invvel]

      //(code)

[unknown]

      //(code)

end;
```

movetime MSB: **This MSB contains a simple motion routine used to trigger a hardfault**

```
zero feedback position;

xm=1;             //default mode setting

// Do a repeating forward and back move
[top]

      time=0;                                          //reset timer
      move in time2 for dist mode xm;     //move forward
      wait for in position;

      move in time2 for -dist mode xm; //move back
      wait for in position;
      movetime=time;                     //update movetime

goto top;
```

10 Appendix: Command Hyperlinks

Statements:

Utility
Set
Program Flow
Common bits and variables
I/O
Simple Motion
Gearing
Position and Capture
Loading Tables
Spline/CAM
Virtual Master
Segmented Moves
Host Register

Utility Statements:

stop { slewed using *rate* **}**
drive enable
drive disable
delay *time* **ms**
variable = expression
zero feedback position
zero target position
zero following error
reset
if *condition* **then** *variable* **=** *expression*
wait until *condition*

Set Statements:

set common bit *number state*
set common var *number value*
set loopperiod *value*
set mode positioning
set mode tracking
set timeout *ticks*
set target position *value*
set feedback position *value*
set target position counts *vcounts*
set feedback position counts *vcounts*
set simulated feedback on/off
offset position *value*
offset position counts *vcounts*
set master *mode* **{ using** *global* **}**

Program Flow Statements:

[*label***]**
start *MSB mode*
end { and start *MSB mode* **}**
abort *MSB*
goto *label*
if *condition* **goto** *label*
on *asynchevent asynchhandler*

Common bits and variables Statements:

set common bit *number state*
wait common bit *number state*
set common var *number value*
wait common var *number range*

I/O Statements:

setout *outputlist*
clrout *outputlist*
pulse *output* **for** *n*
pls *output* **using** *reference definitions*
pls *output state*
wait for *transition* **of** *input* { **or** *condition* }
generate output *output* **rate** *freq*
generate *n* **steps on** *pair*
variable = **ctr[n]**
ctr[n] = *expression*
ctr[n] = *offset*
generate alternate *mode*

Simple Motion Statements:

move to *position* **{ using** *acc, dec* **}**
move at *maxvelocity* **to** *position* **{ using** *acc, dec* **}**
move trap to *position* **using** *rate*
move in *time* **to** *position* {**mode** *n* }
move for *displacement* **{ using** *acc, dec* **}**
move at *maxvelocity* **for** *displacement* **{ using** *acc, dec* **}**
move trap for *displacement* **using** *rate*
move in *time* **for** *displacement* {**mode** *n* }
wait for in position
new endposition *position* **using** *rate*
new endposition relative *displacement* **using** *rate*
slew begin
slew at *velocity* **in** *time*

slew for *displacement*
slew end

Gearing Statements:

gear at *numerator : denominator*
gear at *numerator : denominator* **in** *counts*
gear at *numerator : denominator* **in** *counts* **after** *acounts*
gear for *slavecounts* **in** *mastercounts*
gear for *slavecounts* **in** *mastercounts* **after** *acounts*
offset slave by *slavecounts* **in** *time*
wait master *counts*
wait slave *counts*
wait *source* **within** *start , end*
wait *source* **outside** *start , end*
zero *masslv* **counters**

Position and Capture Statements:

set capture *transition* **of input** *input* { **gate input** *gateinput gatestate* }
set capwin range *start, end* **using** *reference { arm }*
wait capture { **if limit of** *limit* **goto** *limitlabel* }

Loading Tables Statements:

table *n* **clear**
table *n* **addpair** *xexpression , yexpression*
table *n* **addseries** *pairs*
table *n* **copy from** *rowOffset1* **to table** *m rowOffset2 numRows*
table *n* **loadoffset** *rowOffsetFile, numPairs,rowOffsetTable*
table *n* **loadseries** *source fileNumber*

Spline/CAM Statements:

table *n* **continue**
table *n* **precompute**
table *n* **start** *imethod tscale , rpscale , repeatcount*
table *n* **start** *imethod* **cam** *mpscale , spscale , repeatcount*
stop table

Virtual Master Statements:

move master at *rate* **for** *limit* { **using** *ramp* }

Segmented Move Statements:

segmove table clear

segmove table accdec to vel using rate
segmove table accdec to vel for displacement
segmove table slew until position
segmove table stop at position using rate
segmove table start relative

Host Register Statements:

host read *variable, register {, row, column}*
host write *variable, register {, row, column}*

QuickBuilder PID Reference Guide

Copyright © 2004 - 2007 Control Technology Corp. All Rights Reserved.

Control Technology Corp.
25 South Street
Hopkinton, MA 01748
Phone: 508.435.9595 • Fax 508.435.2373

Document No. 951-530031-006

> ⚠ **WARNING:** Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.

The information in this document is subject to change without notice. The software described in this document is provided under license agreement and may be used and copied only in accordance with the terms of the license agreement. The information, drawings, and illustrations contained herein are the property of Control Technology Corporation. No part of this manual may be reproduced or distributed by any means, electronic or mechanical, for any purpose other than the purchaser's personal use, without the express written consent of Control Technology Corporation.

The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions.

See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

Model Number	QuickBuilder Revision	Firmware Revision
5300	>=1.2.2596	>= 5.00.90

1 Chapter 1: Overview

This document details the operation of QuickBuilder's PID object. The main purpose of this guide is to document the PID loop object used by QuickBuilder on Blue Fusion 5300 series controllers so that experienced users can best apply it in their applications. PID objects are set up using the QuickBuilder Automation Suite and then downloaded to a Blue Fusion Model 5300 controller. The PID object allows the Model 5300 automation controller to precisely control temperature, pressure, flow or even simple motion applications. (Note: for most motion control applications, CTC recommends using a dedicated motion module such as the M3-40 series).

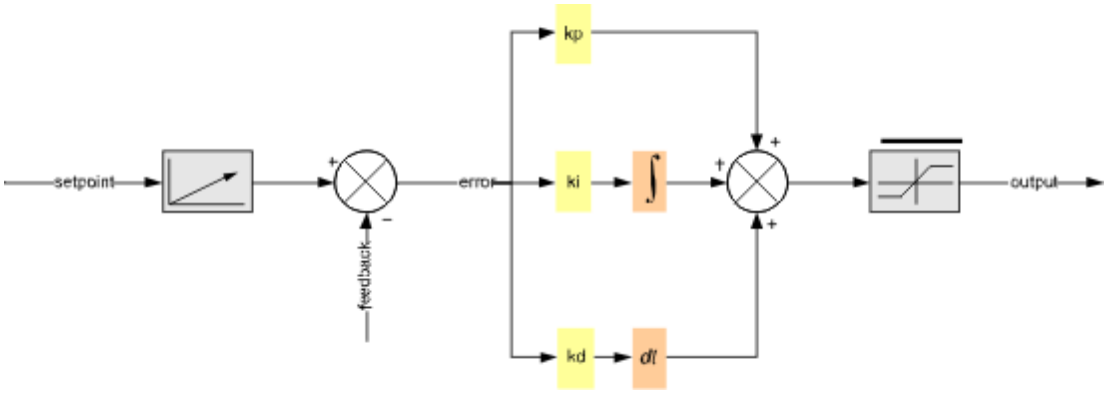
What is a PID loop?

The term PID stands for **P**roportional plus **I**ntegral and **D**erivative control. The PID control loop is ideal for applications where a desired setpoint value needs to be accurately maintained by the output (known as a "Process Variable") of a control system even when the control system experiences load disturbances and / or measurement noise. And, most importantly for industrial applications, a PID loop when properly tuned will reduce the error between the setpoint and the process variable in the minimum possible time.

The PID loop does this by measuring the output of the process via some type of feedback sensor and then calculating the difference (error) between the output and the setpoint. If an error exists, the controller tries to minimize this error by adjusting the output to bring the process closer to the desired setpoint.

PID loops are calculated repetitively at precise intervals and are able to use the history of error information measured during previous cycles to determine how best to adjust the output in the current cycle. The way in which the PID loop parameters are set up will determine how the loop responds to a measured error. If the loop parameters are set too aggressively (under damped), it may cause the process to become unstable and oscillate. If the loop parameters are not aggressive enough (over damped), the system may require too much time to return to the setpoint.

The following diagram shows how a basic PID loop is calculated.



As can be seen from this diagram, the aggressiveness of the output response is directly controlled by the P, I, and D factors.

The **Proportional** factor gives an immediate response that is directly proportional to the error. The larger in magnitude *kp* is, the greater the response to an error.

The **Integral** factor is the term that allows the PID loop to eliminate steady-state errors. Increasing the value for *ki* will allow the error to be eliminated more quickly, but may also result in overshoot of the desired setpoint value.

The **Derivative** factor gives the PID loop a "forward looking" input since it is based on the slope of the error. A larger value for *kd* will reduce overshoot and settling time, but will also make the system less responsive to short term disturbances.

Setting and adjusting the PID parameters is called tuning. While systems can be successfully tuned by trial and error, better results are obtained by personnel experienced with both the process to be controlled and PID loop tuning methods. For more information on tuning, see Appendix A.

2 Chapter 2: The QB PID Object

2.1 Features

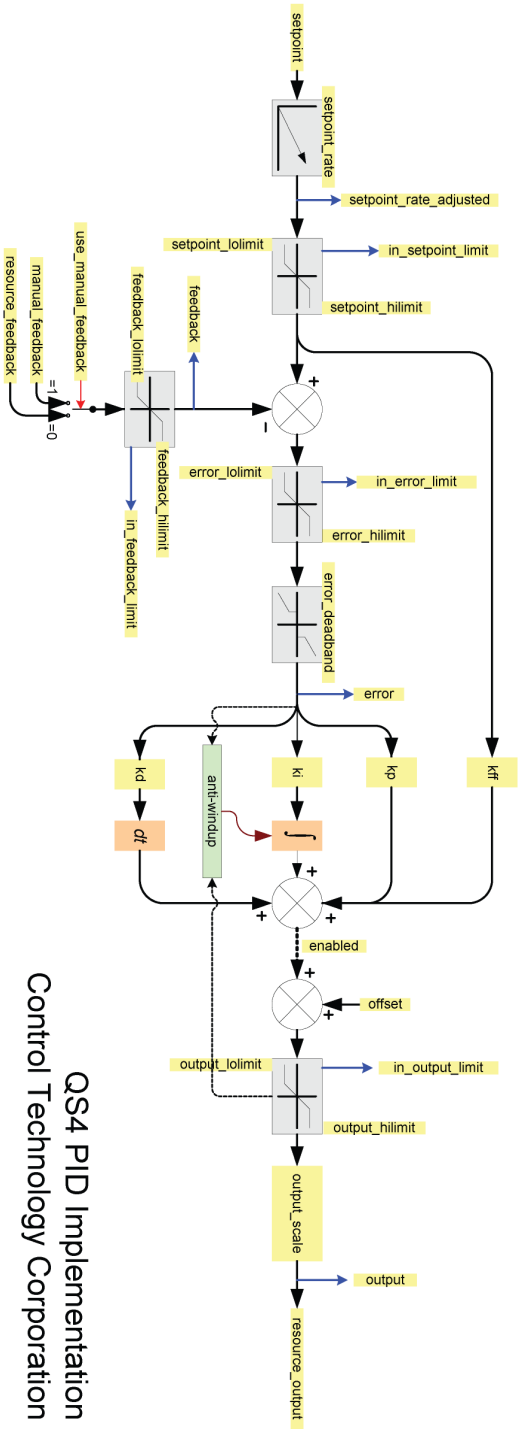
The PID Object in QuickBuilder allows users to set up very sophisticated high performance PID loops on the Model 5300 automation controller. The QB PID Object has many advantages over more simplistic PID loop implementations. Some of the key features are outlined below:

Feature	Benefit
Up to 256 PID loops / CPU	The Model 5300 can tackle even the most demanding applications
Advanced PID loop equation	CTC uses a state-of-the-art loop equation with more than 20 settable parameters and multiple alarms and status outputs. This allows the QB PID Object to solve a wide variety of applications automatically without adding auxiliary control logic to the project.
Floating point calculations	Using 64-bit floating point calculations ensures the most precise results, yielding improved loop response.
Timing accuracy < 200 nanoseconds	Ensures quick response and fast returns to steady state conditions.
Fast loop update	User settable down to 1ms, it allows the QB PID loop to be used for a wider variety of applications.

Feature	Benefit
Individually settable loop update rates	Ability to optimize each loop independently. Also allows more control over CPU utilization.
Loops implemented as QB objects	Loops do not consume user memory, QuickStep steps, or user variables. Loops run automatically as background tasks and do not decrease the user task limit.
Table-driven setup	Simplifies setup process. No need to code PID initialization steps.
Multiple properties	Easily set up and customize PID loops for a wide variety of applications.
Properties changeable on-the-fly	Allows the QuickBuilder program to adjust loop behavior based on external events.
Multiple status and alarm parameters	Eliminates the need to code these items separately, saving time and resources. Also provides faster notification.
Programmable deadband	Eliminates excessive dithering around a setpoint.
Multiple modes	Easily set up manual, automatic, or cascaded control loops.

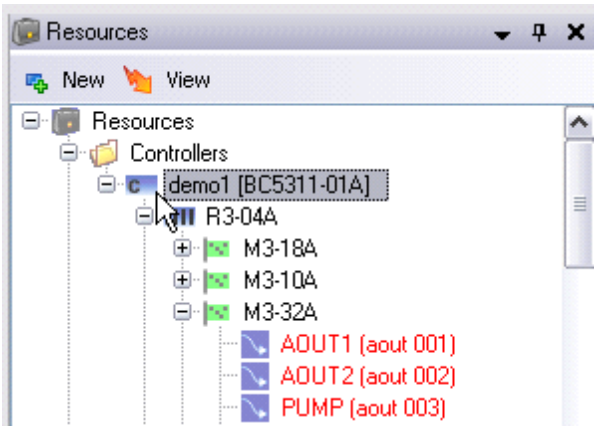
2.2 PID Loop Algorithm

The actual loop algorithm used by the PID object is shown in the diagrams below. In the next section we will explain the process of adding a PID loop to a controller. Following the setup section we will define all of the parameters and variables shown in the PID diagram.

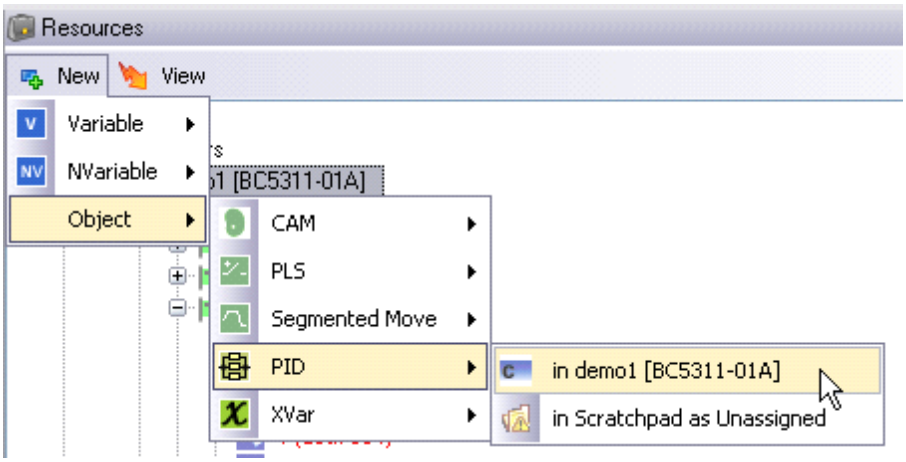


2.3 PID Object Setup

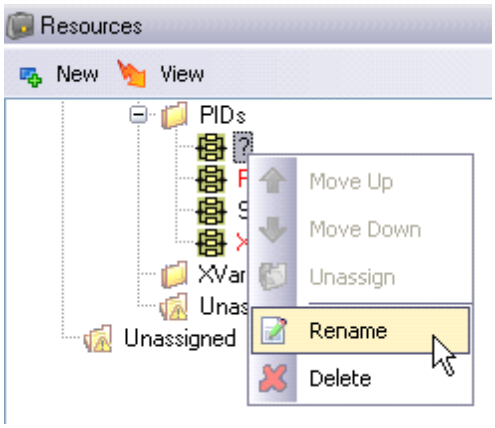
Adding a PID object to your QuickBuilder project is easy. PID objects are associated and linked to Model 5300 CPU processors. To add a PID loop, simply click on the controller icon in the Resources window to select the destination controller for the PID loop.



Then go to the **New** menu item and add a PID Object.

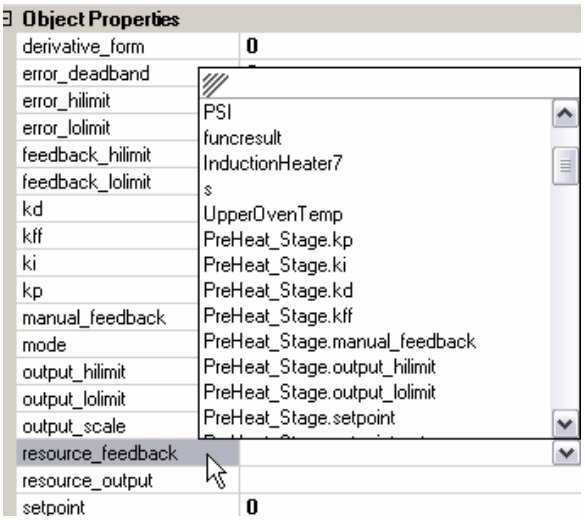


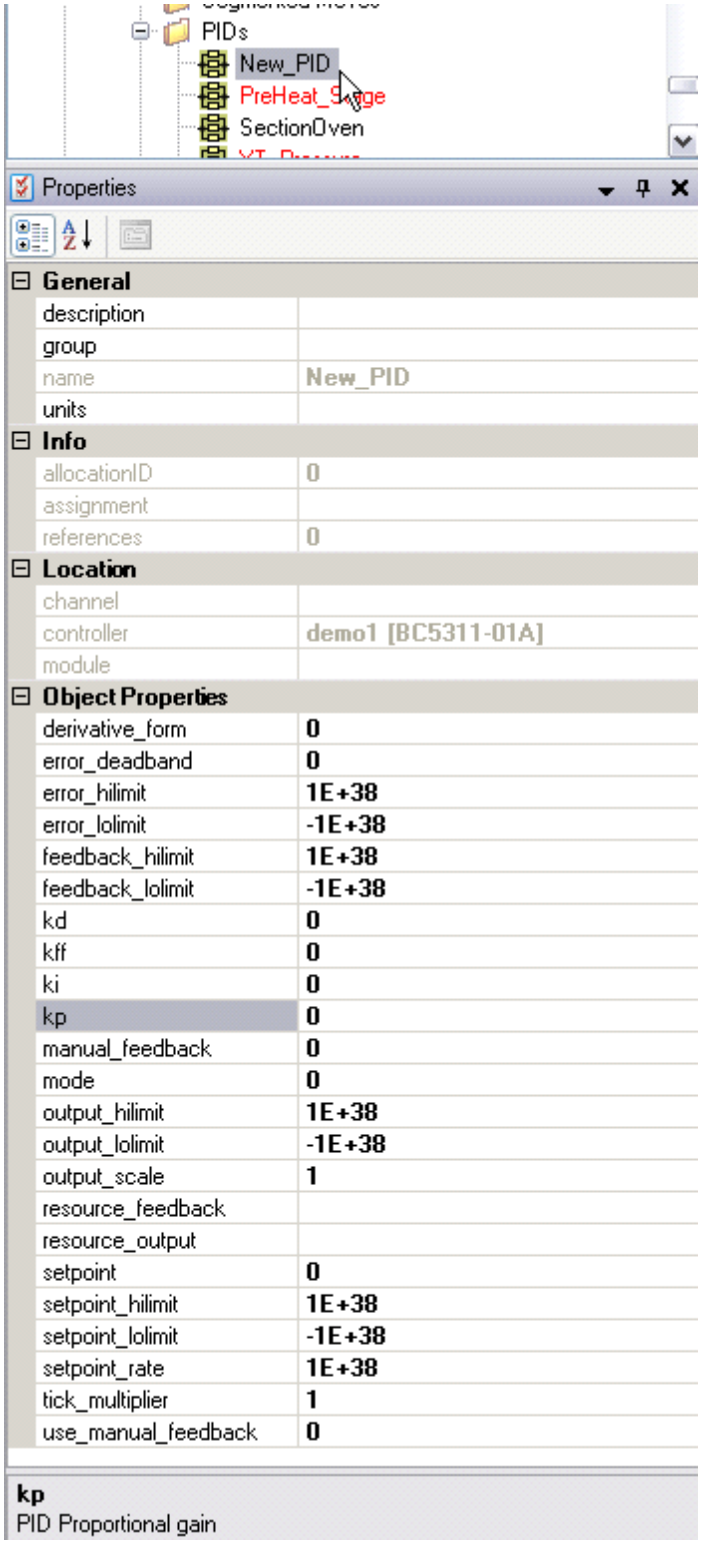
At this point a new PID loop object will be added to the selected controller. The PID object should now be given a meaningful name and then you will be ready to set up its parameters.



For this example, we will name the PID object "New_PID." Just like the other resources in the Resource window, when you highlight a PID Object, its properties are automatically brought up in the Properties window. The screen capture on the following page shows the Properties window for our New_PID.

You will notice that all of the Object Properties are preloaded with default values except for *resource_feedback* and *resource_output*. These properties must be associated with actual controller resources, and a pop-up selection window is provided for this purpose.





The Properties window for New_PID

2.4 PID Object Properties

The following properties can be set in the Properties window of QuickBuilder. Most can also be altered in QS4 code as well through *dot property notation*:

```
pidname.property
```

Only items marked [REQUIRED] need be filled in. All other parameters are optional and need only be applied where they improve or are required for the process.

- **derivative_form**: When this parameter is set to a non-zero value, the PID algorithm is followed by an additional derivative. This is used when the process being controlled is self-integrating.
- **error_deadband**: This value controls when the loop ignores small values of *error*. The absolute-value of the *error* is compared to the value specified. If the absolute error value is less than or equal to this value, the *error* for this update is set to be zero.
- **enabled**: Controls whether or not the PID loop is active. When set to a zero value, the loop is effectively disabled, since only the *offset* is routed to the output limiter.
- **error_hilimit**: Limits the maximum value of the error fed into the PID equation.
- **error_lolimit**: Limits the minimum value of the error fed into the PID equation.
- **feedback_hilimit**: Limits the maximum value of the feedback signal fed into the error calculation.
- **feedback_lolimit**: Limits the minimum value of the feedback signal fed into the error calculation.
- **integrator_unwind_constant**: A factor that determines how fast the integrator should self-discharge when either the *output* is in limit or the value for *ki* is set to 0.

  A value of 1.00 (the maximum) means that the integrator should hold its last value and not discharge in those two cases. A value >0 but <1 discharges the integrator by multiplying the integrator by that value each update.
- **kd**: Constant that determines the derivative gain.
- **kff**: Constant that determines the feed-forward gain. Provides improved response when the setpoint value is changed.
- **ki**: Constant that determines the integral gain.
- **kp**: Constant that determines the proportional gain.
- **manual_feedback**: The value that is used in place of *resource_feedback* when the *use_manual_feedback* parameter is set equal to a non-zero value.
- **mode**: Reserved for future use.
- **offset**: Offsets the generated output value. Can be used in conjunction with *enabled* to force the output of the PID to a specific value (by setting *enabled* to 0 and the value to force the PID output to into the property *offset*).
- **output_hilimit**: Limits the maximum value of the PID loop output (before output scaling).
- **output_lolimit**: Limits the minimum value of the PID loop output (before output scaling).
- **output_scale**: The PID output is multiplied by this value to get the *resource_output* value. Setting this equal to -1 effectively negates the output value when required.
- **resource_feedback**: This is the controller resource used for feedback to the PID loop. **[REQUIRED]**

- **resource_output**: This is the controller resource connected to the scaled output of the PID loop. **[REQUIRED]**
- **setpoint**: The desired initial value for the setpoint.
- **setpoint_hilimit**: Limits the maximum allowable value for the setpoint.
- **setpoint_lolimit**: Limits the minimum allowable value for the setpoint.
- **setpoint_rate**: Limits the rate at which a change in setpoint is presented to the system (/sec).
- **tick_multiplier**: All Model 5300 CPU modules have a settable tick rate (default tick = 50ms) that is used to limit how fast it performs certain operations such as filtering analog I/O points. The PID update rate = (controller tick) * (tick_multiplier).
- **use_manual_feedback**: If set = 0, then *resource_feedback* is used. If set = 1, then *manual_feedback* is used.

2.5 Accessing Properties in QS4 code

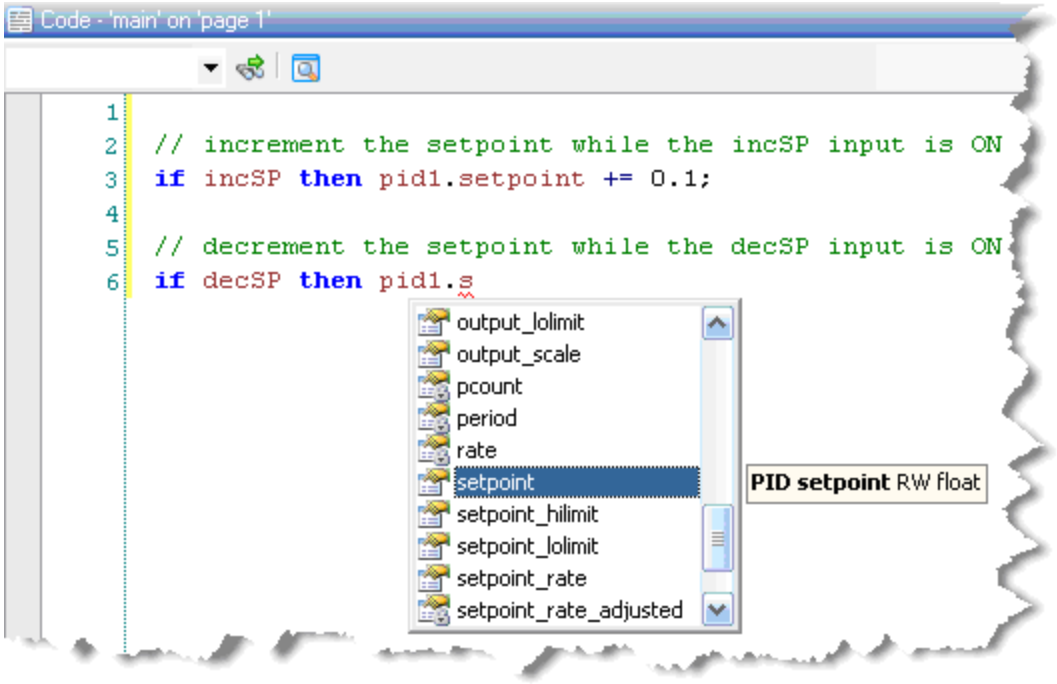
The screen shot below shows *dot properties* for a PID object named "pid1" being accessed in the Code window of QuickBuilder. This is a very powerful feature of the PID object that lets the application designer manipulate most aspects of the PID loop under program control. The selection box shown below pops up automatically as soon as the period key is pressed after typing a PID object name. Dot properties enable object properties to be accessed directly in code. For example, `pidOne.rate` refers to the loop rate of the PID object named `pidOne`.

Note that the bubble help also tells what type of variable is used for the property: in this case *setpoint* is a read/write floating point value.

The properties listed below can only be accessed in QuickBuilder code via the dot properties. They cannot be set in the Property Window, as they are read-only values, or they are computed on-the-fly by the PID loop.

- **error**: Current (nth value) calculated error value after limiting and deadband.
- **error0**: Previously (n-1) calculated error value after limiting and deadband.
- **error1**: Previously (n-2) calculated error value after limiting and deadband.
- **feedback**: Current value of feedback (manual or resource) after limiting.
- **in_error_limit**: True when the error value is currently being limited.
- **in_feedback_limit**: True when the feedback value is currently being limited.
- **in_output_limit**: True when the output value is currently being limited.
- **in_setpoint_limit**: True when the setpoint value is currently being limited.
- **integrator**: The current value of the integrator (derivate-form=0 only).
- **iperiod**: An internal PID value used to determine the PID period.
- **output**: Value of the PID output after scaling and limiting.
- **pcount**: PID processed counter – holds the number of times the PID loop has run.
- **period**: the actual PID period (sec) – a computed read-only value.
- **rate**: the actual PID rate (Hz) – a computed read-only value.
- **setpoint_rate_adjusted**: The rate-adjusted setpoint value.
- **subtick**: PID sub tick – counts up to tick_multiplier.

Here is an example of dot properties in QS4 code:



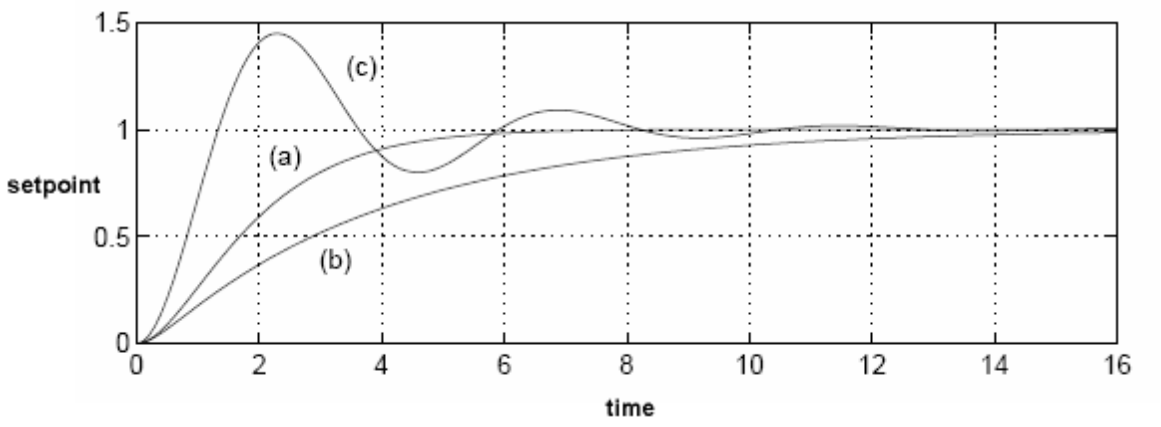
3 Appendix A: PID Loop Tuning

The following table gives general guidelines as to the effect of changing the PID tuning gains.

This is only a general guideline, because there are interdependencies between these variables and changing one will impact the other two.

PID Factor	Rise Time	Overshoot	Settling Time	Steady State Error
larger kp	Decreases	Increases	Small Effect	Decreases
larger ki	Decreases	Increases	Increases	Eliminates
larger kd	Small Effect	Decreases	Decreases	Small Effect
larger kff	Decreases	May Increase	Generally Decreases	No Effect

Tuning Response Curves



The plot above shows the system response based on three different tuning setups based on a setpoint change from 0 to 1.

a) Critically Damped: The optimally tuned system is shown in curve (a). This system is said to be critically damped. It does not overshoot the setpoint value and settles quickly (6 seconds) at the new setpoint value.

b) Over Damped: Curve (b) shows a system that is over damped. It does not overshoot the setpoint; however, it takes too long to reach the desired setpoint.

c) Under Damped: Curve (c) shows a system that is under damped. It overshoots the setpoint and then oscillates around the setpoint**.**

QuickScope Reference Guide

Copyright © 2004 - 2010 Control Technology Corp. All Rights Reserved.

Control Technology Corp.
25 South Street
Hopkinton, MA 01748
Phone: 508.435.9595 • Fax 508.435.2373

Document No. 951-530032-005

⚠ **WARNING:** Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.

The information in this document is subject to change without notice. The software described in this document is provided under license agreement and may be used and copied only in accordance with the terms of the license agreement. The information, drawings, and illustrations contained herein are the property of Control Technology Corporation. No part of this manual may be reproduced or distributed by any means, electronic or mechanical, for any purpose other than the purchaser's personal use, without the express written consent of Control Technology Corporation. Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

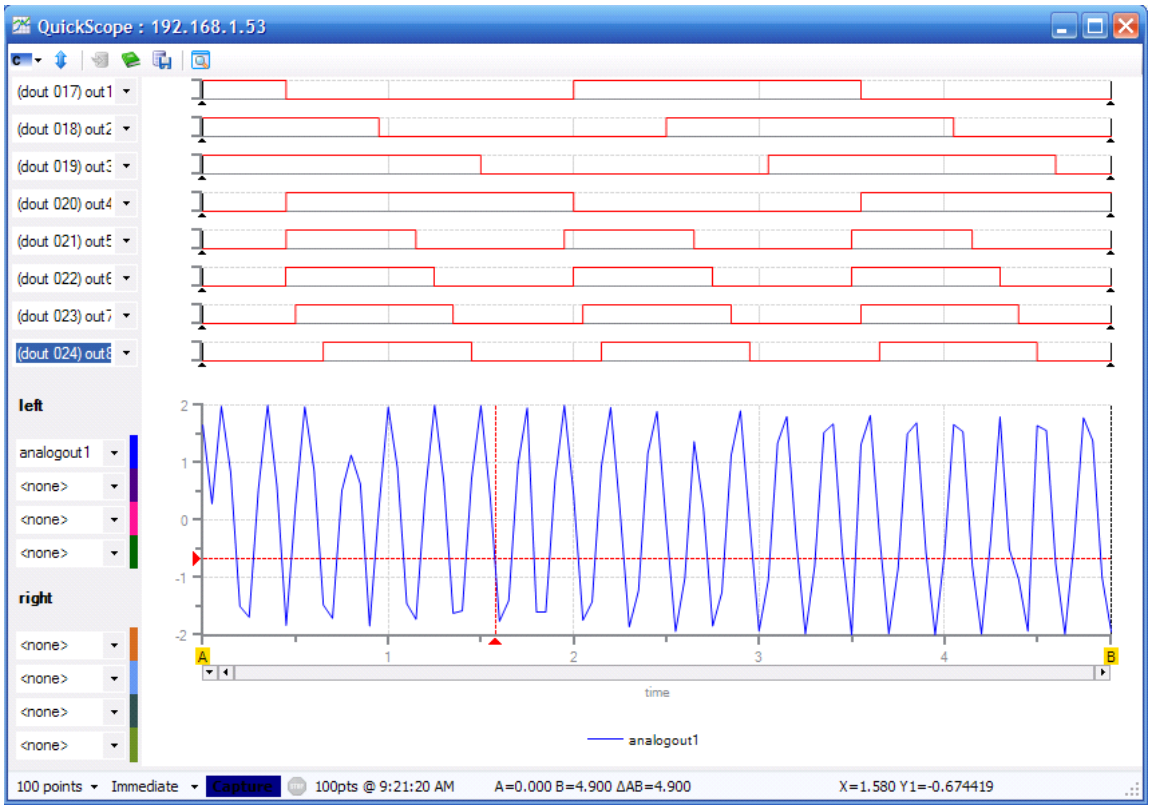
While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions.

See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

1 Chapter 1: Overview

This document will introduce you to the powerful QuickScope tool available for CTC's 5300 series controller. QuickScope is a graphical "digital scope" and extremely useful debug tool.



2 Chapter 2: QuickScope and QuickView Features

There are two components to QuickScope.

- QuickScope (QS) captures data and displays it in a graphic format.
- QuickView (QV) displays and allows editing of data in a tabular format. This is similar to but better than CTCMon since it doesn't deal with registers, but with named resources.

QS & QV interrogate a running program to find out the named resources within the controller.

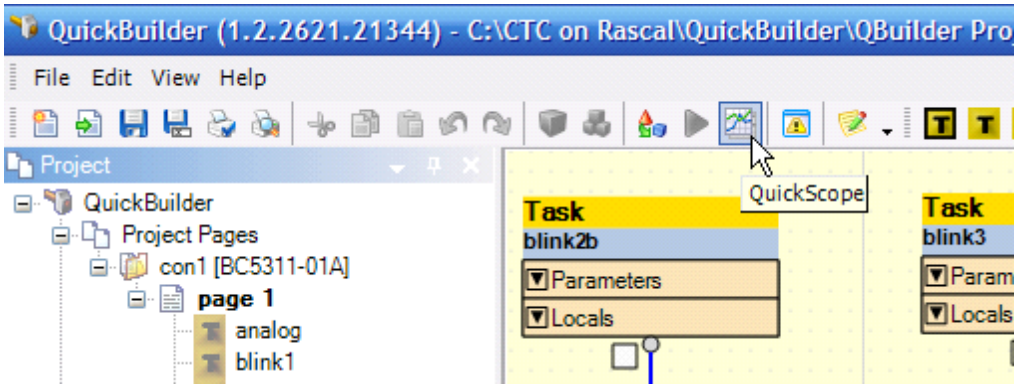
QS & QV will always interrogate the controller for these symbols when connecting to the controller. This means that these named resources are "always" right – there can be no "out of sync" issues as in the old QS2 way of using symbols.

QS & QV can be started as stand-alone applications to monitor the operation of the controller or from within QuickBuilder.

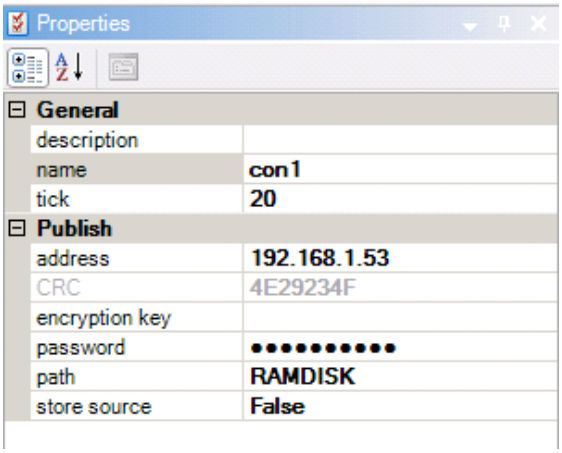
QS Captures can be initiated manually from QuickScope or triggered from within a QuickBuilder (QS4) program using a `$TRIGGER = 1;` command. Refer to the *QuickBuilder QuickStart Guide* for additional information.

2.1 Invoking QuickScope

QuickScope is invoked by clicking on the QuickScope icon shown below.



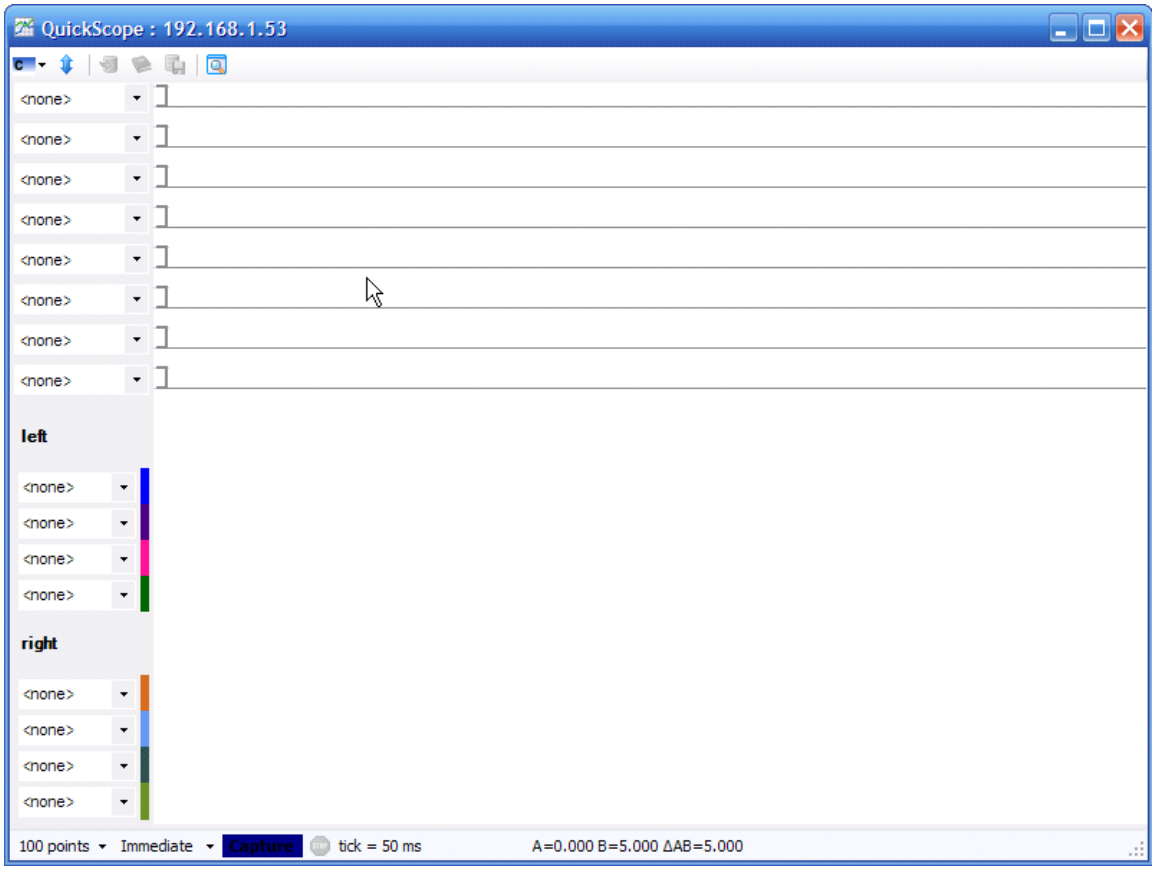
The rate between data captures is determined in QuickBuilder by the **tick** property for the controller. Any adjustment to this rate must be translated, published, and run before the new rate will be implemented in future captures.



The **tick** rate is set by selecting a controller in the resource window, and then adjusting its **tick** parameter. The tick rate can be set as low as 10 ms, with a default of 50 ms.

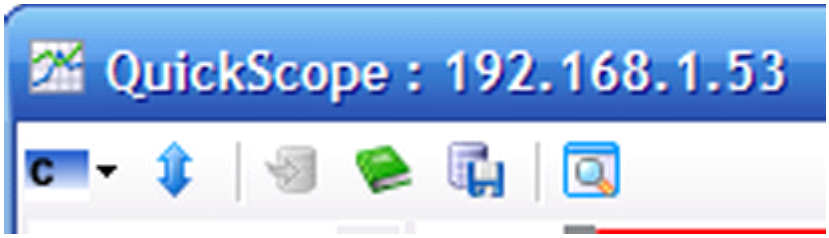
☑ **Note:** Capturing too much data at a very fast rate may impact the performance of the application project.

The following screen appears once you invoke QuickScope:



2.2 Toolbar Summary

There is a toolbar at the top of the QS window:



1. The first drop-down button selects the controller to connect to. Controllers are auto-discovered just like they are in WebMon 2.0. There is a menu item to add a controller for controllers that cannot be auto-discovered (for example, remote controllers not on the same subnet).

2. The second button re-synchronizes the symbol table. At the present moment, the symbol table is only read once when QS connects to the controller. If you change the program, you will be alerted to re-synchronize the symbol table.

   ☑ **Note:** Some time in the future, this manual re-sync will no longer be necessary as QS (and QV) will "know" that the symbol table has been modified and inform the user that it will now resync on its own.

3. The third button imports saved trace data for re-display.

4. The fourth button will produce a PDF report of the trace data in graphical form.

5. The fifth button writes the captured data as an XLS (Excel) file – not a CSV file. This allows the user to further analyze the captured data. All named logical resources (as well as the "main" user-specified resources) are written to the file – not the selected logical-traces.

6. The last button brings up a QuickView window for the selected controller.

2.3 Status Bar Summary

There is a status bar at the bottom of the QS window:



The *Status* bar allows you to:

1. Choose how many points to capture.

2. Choose *how* and *when* to capture. **Immediate Capture** means to capture when the capture button is clicked. **Triggered** means to capture when a signal is generated by the QS4 program using this statement :

```
$TRIGGER = 1;
```

If already in a capture, re-triggers are ignored.

3. Stop the capture (when in capture mode). By clicking the **Stop** button, the capture is aborted and no capture data (even partial) is returned.

4. Next to the **Stop** button in the lower *Status* bar, there is a readout that indicates the current status. Initially when a connection is made to the controller, the tick period is displayed. This is the capture period *per point* for the data capture. When capturing, there are several messages displayed here:
   - **Waiting** — This means that another capture is in progress (perhaps by another QS program) and that it is waiting until that one completes.
   - **Initing** — This means that trace set-up data is being sent down to the controller in preparation for a capture.
   - **Capturing…** — This means that the controller is recording data. A percent complete is displayed, as it can take a while for some tick values and "# of points" to process.
   - **Wait4Trig** — This means that **Triggered** mode was selected, and the controller is waiting for a QS4-based signal (see item 2 in this list).
   - **Loading** — This means that the capture is complete and QS is retrieving the captured data points.

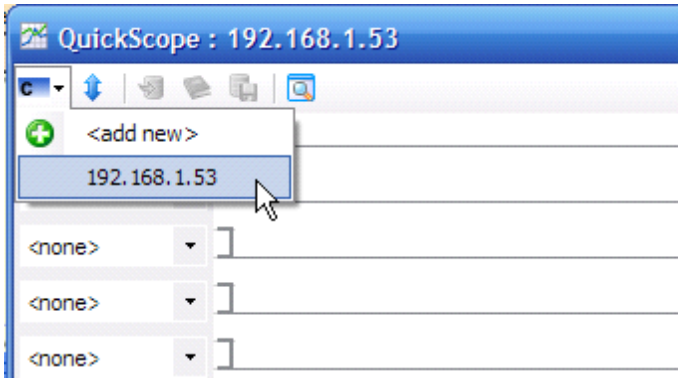
5. In the middle of the *Status* bar you will find readouts for the A and B cursor as well as the difference between them. The A and B cursors can be moved by dragging them from their initial full-right and full-left positions. They can be moved in either the upper or lower plot areas – they are vertically synched between the two plots. This is a fast and accurate way to measure between two items.

6. At the far right in the *Status* bar there are **X**, **Y1**, and **Y2** values displayed. These are used with the red crosshairs in the lower *Main* trace window. These allow you to measure the value (both X & Y) for each of the two axes of captured data.

2.4 Connecting to a controller

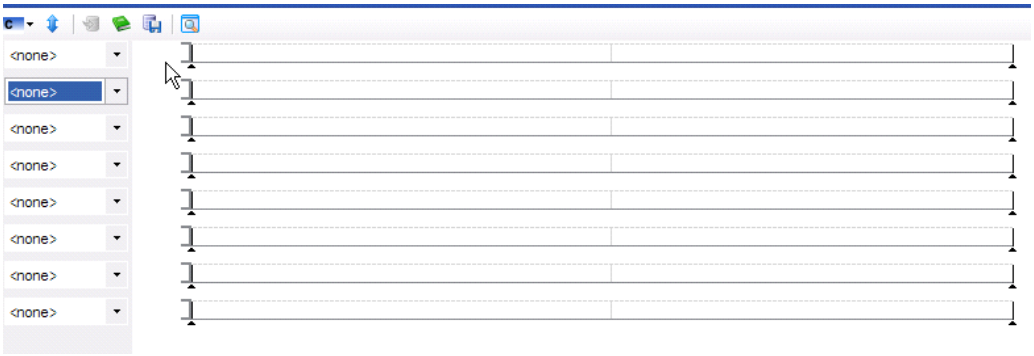
QuickScope should interrogate your network and find the available controllers. You can click on the **Connect to Controller** icon and select the controller you want to connect to.

If you do not see the controller you want to connect to, simply select **<add new>** and type in the address of the controller manually.



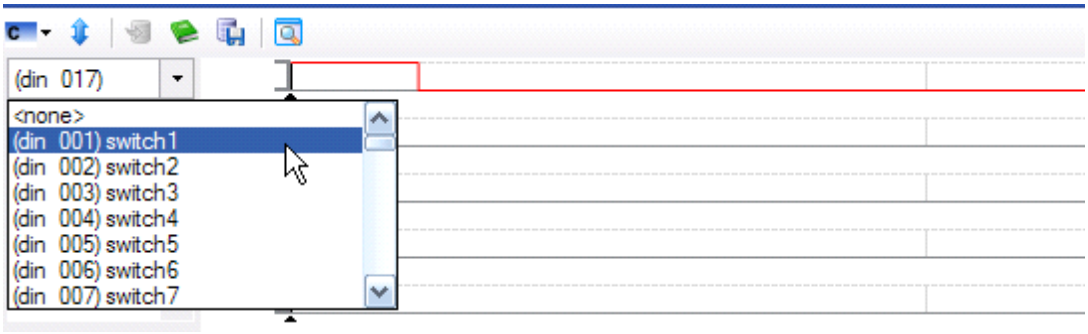
2.5 Setting up traces

The QS window consists of top and bottom trace areas. The top (shown below) consists of 8 "logical" trace charts that allow you to select any of the first 64 inputs or outputs.

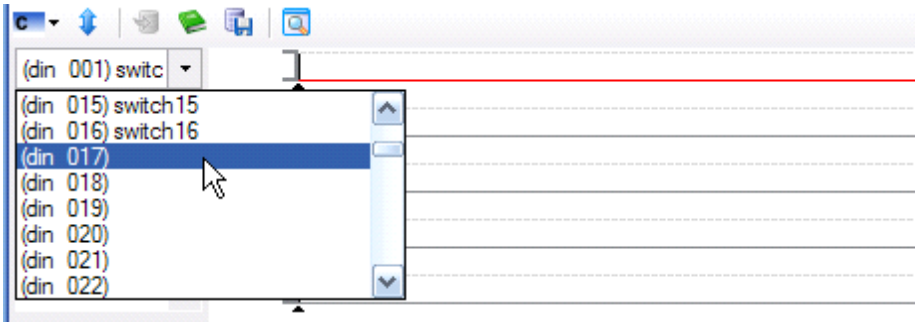


You can select any of the first 64 I/O to be displayed in these 8 trace windows *even after* a capture since the first 64 digital inputs **as well as the first** 64 digital outputs are *always* captured.

When an IO point is named (from the running QS4 program), its name will appear in the dropdown trace selector combo box to the left of the trace as shown below.

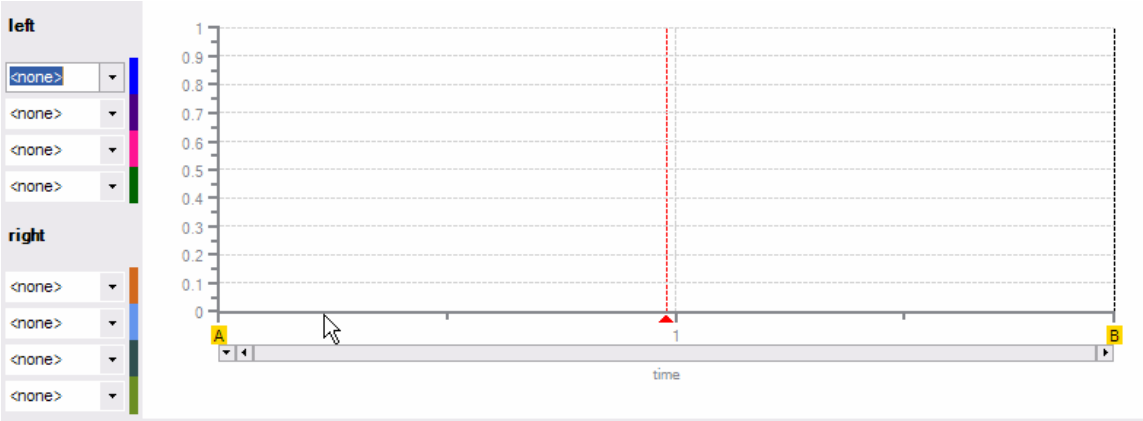


You can also select unnamed I/O in the logical trace window by selecting its input or output channel from the dropdown menu.



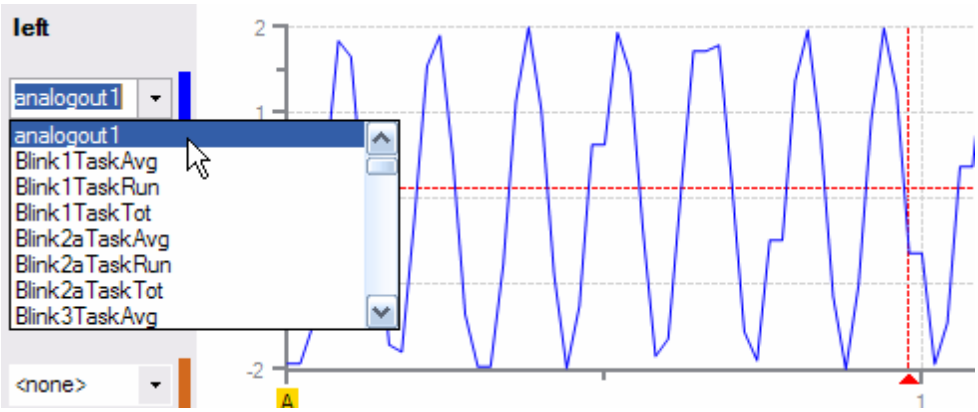
☑ **Note:** If you need to capture an IO point *beyond* the first 64 ins/outs, the lower *Main* trace combo box selectors need to be used ***prior*** to capturing the data.

The lower *Main* trace (shown below) allows you to capture 8 additional resources of your choice.



☑ **Note:** Traces in the lower *Main* trace must be set up *prior* to a capture by using the 8 combo boxes to the left of the main trace window.

You will be able to choose from all variables and *named* analog and digital I/O in this lower *Main* trace area.

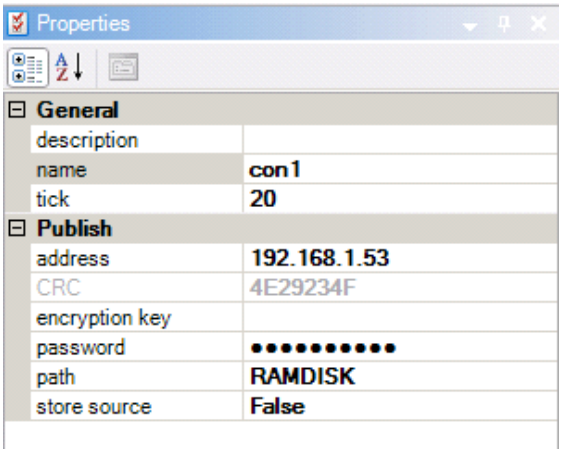


The lower Main trace window allows these 8 items to be grouped in two scalings: left and right.

- If you need to capture some analog inputs (e.g., -10 to +10V), you may want to put those on the left axis so they all have the same scale factor. Then you can use the right axis for something else, perhaps something that is not close in value to +/-10.
- The left and right axes in the *Main* trace scale independently and automatically.

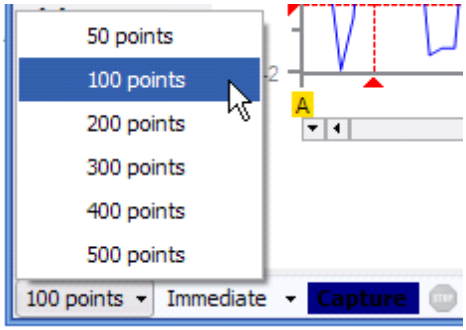
2.6 Capturing Data

The Controller's **tick** property allows you to set the capture rate within QuickBuilder.

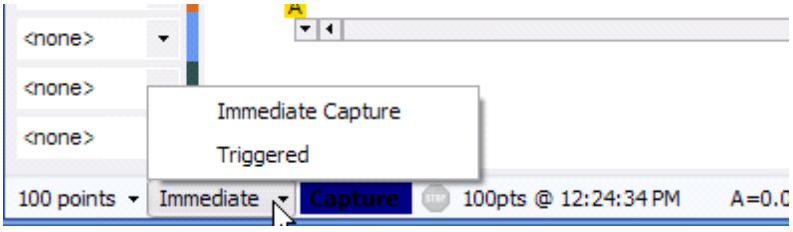


☑ **Note:** Data points will be captured each tick for the 128 digital I/O as well as up to 8 variables for the lower Main trace window. Capturing this amount of data does consume processor resource and users should be careful not to set the tick rate too low, as this could impact the step execution time of the QuickBuilder program. In general, these effects are minimal for tick rates greater than 20 ms.

QuickScope's *Status* Bar allows you to set the number of points to collect during a capture.



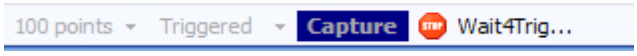
As mentioned in the *Status* bar Summary section, there are two ways to capture data:



Selecting **Immediate Capture** means data will be captured when the **Capture** button is clicked.

Selecting **Triggered** means data will be captured by the QS4 program when the following statement is generated:
`$TRIGGER = 1;`

Once you select trigger mode, you must click on **Capture** in QuickScope. You will then see the following in the *Status* screen:

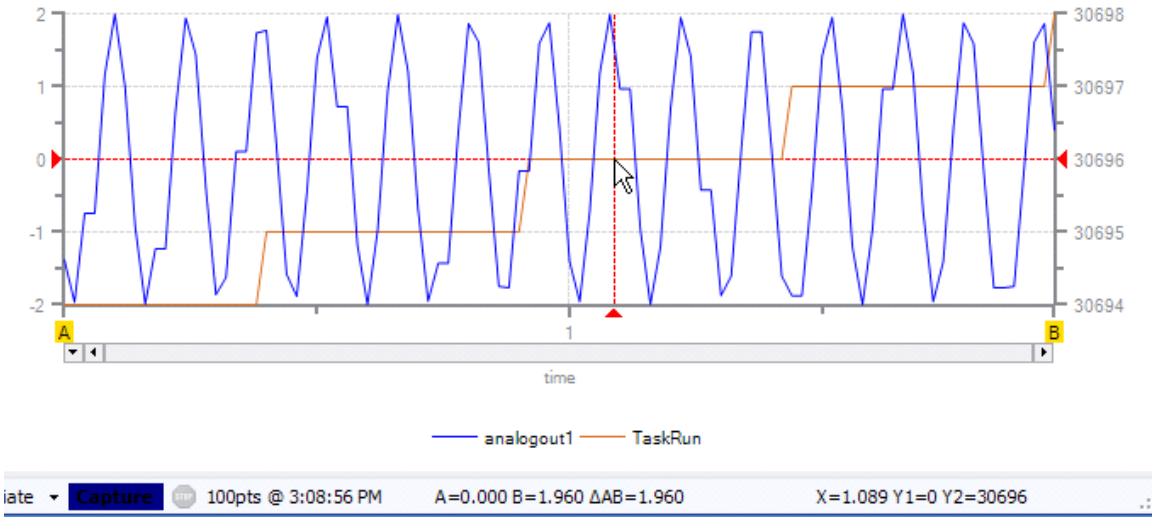


Once your Quickbuilder code initiates the trigger (`$TRIGGER = 1;`), you will see the following and QuickScope will display the captured data when it is completed:



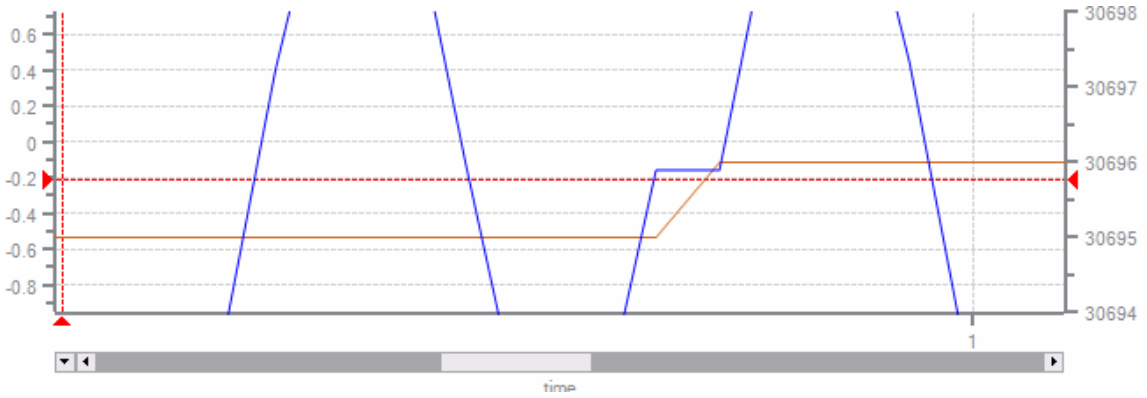
2.7 Evaluating Data

X, Y1, and Y2 values are displayed at the far right in the *Status* bar. These are used with the red crosshairs in the lower *Main* trace window. These allow you to measure the value (both X & Y) for each of the two axes of captured data. Y1 will represent your left trace Y values, and Y2 will represent your right trace Y values. When X represents time, the units will be in seconds.



2.7.1 Zoom

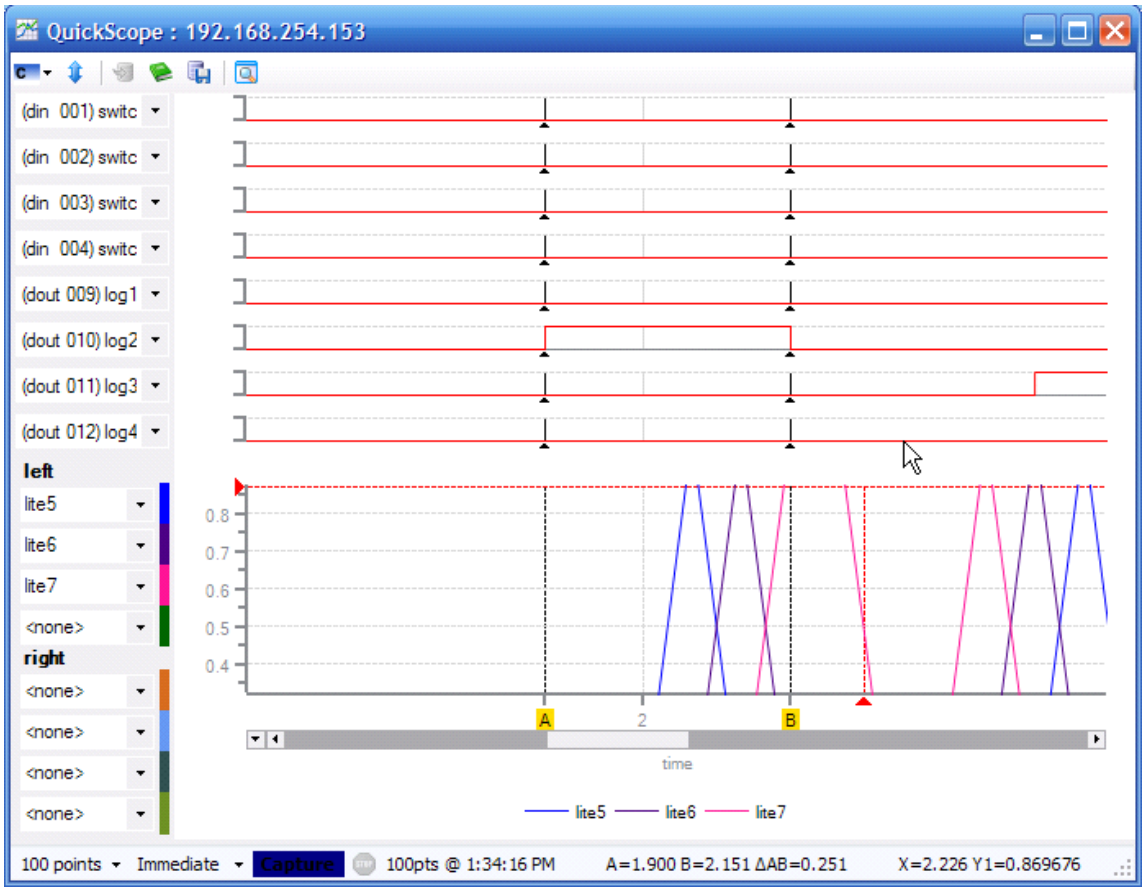
You can also zoom in to get a more precise X, Y reading to an area by clicking and dragging the two desired corners of the window you would like to zoom into.



You can zoom back out by double clicking anywhere outside the lower *Main* trace area.

2.7.2 A and B Cursors

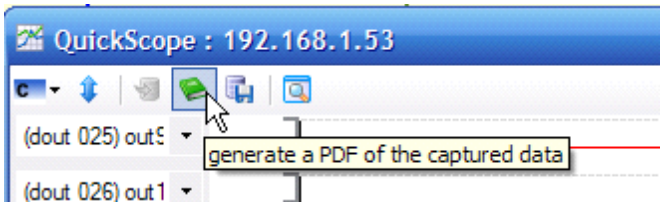
The **A** and **B** cursors are great for obtaining more precise information. In the example below, the trace has been zoomed in on and the **A** and **B** cursors have been dragged and dropped to measure the time that **log2** was on. The bottom of the *Status* bar shows the deltaAB result as being 0.251 seconds.



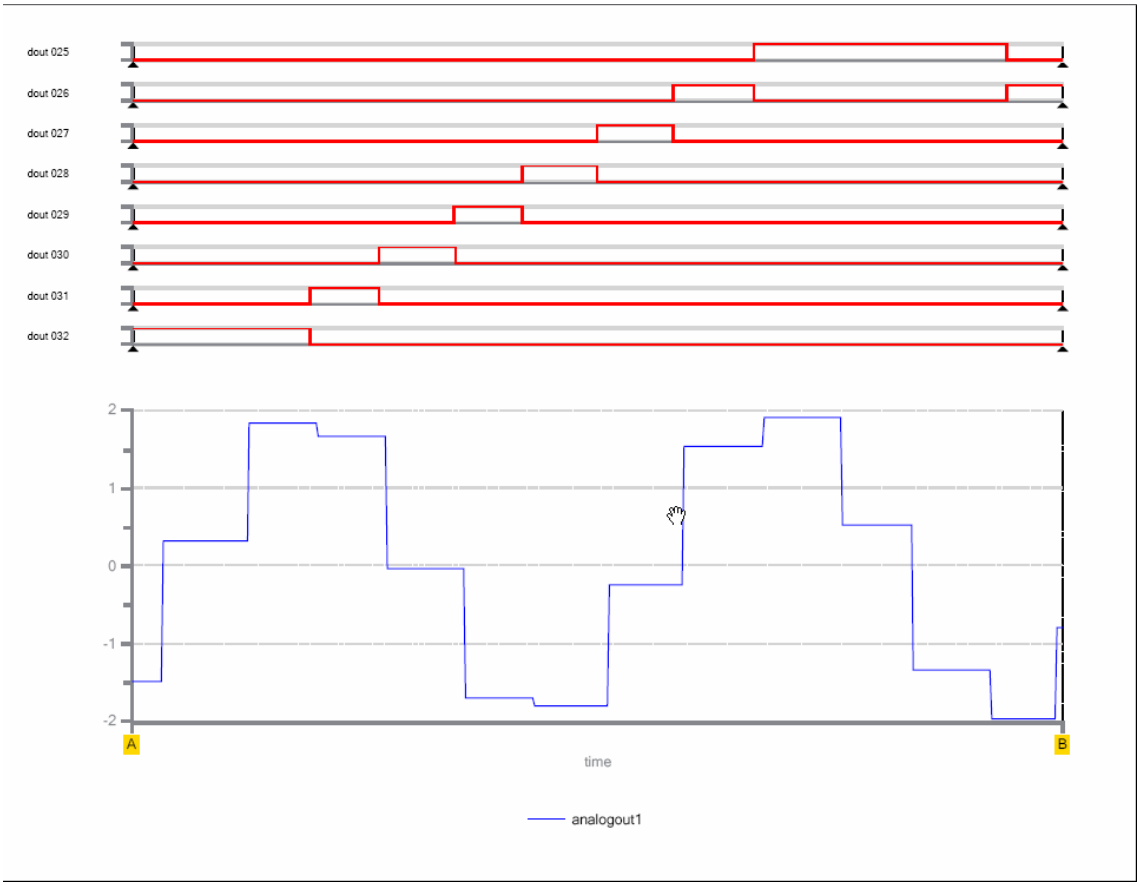
☑ **Note:** When you zoom in on a chart, the yellow A and B cursor handles may not be visible. To re-align them with edges of the current view, simply click on the A and B read out area in the *Status* bar.

2.8 Creating a PDF file

You can create a PDF file using the **generate a PDF of the captured data** icon shown below:

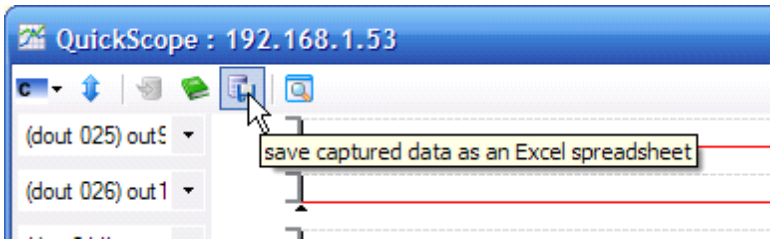


This will create a PDF showing traces of all named digital I/O in the upper trace area(s) and the selected traces in the lower trace area. If you have more then 8 named digital I/O among the first 64 inputs and outputs, your PDF will have multiple pages.



2.9 Creating an Excel Spreadsheet

You can create an Excel spreadsheet file using the **save captured data as an Excel spreadsheet** icon shown below:

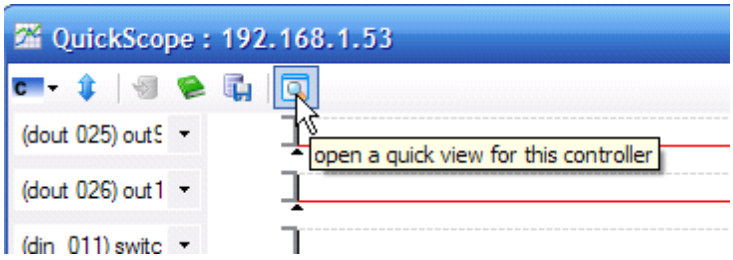


The format of the Excel spreadsheet created appears as follows (time is in units of seconds):

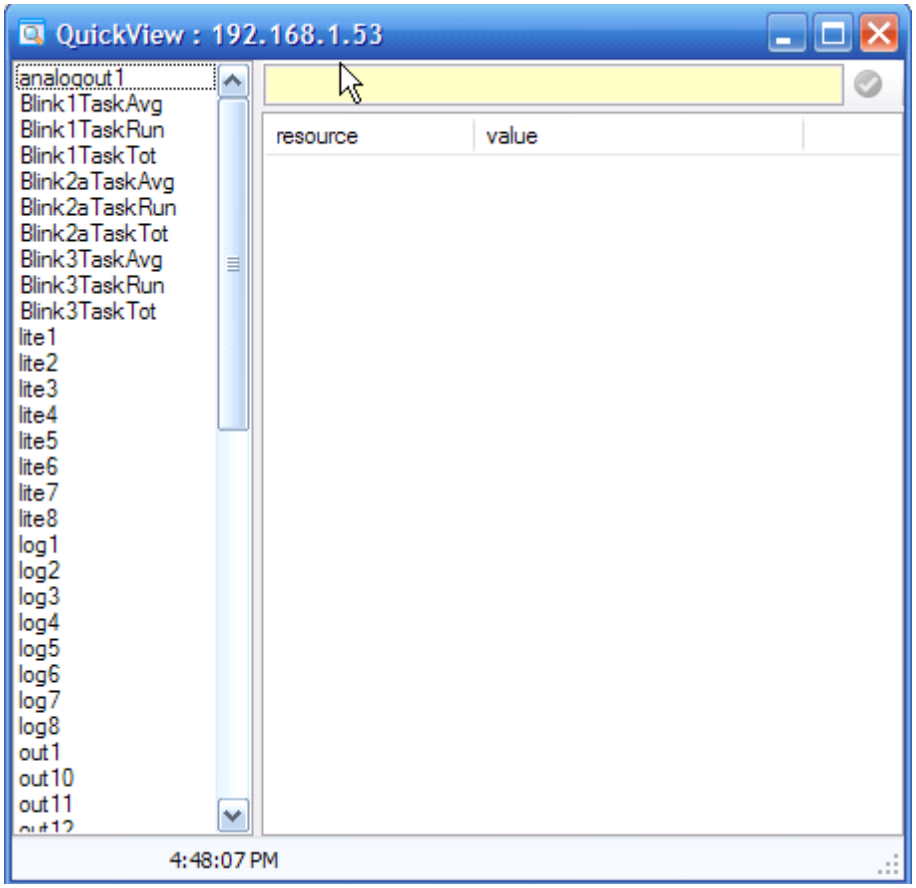
	A	B	C	D	E	F	G
1	time	analogout1	TaskRun	<none>	switch1	switch2	switch3
2	0	-1.3794	30694	0	1	0	0
3	0.020003	-1.9639	30694	0	1	0	0
4	0.040012	-0.74281	30694	0	1	0	0
5	0.059998	-0.74281	30694	0	1	0	0
6	0.080001	1.161222	30694	0	1	0	0
7	0.1	1.99763	30694	0	1	0	0
8	0.120006	0.997426	30694	0	1	0	0
9	0.140004	-0.91981	30694	0	1	0	0
10	0.16	-1.99137	30694	0	1	0	0
11	0.180003	-1.23208	30694	0	1	0	0
12	0.199995	-1.23208	30694	0	1	0	0

2.10 QuickView

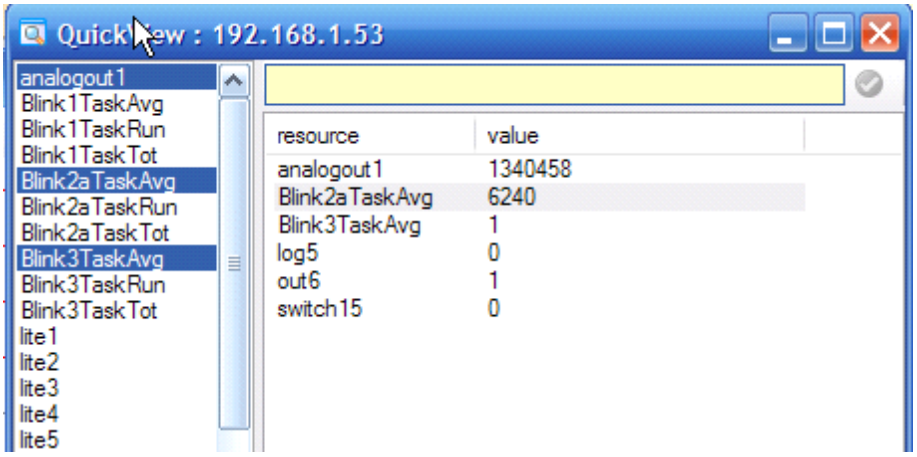
To open QuickView, click on the **open a quick view for this controller** icon from QuickScope:



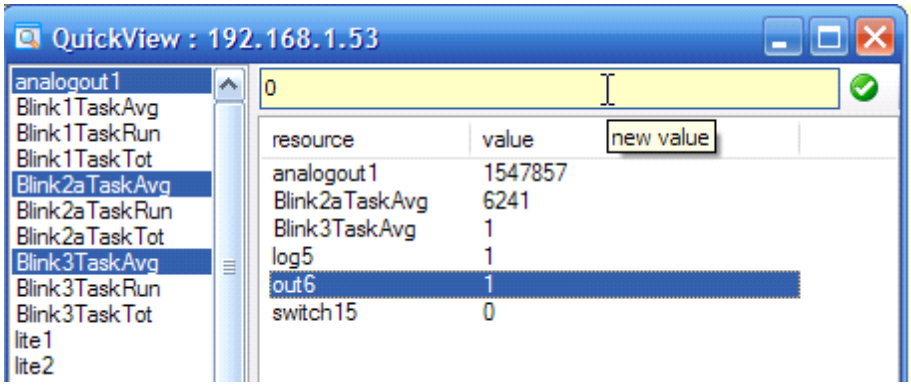
Notice the left side of the screen displays a list of all named resources.



Click on the resources you would like to monitor and they will be added to the right side of the screen along with their values as shown below.

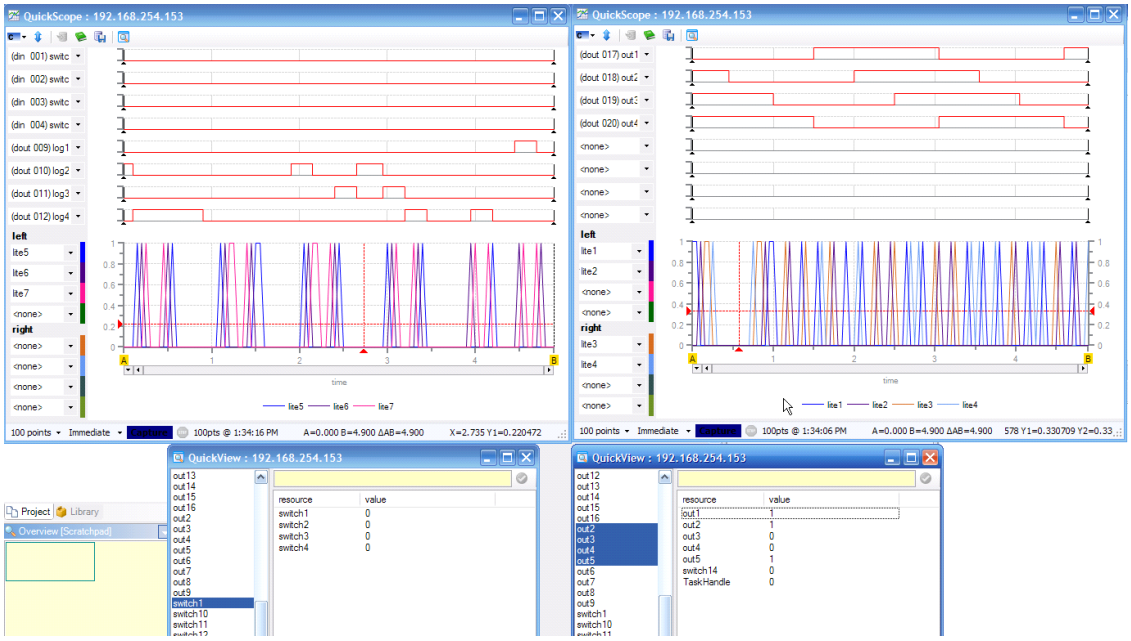


To write a value to the controller, click on the resource you want to change and then enter the new value as shown below. Then click on the green check icon.



2.11 Multiple Windows

You can open multiple instances of QS and QV as shown below. This will allow you to track more resources and monitor more than one controller at a time.



Index

- A -

addr function 77
AnalogInput 65
AnalogOutput 65
arc sine 73
arc tangent 73
arrays 66, 69
axis module 166, 167
axis object: 65, 166, 168
  statements 98
axis properties:
  acc/dec 175
  cmode 175, 250
  driveenable 175
  imposw 175
  neglim/poslim 175
  overnegin/overposin 175
  perrlimit 175
  ppr 175
  tmax 175, 250
  uun/uud 175
  vmax 175, 250
axis setup: 174
  operating modes 180
  positioning 180
  slewing 180
  tracking 180

- B -

begin statement 59, 97
begin step 59
bit function 77
boolean operators 72
break statement 94

- C -

calculations, in expressions 72
call statement 56, 89
camming and data table commands:
  loading tables 227
  manipulating master position 238
  manipulating tables 231
  reading and writing data to/from tables 235
  using data from Excel spreadsheets 237
camming and data tables: 224
cancel statement 97
clrbit statement 88
coding, in QS4 52
command outputs 151
comments, in QS4 86
common bits 194
common variables 194
constants 11, 67
continue statement 94
controller resources 65
cosine 73

- D -

decision step 59
delay statement 95
DigitalOutput 65
do statement 96
do step 57
document:
  general info (QuickBuilder Reference) 7
  general info (QuickMotion Reference) 148
  general info (QuickScope Reference) 323
  version number (QuickBuilder Reference) 7
  version number (QuickMotion Reference) 148
  version number (QuickScope Reference) 323
done statement 60, 98
done step 60

- E -

enable, disable (event) statements 96
encoders 151
events:
  asynchronous 54
  definition of, in QS4 54
  parallel 54
expressions:
  - 72
  ! 72
  != 72
  % 72

expressions:
& 72
&& 72
* 72
/ 72
^ 72
| 72
|| 72
~ 72
+ 72
< 72
<< 72
<= 72
== 72
> 72
>= 72
>> 72
addition 72
binary - 72
binary + 72
boolean constant 72
division 72
logical-and 72
logical-or 72
modulus 72
multiplication 72
pi 72
remainder 72
subtraction 72
unary - 72
unary + 72

- F -

Fault Codes:
Codes, EtherCAT 294, 295, 296
Codes, General 293, 294
MF_ABORT_REQUESTED 295
MF_BADARGUMENT1 293
MF_BADARGUMENT2 294
MF_BADARGUMENT3 294
MF_BADARGUMENT4 294
MF_BADINPUTNO 293
MF_CANTCONSUME 294
MF_DCSYNC 294
MF_DRIVE_ERR_MSG 294
MF_ECAT_OFFLINE 294
MF_ERROR_DC_RESTORE 296
MF_ERROR_DC_SYNC01 296
MF_ERROR_ILLEGAL_VAL 296
MF_ERROR_INIT_CONFIG 295
MF_ERROR_INIT_PDO 296
MF_ERROR_NO_INIT 295
MF_ERROR_NO_OPERATIONAL 296
MF_ERROR_NO_PREOP 295
MF_ERROR_NO_SAFEOP 295
MF_ERROR_NO_SERVO 295
MF_ERROR_NOT_ONLINE 296
MF_ERROR_OO_RANGE 295
MF_ERROR_REMAP_CONFIG 296
MF_FGMSBLIMIT 293
MF_FOLLOWERR 293
MF_GENERICFAULT 293
MF_INVALID_ECAT_MODE 294
MF_INVALID_TBL_OP 293
MF_INVALID_TBL_OP_Y 295
MF_INVALID_TBL_OP_Z 295
MF_INVALID_TRACKING 295
MF_INVALIDACC 293
MF_INVALIDDEC 293
MF_INVALIDRATE 293
MF_INVALIDTIME 293
MF_INVALIDVEL 293
MF_MOTIONACTIVE 293
MF_NO_ERROR 293
MF_NOCAMFILE 294
MF_NOMSBFILE 294
MF_NOTENABLED 293
MF_NOTINSLEW 293
MF_NOTINTRACK 294
MF_ONLYINBG 293
MF_READONLY 295
MF_REMOTE_READ 294
MF_REMOTE_WRITE 294
MF_SDO_READ 294
MF_SDO_WRITE 294
MF_SESGMOVE_ERROR 294
MF_SESGMOVE_SIZE 294
MF_STACK_ERROR 295
MF_TIMEOUT 295
MF_UNIMPLEMENTED 293
MF_WRONGMODE 293
flags, in QS2 88
flowcharting, in QS4 52, 56
for statement 93
function 56

function definition 56

- G -

global definitions 11
goto next statement 89, 90
goto statement 57, 89, 90

- H -

hyperbolic 73

- I -

icons used in this manual 150
icons, in QuickBuilder toolbar 53
if/then/else statement 90
indirect variables 69
indirection 69, 77
interpolation, for splines and CAM tables:
  cubic 224
  linear 224
  quadratic 224
isdone function 77

- K -

knots 224

- L -

local definitions 11
local variables 54

- M -

M3-40A servo module: 150, 153, 155, 167
  LED mapping 158
  pinouts 158
M3-40B stepper module:
  LED mapping 159
  pinouts 159
M3-40C stepper module:
  LED mapping 160
  pinouts 160
M3-40D servo module 150
mathematical operations 72
MF_ERROR_DC_RESTORE 293
Model 5300 controller 153, 162, 171
monitor statement (QS2) 95
motion control programming: 182, 187, 197, 205, 215, 219
  and QuickStep 180
  operators 181
motion control:
  getting started 174
  statements 98, 170
  tuning 176, 177, 178
  tuning wizard 176, 177, 178
motion sequence blocks (MSBs): 61, 153, 165, 166, 180
  and QuickStep 248
  background MSBs 169, 170
  foreground MSBs 169, 170
  sample code 302
  variables 248, 250

- N -

non-scalar variables (arrays) 69
numeric assignment statement 86
numerical functions 73
NVariable 65, 66

- P -

PID:
  definition 311
  diagram 311
  loop 311
  theory 311
  variables 311
positioning mode 180
programmable limit switch (PLS) 155
programming:
  branching 90, 95
  jump (goto) 89
  looping 92, 93
  structure 52

- Q -

QS2 (QuickStep2): 8

QS2 (QuickStep2): 8
differences from QS4 52, 88, 90
QS4 (QuickStep4): 8, 52, 162, 164, 248
begin step 59
color codes, for text 86
decision step 59
do step 57
done step 60
flow 56
function definition 56
hardware compatibility 150
max tasks 71
motion control statements 170
shortcut keys 144
start statement 170
steps 56
stop statement 170
task definition 54
transitions 56, 95
QS4 system variables:
$CBITS 79, 82
$CVARS 79, 82
$DINPUTS 79, 80
$DOUTPUTS 79, 80
$REGISTERS 79, 81
$TASKTIMER 79
$TRIGGER 79, 81
QuickBuilder 8, 162, 163
QuickBuilder PID:
features 313
PID loop algorithm 314
PID object 315
PID object, properties 318, 319
PID tuning 321
QuickMotion 162
QuickMotion commands:
abort 188
asynchronous event handling 189
clrout 197
counter = expression, offset 202
counter read, write, offset 202
delay 183
drive disable 183
drive enable 183
end 188
gear at (ratio) 215
gear at (ratio, counts) 215
gear for (slavecounts, mastercounts) 216
generate alternate mode (alternate/standard pins) 203
generate output rate (pulse) 200
generate steps on (step/direction) 201
goto 188
host read 280
host write 281
if/goto 189
if/then 185
move at (maxvelocity) for (displacement; trapezoidal) 208
move at (maxvelocity) to (position; trapezoidal) 206
move for (displacement; triangular) 208
move in (time) for (displacement; trapezoidal) 209
move in (time) to (position; trapezoidal) 207
move master at 238
move to (position; triangular) 205
move trap for (displacement; trapezoidal) 209
move trap to (position; trapezoidal) 206
new endposition (position or displacement) 210
offset position 192
offset slave (position) 216
on 189
pls (output) on/off 199
pls (output) using 198
pulse (output) for 198
reset 185
segmove <n> accdec...rate 241
segmove <table> accdec...disp 242
segmove <table> clear 241
segmove <table> slew 242
segmove <table> start relative 243
segmove <table> stop 242
set capture (registration input) 219
set capwin range (start, end) 219
set common bit 195
set common var 196
set feedback position 192
set loopperiod 191
set master source 193
set mode positioning 191
set mode tracking 192
set simulated feedback 192
set target position 192
set timeout 183
setout 197
slew begin 212

QuickMotion commands:
slew end 213
slew for (displacement) 213
start 187
statement 187
stop 182
stop table 234
table <n> addpair 227
table <n> addseries 228
table <n> clear 227
table <n> continue 231
table <n> copy 228
table <n> loadoffset 229
table <n> loadseries 229
table <n> precompute 231
table <n> start <imethod> <tscale>... 232
table <n> start <imethod> cam... 233
variable assignment (to expression) 184
wait capture (registration input) 220
wait common bit 196
wait for (transition) of (input) 200
wait for common var 196
wait for in position 210
wait master (counts) 216
wait outside (position range) 217
wait slave (counts) 216
wait until 186
wait within (position range) 217
zero (master/slave) counters 217
zero feedback position 184
zero following error 184
zero target position 184
QuickMotion programming:
gearing statements 215
I/O statements 197
operators 181
position capture and queue statements 219
program flow statements 187
simple motion statements 205
utility statements 182
QuickMotion variables:
_highBW 257
_inertia 257
_wn 257
_zeta 257
acc 254
accVector 277
accX 277
accY 277
accZ 277
activeBG_MSBs 272
activeCAM_row 251
activeFG_MSBs 272
aff 257
angle 277
angleStart 277
angleSweep 278
antibackup 264
axisnum 251
axisptr 273
axisY 278
axisZ 278
camming_invertend 259
camRequest 251
capArmed 270
capEdge 270
capGate 270
capGateState 271
capInput 271
capLimit 271
capLimitflag 271
capMod 271
capOffset 271
cappos 271
capposc 271
capStatus 271
capTriggered 271
capWait 271
capwaitBranch 271
capwinEnd 271
capwinStart 271
capwinType 271
cmode 254
ctr# 262
debugTable 236, 272, 292
debugTableRow 236, 272, 292
debugTableRows 236, 272, 292
debugTableX 236, 272, 292
debugTableY 236, 272, 292
dec 254
decVector 278
decX 278
decY 278
decZ 278
dim_factor_denom 273
dim_factor_num 273

QuickMotion variables:

din# 262
dins 263
dout# 262
douts 263
driveenable 263
dwSlaveID 273
eCAT_driveType 273
eCAT_manufID 274
eCAT_productCode 274
enabled 251
encoder_mode 254
encoderZ 259
encoderZ3 259
fault# 251, 292
faulted 251, 292
fpos 259
fposc 259
global_flag# 263
global_inputs 263
global_outputs 263
global_reg# 263
gratio 259
gtimebase 254
homing_speed1 255
homing_speed2 255
inpos 251
inpos_t 255
inpos_w 255
inposw 259
invel_t 255
invel_w 255
invertcmd 259
invertfeed 259
invertmaster 259
jerk_a 255
jerk_a_req 255
jerk_d 256
jerk_d_req 255
kd 258
kfilt 258
kgain 258
ki 258
kv 258
kvf 258
last_ALStatusCode 272
last_errorCode 272
last_errorRegister 272
last_errorType 272
lastOverall 272
loopperiod 272
looprate 272
magnitude 278
master_feedback 264
maxLoopTime 273
maxOverall 273
mcinv 265
mdelta# 265
minLoopTime 273
minOverall 273
mmc 266
move_master_counts 267
move_master_ramp 267
move_master_rate 268
move_master_rate_target 267
mpgai 268
mpgfi 268
mposc 266
mposc# 266
mppr 260
msource 271
neglim 260
newvel 256
nonvolatile 258
overflowFlag 273
overneg 251
overnegin 263
overpos 251
overposin 263
overtrq 251
pdead 258
perr 260
perrlimit 260
pff 258
poslim 260
ppg 258
ppr 260
pstate 251
qs_decel_time 274
QS2_CAP_WINEND_REL 288
QS2_CAP_WINOFFSET 289
QS2_CAP_WINSTART 288
QS2_Cmd 288
QS2_CMD_CNT 289
QS2_FILTER_MODE 289
QS2_Holding 288

QuickMotion variables:
QS2_HOLDING_CNT 289
QS2_HOME 289
QS2_LAST_CMD 289
QS2_MSB_STATE 289
QS2_OVERRIDE_CNT 289
QS2_Overrides 288
QS2_PARAM_CNT 289
QS2_Params 288
QS2_REG_STATUS 289
QS2_Status 288
QS2_TMP1 289
QS2_TMP2 289
QS2_TMP3 289
QS2_TMP4 289
QS2_VAR_MTD 289
QS2_VAR_MTN 289
QS2_VAR_NEW_ACC_FEEDFORWARD 289
QS2_VAR_NEW_ACCELERATION 288
QS2_VAR_NEW_DECELERATION 288
QS2_VAR_NEW_DIFFERENTIAL 288
QS2_VAR_NEW_FORCE_CUMULATIVE 288
QS2_VAR_NEW_FORCE_POSITION 288
QS2_VAR_NEW_HOLDING_MODE 288
QS2_VAR_NEW_INTEGRAL 288
QS2_VAR_NEW_MAX_SPEED 288
QS2_VAR_NEW_PROPORTIONAL 288
QS2_VAR_NEW_VEL_FEEDFORWARD 288
radius 278
RFID_address 275
RFID_bytesTotal 276
RFID_bytesTransferred 277
RFID_channel 274
RFID_control 276
RFID_controlActive 276
RFID_count 276
RFID_data_readh 275
RFID_data_readl 275
RFID_data_writeh 276
RFID_data_writel 275
RFID_error 275
RFID_index 275
RFID_lasttagIDh 277
RFID_lasttagIDl 277
RFID_state 274
RFID_status 276
RFID_tagIDh 277
RFID_tagIDl 277
RFID_totalChannels 274
running 264
runv 260
scanning 273
sdc 268
settling 260
sfmod 260
sfpos 260
sfposc 260
sign 260
smark 268
smarkfall 269
smarkrise 268
smod 268
smodc 268
spgai 268
spgfi 268
sphase 269
sppr 256
stepsout 260
stoprate 256
substep 260
theta 256
time 253, 256
timebase 256
tlim 256
tmax 256
tmc1 269
tmc2 269
tmodc 269
touchProbeStatus 253
tpos 261
tposc 261
tr 261
tracking_pstate 253
tracking_sign 270
tracking_status 270
tracking_tpos 270
tracking_tposc 270
trqc 261
tsc1 269
tsc1fall 269
tsc1rise 269
tsc2 269
tsc2fall 270
tsc2rise 269
units_ratio 261
uud 261

QuickMotion variables:
uun 261
vcmd 261
vectorY 278
vectorZ 278
vel 261
vel_accel_time 274
vel_decel_time 274
velVector 279
velX 279
velY 279
velZ 279
verr 262
vff 258
vmax 257
vmdelta 270
vmin 274
wControlWord 273
wStatus 273
zfpos 262
zpulse 253
ZPULSE_NEG 262
ZPULSE_POS 262
ztheta 257
ztpos 262
QuickScope traces:
set-up 329
windows 329
QuickScope:
A and B cursors 334
controller, connecting to 328
data collection rate 325, 331
features 325
immediate capture 327
invoking 325
multiple windows 338
overview 324
pdf creation 335
performance 325
reading data 332
status bar 327
tick rate 325, 331
toolbar 327
triggered capture 327
xls (Excel) creation 336
zoom feature 333
QuickStep 8
QuickView: 325, 336
multiple windows 338

- R -

registration inputs 158
repeat/until statement 92
Resource Manager (RM): 8, 39, 66, 153
specifications and max limits 11
resource types 65
resources 65
return statement 89

- S -

scalars 66
servo drives 151
servo motors 151
servo operating modes:
positioning 180
slewing 180
tracking 180
set statement 88
setbit statement 88
SFC (sequential function chart) 8, 53
sine 73
slew at (velocity, time) 212
slewing mode 180
splines 224
square root 73
stack overflow, cause 89
start task 54
statements: 86
assignment (numeric) 86
assignment (string) 87
begin 97
break 94
call, return 89
cancel 97
continue 94
delay 95
do 96
done 98
enable, disable (event) 96
for 93
goto 89, 90
goto next 90
if/then/else 90

statements: 86
monitor (QS2) 95
repeat/until 92
set 88
setbit, clrbit 88
start 98
stop 98
store 88
when 95
while 92
stepper drives 152
stepper motors 152
steps 56, 71, 86
store statement 88
string assignment statement 87
string function 76
strings 76, 87
symbolic names 65
symbols used in this manual 150
syntax 86

- T -

tables 66
tangent 73
tasks:
construct 54
definition 54
limit 71
parameters 54
text 76
toolbar 53
tracking mode 180
transitions 56, 95
trig functions 73

- V -

Variables, Pre-defined:
Capture Variables 270
Control Variables 254, 255, 256, 257
Diagnostic Variables 236, 272
Fault Variables 292
Feedback Variables 259, 260, 261, 262
IO and Register Variables 262, 263, 264
Quickstep Variables 288, 289
RFID Variables 274, 275, 276, 277
Setup Variables 273, 274
Status Variables 251, 252, 253
Tracking Variables 264, 265, 266, 267, 268, 269, 270
Tuning Variables 257, 258, 259
Vector Variables 277, 278, 279
variables: 66
non-volatile 65
resources 65
volatile 65
vector 66

- W -

when statement 95
while statement 92