

Subject: MotionWorks+ Template Program

Product: MP-940, MotionWorks+ v2.56 or later

Engineer: Michael J. Miller

Who should read this document?

This document illustrates a standardized template program to be used while programming the MP-940 with MotionWorks+. The template includes three main programs:

- 1) Supervisor
- 2) Manual
- 3) Automatic

In addition, eight subroutines are called from two of the main programs. The subroutines include:

- 1) Jog Forward
- 2) Jog Reverse
- 3) Homing
- 4) Indexing with Programmable Limit Switch (PLS)
- 5) Gearing
- 6) Camming
- 7) Torque
- 8) Latch with PLS

The program described in this document is meant to be used as a starting point for virtually any MP-940 MotionWorks+ application. Make use of the subroutines that are appropriate for the application and discard the subroutines that are not.

Table of Contents

Summary	4
Programs	4
Program Definition	4
Program Guidelines	5
1: Supervisor	6
Start-up	7
Fault Detection	8
Disable Handler	8
Fault Recovery	9
2: Manual	10
3: Automatic	11
Subroutines	12
02 Jog +	12
Increasing Jog Speed	13
03 Jog –	13
Increasing Jog Speed	14
04 Home	14
05 Move	15
Programmable Limit Switch (PLS)	15
Timer	16
06 Gear	17
Engaging/Disengaging	17
Running	18
07 Cam	19
Engaging	20
Disengaging	20
08 Torque	22
09 Latch	22
Latch Target Block	23
Configuration	23
System Parameters	24
System Properties	24
SGDH	25
MP940	26
COM1	27
COM2	28
Network	29
External Encoder	30
Data	30
Constants	31
Network	32
Tables	33

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085
 (800) YASKAWA - Fax (847) 887-7280

Variables	34
I / O	35
Local Input functionality	36
Local Output functionality	37
System Variables	38

Summary

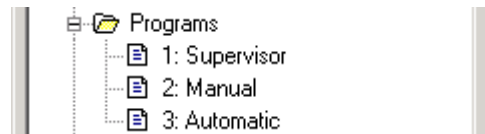
This appendix outlines the details of a template example program for an MP-940/SGDH system. In particular, it was designed with the Yaskawa demonstration (YEA Part # DEMO4700) unit in mind; however, it can be modified to suit virtually any application, and is commonly used as a starting point when programming the MP940 using MotionWorks+.

There are three main programs: Supervisor, Manual, & Automatic. The Supervisor program is the only of the three that is auto-starting. Once it starts and conditions are satisfied, it starts both Manual & Automatic program. In addition, there are seven subroutines: 02 Jog+, 03 Jog-, 04 Home, 05 Move, 06 Gear, 07 Cam, 08 Torque, and 09 Latch.

While the Manual & Automatic programs may be running, various conditions must be met for them to start a subroutine. All of these programs, as well as the configuration for the system will be discussed. This will be completed in the order that a program in MotionWorks+ is laid out, according to the Project Explorer window.

Programs

The Programs folder contains the following:
Supervisor, Manual, and Automatic programs.



Program Definition

Auto start is also a possibility for each program. This template program has one program that is auto starting (Supervisor). Once that program has started and various conditions have been satisfied, it will start the other programs. If the supervisor detects a fault, error, or other event it will stop the other programs. This programming methodology creates a solid infrastructure to build from so that each individual program does not have to monitor for errors, there is one program the does that and coordinates appropriately.

Program Guidelines

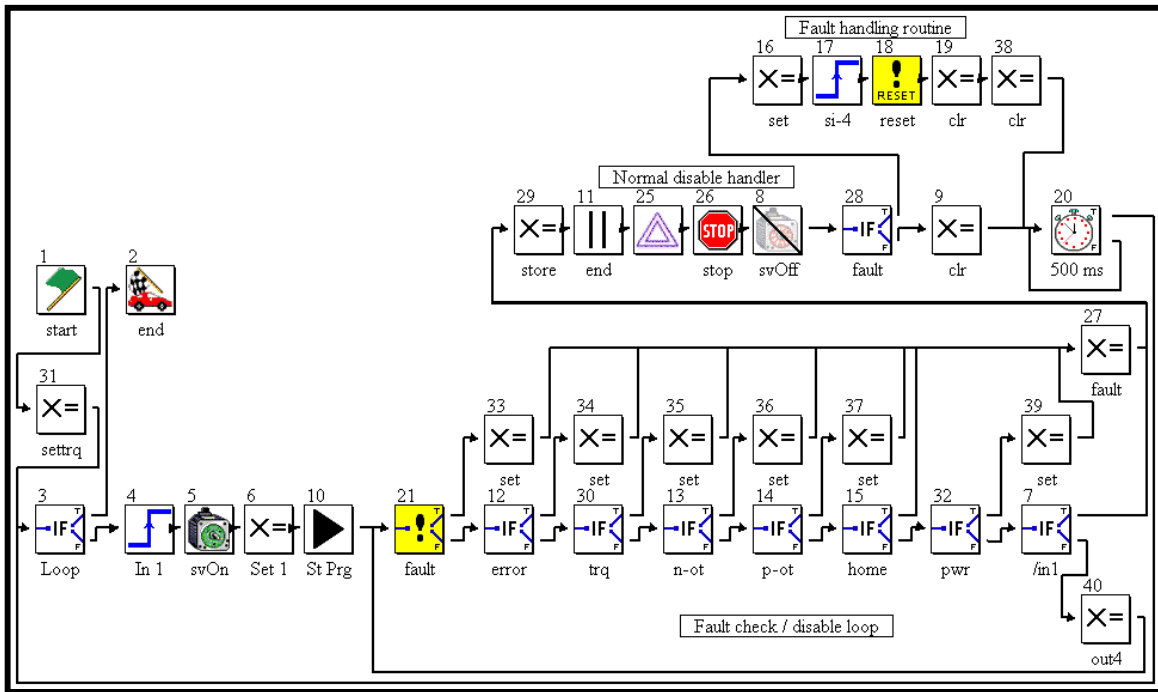
As a guideline it is recommended that each “program” contain a maximum of 64 total blocks including all subroutine blocks called from within the program (not including start and stop blocks). In this example, the Supervisor program utilizes 35 blocks; the Manual program utilizes 43 blocks; the Automatic program utilizes 58 blocks. In addition, it is also recommended that execution of motion blocks only be active in one program at a time. For this reason, the Manual and Automatic routines are interlocked such that they can only operate exclusively.

Motion blocks include the following:

- MOVE AXIS
- JOG
- STOP
- HOME
- CAM
- CHANGE DYNAMICS
- DEFINE POSITION
- GEAR
- LATCH TARGET
- SCALE CAM
- SERVO ENABLE
- TORQUE
- SLAVE OFFSET

The Supervisor routine utilizes the SERVO ON, STOP, and CHANGE DYNAMICS block, but is closely monitoring the other programs to ensure that there is no overlapping. Some precautions must still be followed with the use of Motion blocks.

1: Supervisor

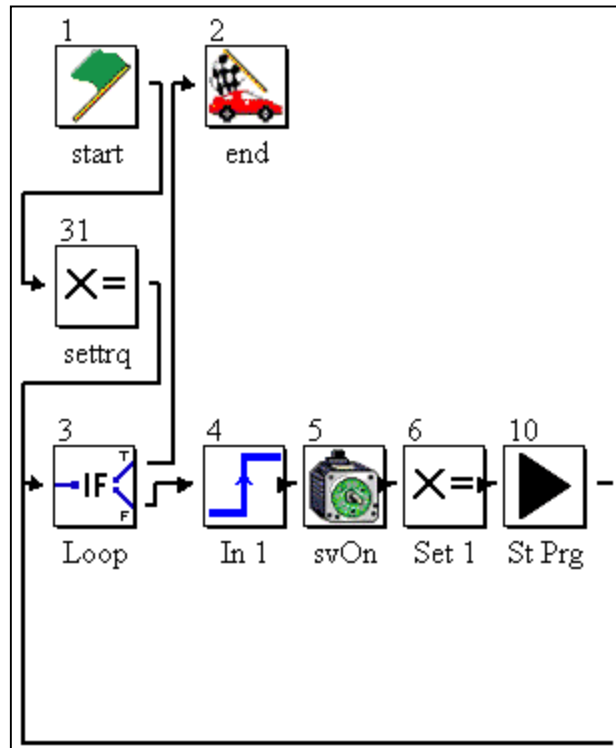


The Supervisor can be separated into four distinct sections: Start-up, Fault Detection, Disable Handler, and Fault Recovery.

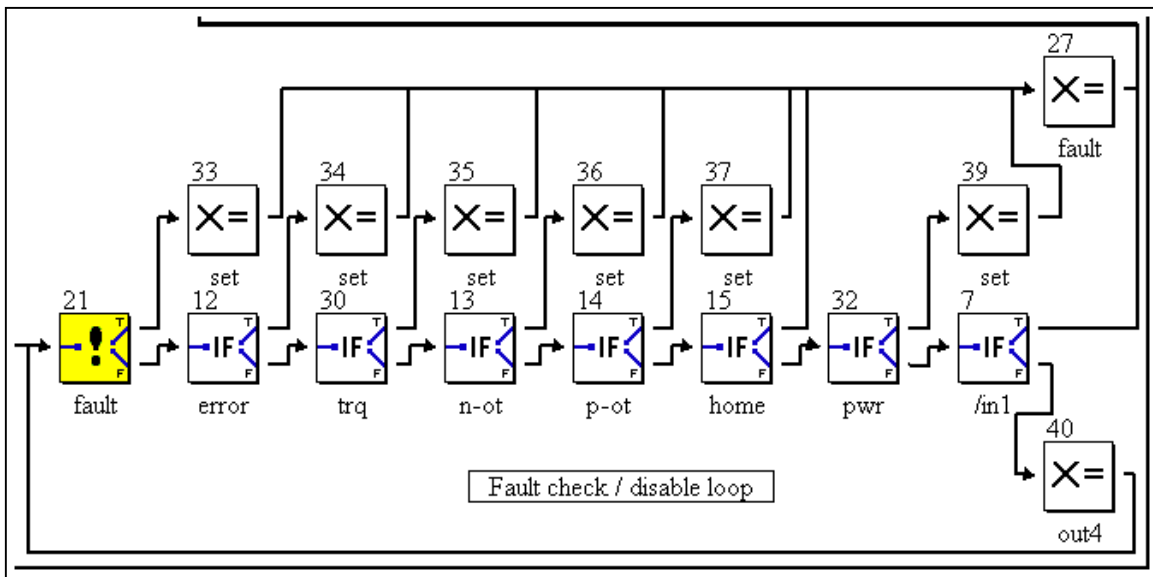
Start-up

The start-up section encompasses block 3-6, 10, and 31. It is responsible for startup. Block 31 is executed only once (upon power up) and can be useful for setting user variables, or outputs that need to be reinitialized at power up, but may be variables themselves. Block 3 has the condition "FALSE" in it, provides a point to loop back the end of the flow chart, and ensures that all blocks have connections. The remaining blocks regulate a normal start up.

Block 4 must see the rising edge of Local_Input1. This assures that input was activated to enable the system, rather than just left on all of time. Block 5 enables the servo. Local_Output1 and (user variable) [SystemOk] is set in Block 6. The [System Ok] variable is used by the two other main programs as a signal that it is Ok to execute. The last block (10) starts the other main programs (Manual and Automatic).

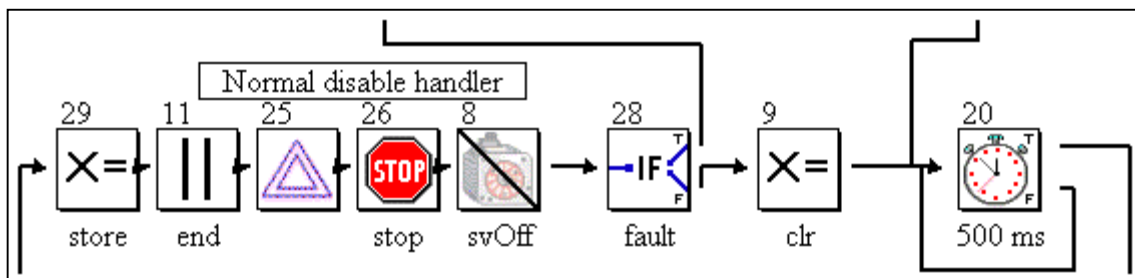


Fault Detection



After the start-up section has been successfully completed, the next section and the section that the program will execute most is the Fault Detection / Disable loop. All of the If EVENT (including a special IF_FAULT) blocks are designed to detect various faults and errors in the system. When an error is detected, the next block executed is a SET VARIABLE block that sets an internal flag to trap the type of error that occurred and may also capture some pertinent data. Lastly, SET VARIABLE block (27) is executed, which sets an internal fault flag and clears the SystemOk flag. Then, execution continues to disable handler. If the user simply turns off of the enable input (Local_Input1), execution continues to disable handler. The last block that deserves discussion is SET VARIABLE block (40); this block handles the homed/homing output, discussed later in the homing subroutine. Additionally it takes care of the modality, or which mode (manual or automatic) the machine is in, and sets the appropriate outputs.

Disable Handler



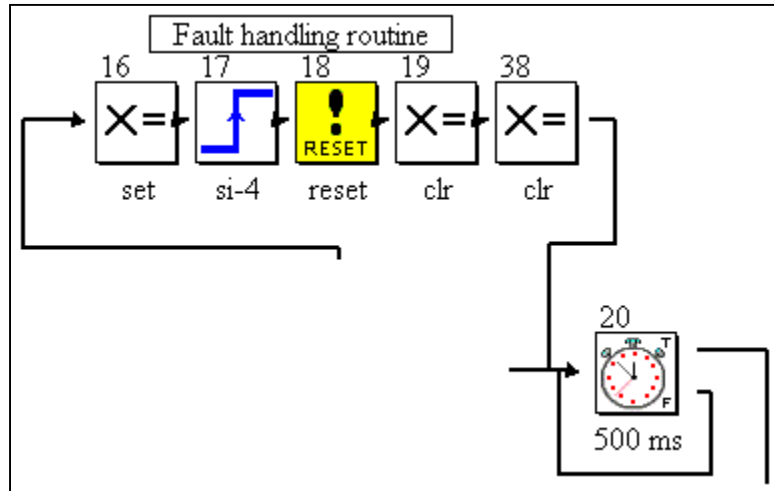
When a fault or normal disable is detected, this section of code is executed. The first block (29) stores the commanded and actual position (these may be useful in a recovery routine). Next, the other main programs are halted; the servo is commanded to go to zero speed (25), then stop (26), and lastly the SERVO OFF block (8) is executed.

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085
(800) YASKAWA - Fax (847) 887-7280

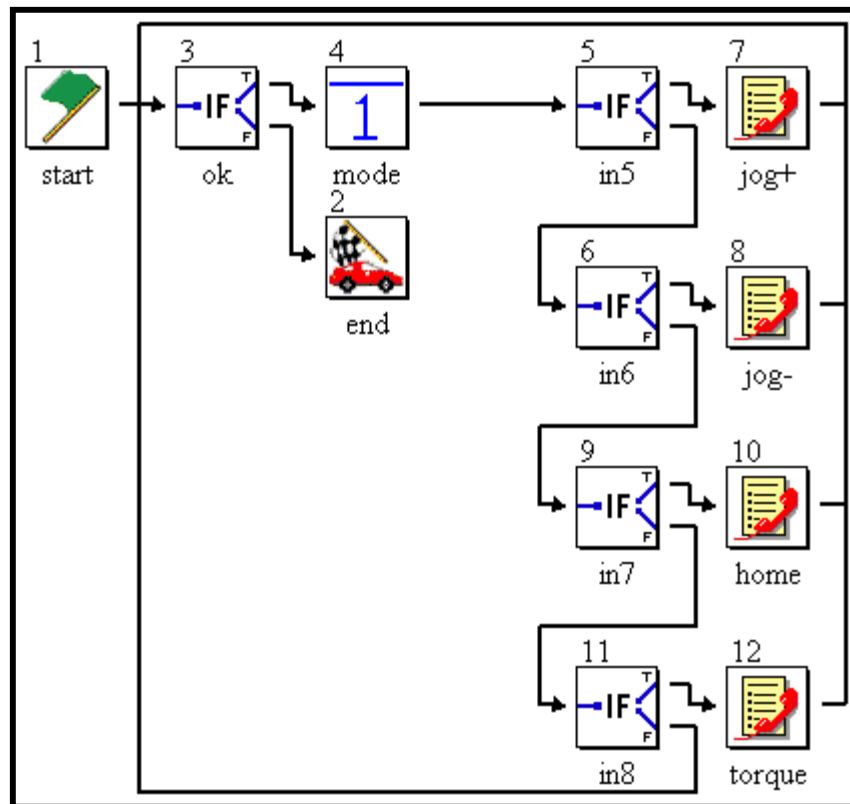
If a fault has occurred, block (28) will direct execution to the fault recovery section. Otherwise block (9) is executed. Block (9) clears all of the outputs that may have inadvertently been left on. The Disable handler and the fault recovery routine both make use of Block (20). It is simply a timer that ensures everything has settled down before attempting a restart. After block (20), execution continues back to block (3).

Fault Recovery

Once it has been determined that a fault occurred and the appropriate blocks have executed, the program ends up in the fault recovery section. Block (16) sets an output to indicate that a fault has occurred (in the case of the demo box, it actually sets all eight outputs). Input block (17) waits to see the rising edge transition of SGDH input SI-4 (coincidentally, the Servo Alarm Reset input when the ServoPack is used alone). Reset fault block (18) is a special block that will reset any ServoPack alarm that does not require a power cycle to reset. Lastly, Set Variable blocks 19 and 38 and clear the alarm output and all of the internal error bits. Block (20) was discussed above in the Disable Handler section.



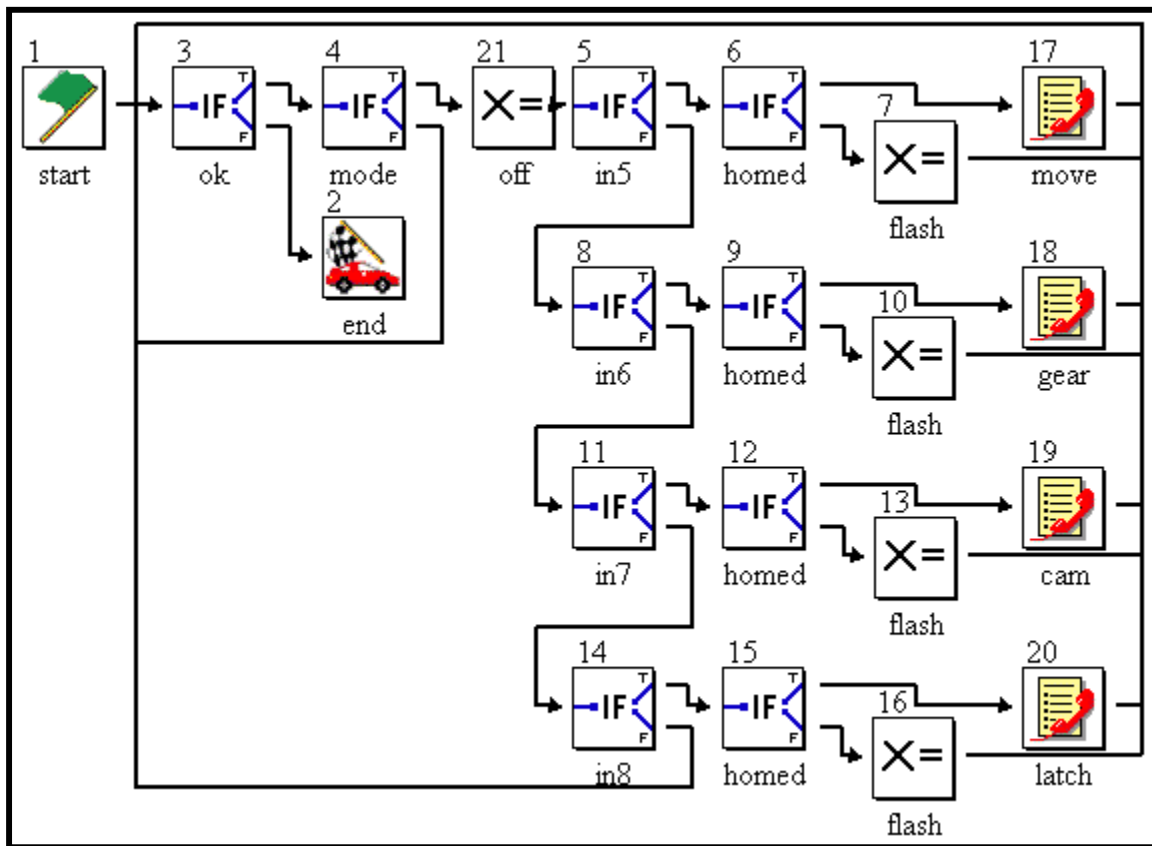
2: Manual



Functions typically performed while in a “manual” mode are included in the manual main program. While this program is executing, it is waiting for the user to activate an input to select a subroutine program. As long as no input is selected, the program scans the blocks in order of 3, 4, 5, 6, 9, and 11 then back to 3.

Two key features in this program are important to mention. First, if event block (3) detects if (user variable), [SystemOk] is true. As long as this condition is true the program will stay running (recall SystemOk is controlled by the supervisor program). Second, input block (4) ensures that (user variable) [ModeManual] has been selected. This is a critical interlock that guarantees that more than one main program is not attempting motion at the same time. This can be especially unsettling if the auto program is running the servo in camming mode and the manual program is attempting to jog.

3: Automatic



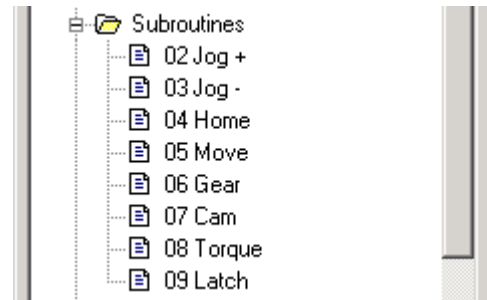
Functions typically performed while in an “automatic” mode are included in the automatic main program. While this program is executing, it is waiting for the user to activate an input to select a subroutine program. As long as no input is selected, and not in manual mode, the program scans the blocks in order of 3, 4, 21, 8, 11, 21 then back to 3.

Two key features in this program are important to mention. First, if event block (3) detects if (user variable), [SystemOk] is true. As long as this condition is true, the program will stay running (recall SystemOk is controlled by the supervisor program). Second, if event block (4), ensure that (user variable) [!ModeManual] has been selected (in other words if the machine is not in manual mode, it is in automatic mode). This is a critical interlock that guarantees that more than one main program is not attempting motion at the same time. This can be especially unsettling if the auto program is running the servo in camming mode and the manual program is attempting to jog.

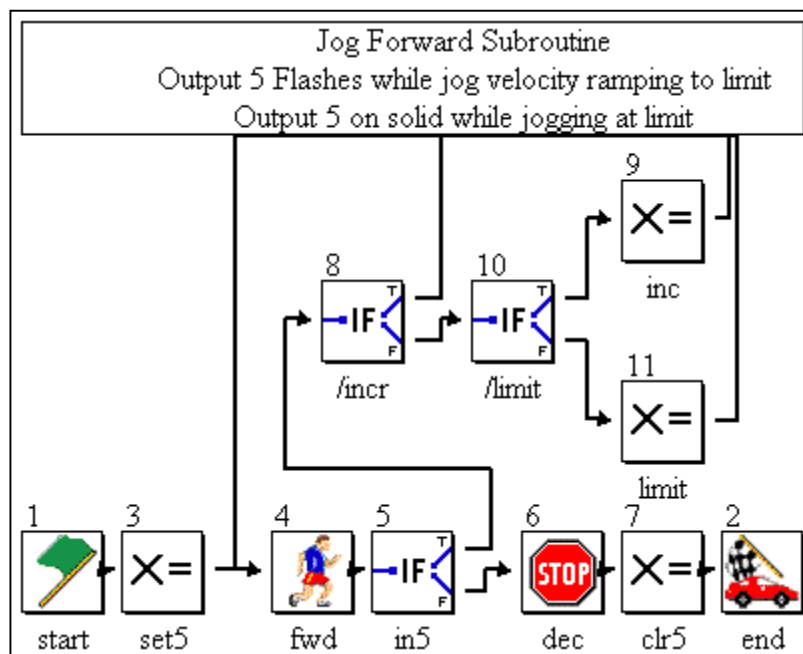
Automatic mode operation additionally requires that the system be homed prior to activating any of the automatic subroutines. If a user puts the machine in automatic mode, attempts to execute an automatic subroutine, and it has not been homed, the corresponding output will flash rather than its normal operation as described in the subroutines below.

Subroutines

The Subroutines folder contains the following:



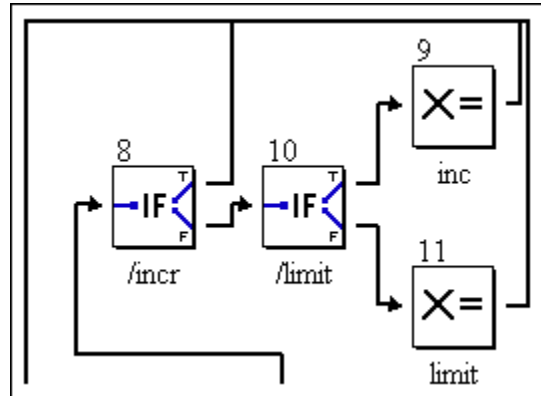
Jog + Subroutine



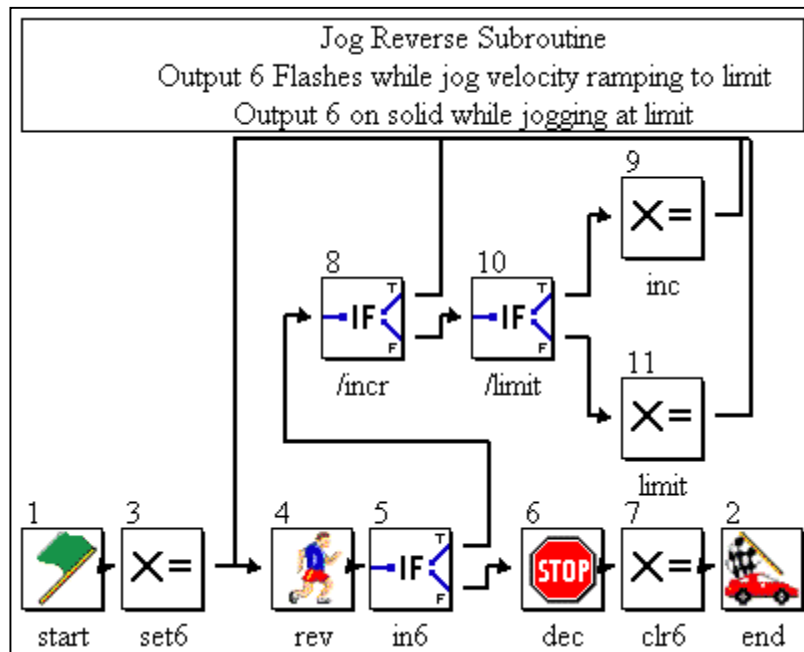
The jog + subroutine jogs the servo in the forward direction while the jog forward request (Local_Input2) is activated. When the user deactivates the jog forward request, the servo decelerates to a stop and execution is returned to the main program. LocalOutput2 is activated while the servo is in motion jogging at the limit (discussed below). Otherwise, the output flashes.

Increasing Jog Speed

Blocks 8 – 11 implement a jog speed increase while the jog forward request is activated up to a user defined speed limit. Block (8), IF EVENT, allows the incremental velocity increase to occur every (user variable [VelJogIncrementTime]) time period. Block (10), IF EVENT, verifies that the velocity has not reached (user variable [VelJogLimit]) the limit. If the limit has not been reached, set VARIABLE block (9) increments the velocity (user variable [VelJogIncrement]). Otherwise, set VARIABLE block (10) maintains the jog velocity at the limit.



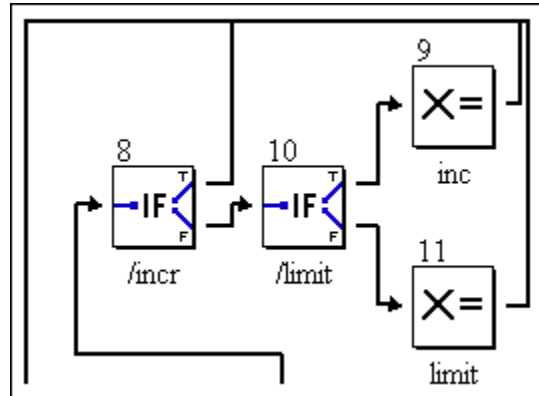
Jog –



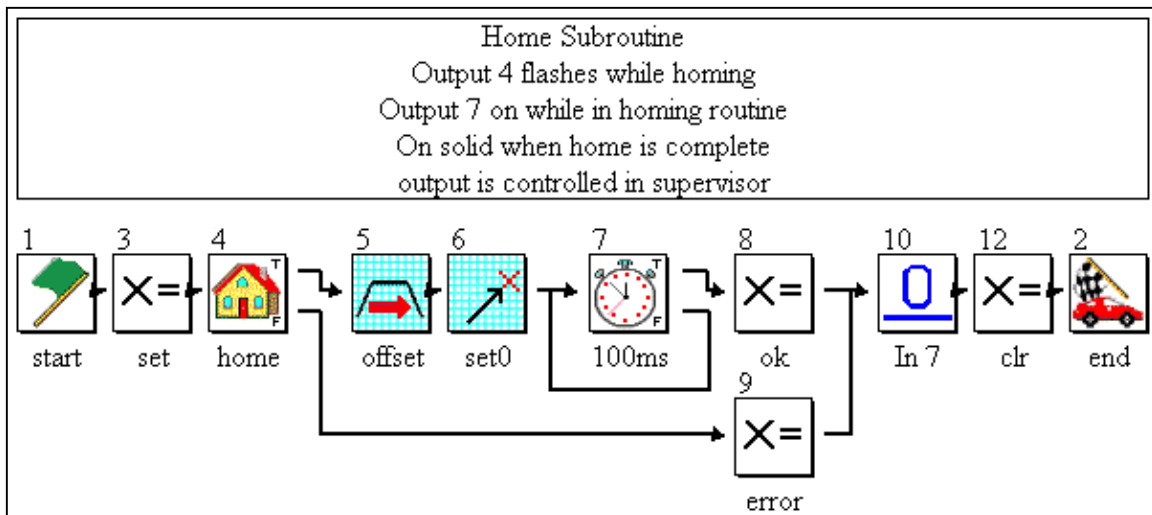
The jog – subroutine jogs the servo in the reverse direction while the jog reverse request (Local_Input3) is activated. When the user deactivates the jog reverse request, the servo decelerates to a stop and execution is returned to the main program. LocalOutput3 is activated while the servo is in motion jogging at the limit (discussed below). Otherwise, the output flashes.

Increasing Jog Speed

Blocks 8 – 11 implement a jog speed increase while the jog reverse request is activated up to a user defined speed limit. Block (8), IF EVENT, allows the incremental velocity increase to occur every (user variable [VelJogIncrementTime]) time period. Block (10), IF EVENT, verifies that the velocity has not reached (user variable [VelJogLimit]) the limit. If the limit has not been reached, set VARIABLE block (9) increments the velocity (user variable [VelJogIncrement]). Otherwise, set VARIABLE block (10) maintains the jog velocity at the limit.

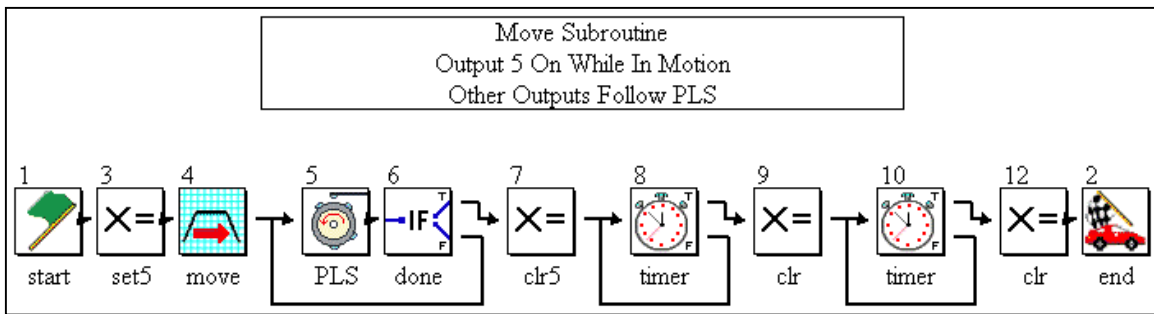


Home



The home subroutines take advantage of the HOME block (4) which has built in functionality for homing to and through a deceleration switch then to the C (zero) pulse of the encoder. After that is complete, an offset move is completed. Then, the SET POSITION block (6) re-defines the position to 0.0 (this could be a user variable). In this program, that same block also sets the external position to 0.0 (again that could be a user variable also). Blocks 3 and 8 take care of activation and deactivation of some internal user variables [Homed] and [Homing]. Block 8 sets a user error variable [ErrorHoming] which is detected elsewhere. Lastly, INPUT block (10) ensures that the user has deactivated the homing request input (Local_Input4). If this block was not in place and the user left the home request on, the machine would home over and over.

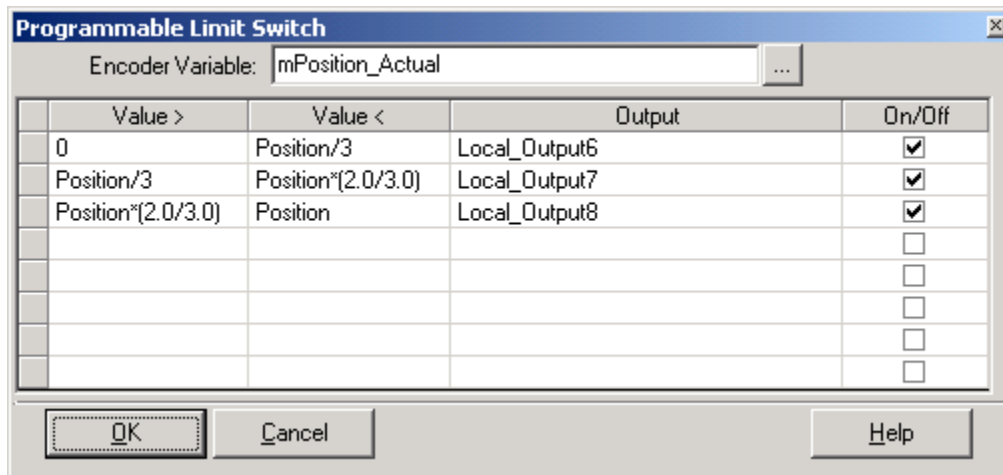
Move Subroutine



Move subroutine is a simple relative move, which incorporates a programmable limit switch (PLS) block and a couple of timers. This subroutine differs slightly from the others in that there is no event that locks it in the subroutine. Once the move is complete and the timers have timed out, the subroutine ends and execution is returned to the main calling program. If the input (input 5) that calls the subroutine is left on, the subroutine will be executed again (and again).

The set VARIABLE block (3) activates output 5; Corresponding block (7) deactivates the output making the output correspond to an “in motion” indicator. The next block, TIMER (8), delays by a user variable [MoveDelay]. Set VARIABLE block (9) deactivates any outputs that may be left on from the PLS. Execution continues to another TIMER block, with the same user variable as the earlier TIMER block.

Programmable Limit Switch (PLS)



Because the controlled axis is configured in rotary mode, the system variable mPosition_Actual rolls over automatically at the system variable sMachineCycle_Main. Therefore, mPosition_Actual can be used as the “Encoder Variable” in the PLS block (5). For systems where rotary mode is not applicable, mPosition_Actual (in a user variable, such as [PositionCapture]) can be captured in the SET VARIABLE block (5) prior to motion. Then, to make the PLS easy to setup (i.e. elements are in relation to move length), the “Encoder Variable” could be configured as a calculation; mPosition_Actual – PositionCapture.

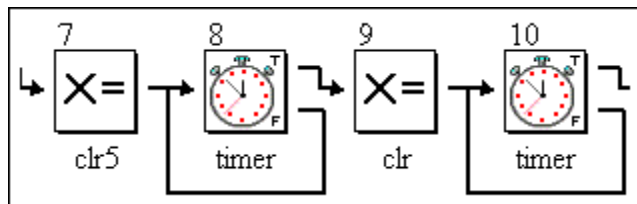
Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085
(800) YASKAWA - Fax (847) 887-7280

The PLS block implemented in this project is executed in this subroutine and therefore relies on the MOVE Block (4) Wait for completion box to be un-checked. It is used in conjunction with the IF EVENT Block (6) that checks for the move to be complete (system variable mPosition_Complete), the PLS is updated while in motion.

Entries for (Value >) and (< Value) can be implemented as fixed numbers, user variables, system variables, and any combination thereof. In addition, the entries can be calculations, especially useful for applications where PLS outputs also depend on speed of machine. Be careful with calculation syntax, as 'C' syntax is followed. Lastly, when implementing values that go through (past) zero, it is best to split them up in to two segments [(Value>) to 0] and [0 to (<Value)] that set internal User bits. Then, in a SET VARIABLE block, use those internal user bits ORed together to set the correct output.

Timer

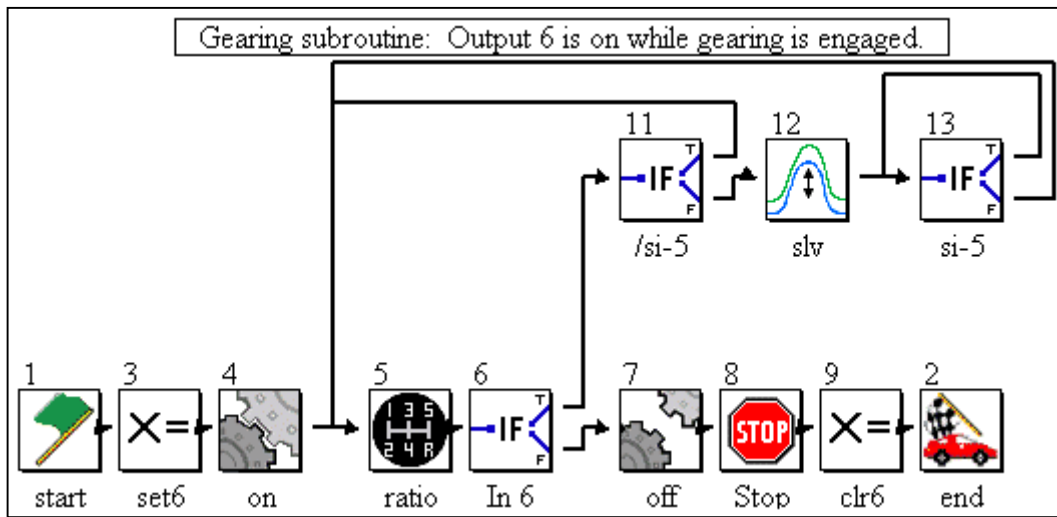
The last few blocks (7-10) are comprised of SET VARIABLE and TIMER blocks. Some special attention should be brought up regarding the Timer block. While the timer block has two out ports, the false port should always be looped back to the in port of the block. Time only



accumulates while the block is being executed, thus the need for the loop back. If items are connected between the false port and the return to the true port, it can cause the timer to take additional time to complete. For example, if a block is inserted between the false port and the return to the true port and it takes one scan to execute that block, the amount of time will be doubled. One last thing to remember about the Timer block is that the time value in it is retentive. In other words, if the timer starts timing and for some reason gets interrupted, (program is halted, etc) upon re-entering said timer block, it will finish timing.

There is an alternative way to construct a timer. Use the system variable mTime, which counts mSec. Capture the value of mTime in a SET VARIABLE block. In the next block, an IF EVENT block, compare if mTime > CaptureTime + DesiredTime. While that condition is FALSE, the program can monitor other events and return to the in port of the IF EVENT block. Be careful with this method, as the timed value can vary if there is a lot of logic between the false port and the return to the in port.

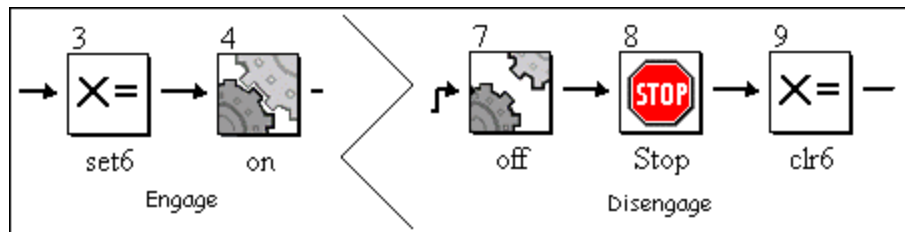
Gear Subroutine



The Gear subroutine is fairly straightforward. For discussion purposes, it is broken down into two sections. The two sections include: Engaging/Disengaging and Running.

Engaging/Disengaging

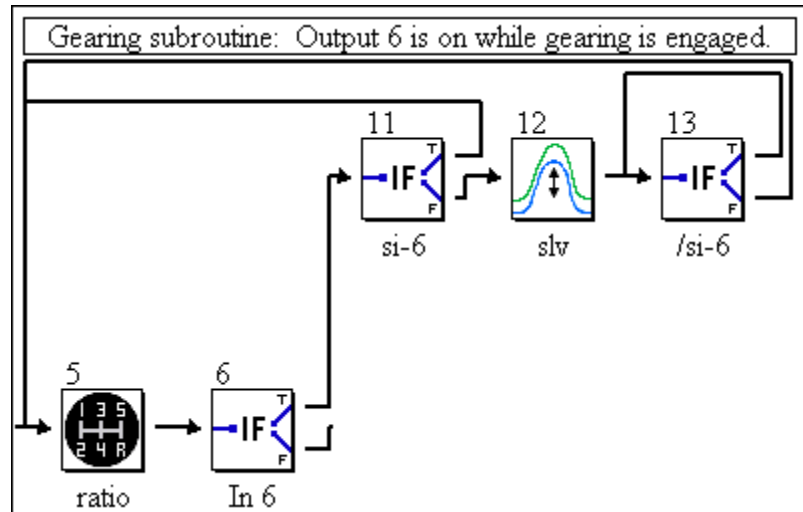
Blocks 3 – 4 & 7 – 9 comprise engaging and disengaging of gearing. Blocks 3 and 9 take care of activating / deactivating



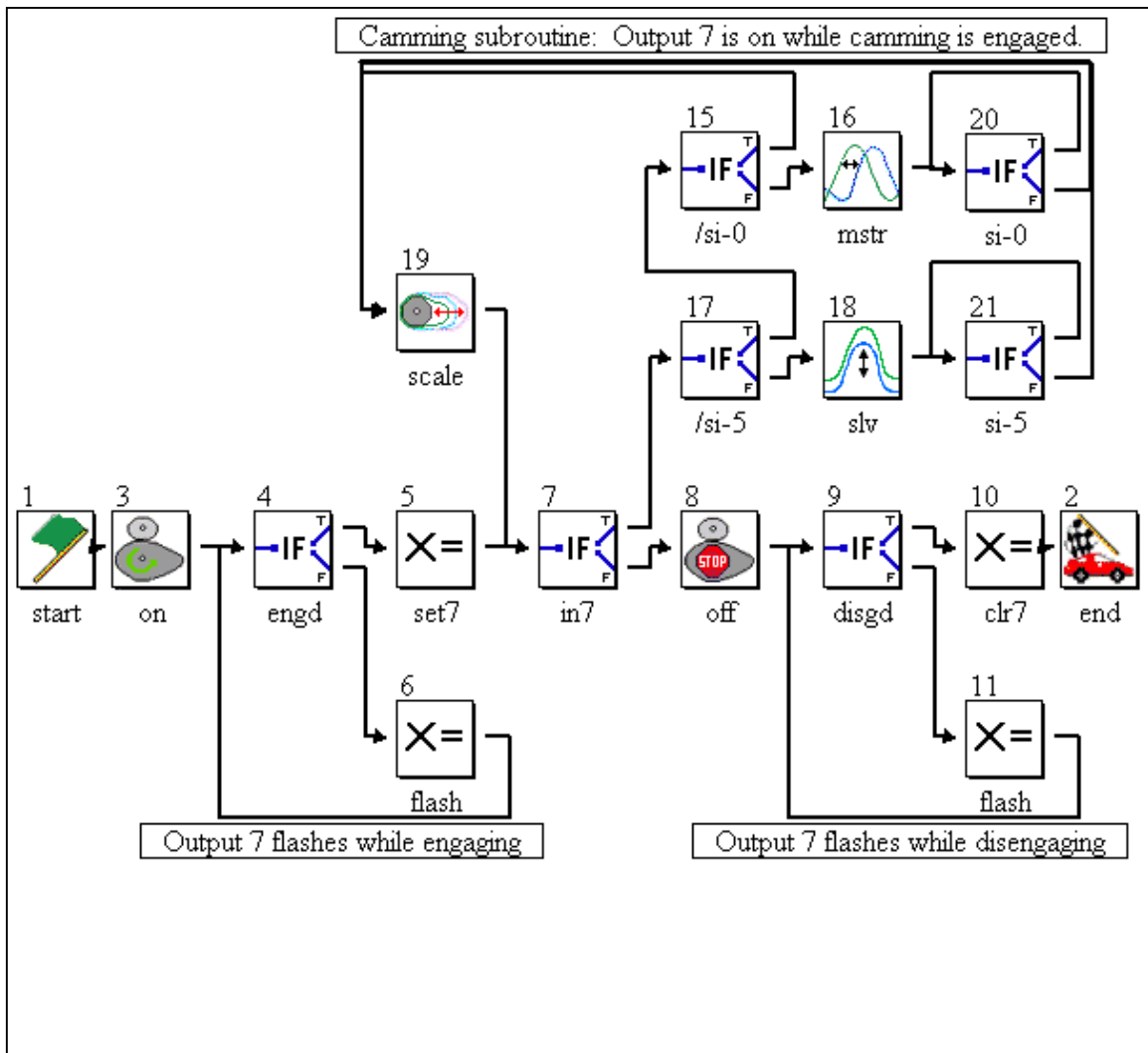
Output 6 while entering / leaving gearing subroutine. The key element to remember is to include a STOP block. If the STOP block is not included and the slave is disengaged while the master is motion, the slave will continue to rotate to the last known calculated speed of the master axis. Additionally, the STOP block will switch the axis to position mode, providing a controlled deceleration to zero speed and will maintain position once stopped.

Running

The rest of the blocks are executed while the system is engaged in gearing (or running). The user is able to adjust the gearing ratio by modifying the variables (user variables [GearMaster] divided by [GearSlave]) in the Gear Ratio block (5). The gear ratio is a fraction of two integer numbers. Be careful not to use floating-point variables for these. The values will be truncated and lost motion may occur. In addition, shifting of the Slave position is possible by activating input SI-6. Amount of offset is relative (in user units, based on user variable [GearSlaveOffsetPosition]) to the current position of the respective axis. The offset can be accomplished over a given amount of time or distance (in user units, based on user variable [GearSlaveOffsetDuration]). System variable sSlaveOffset_Mode determines if the duration is time or position based. (sSlaveOffset_Mode = 0, time based offset; sSlaveOffset_Mode = 1, position based offset).



Cam Subroutine

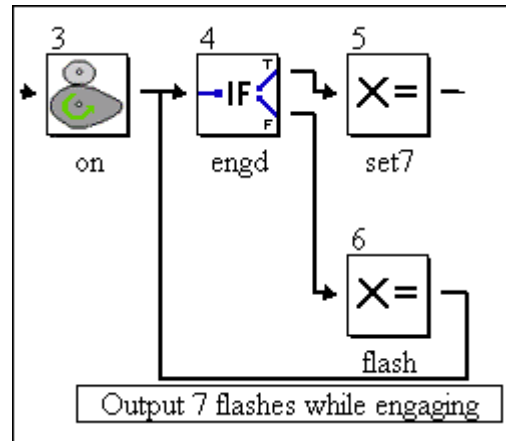


The three sections of the CAM Subroutine include: Engaging, Disengaging, and Running. (Note: Cam table generation, using Cam Tool is beyond the scope of this document)

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085
(800) YASKAWA - Fax (847) 887-7280

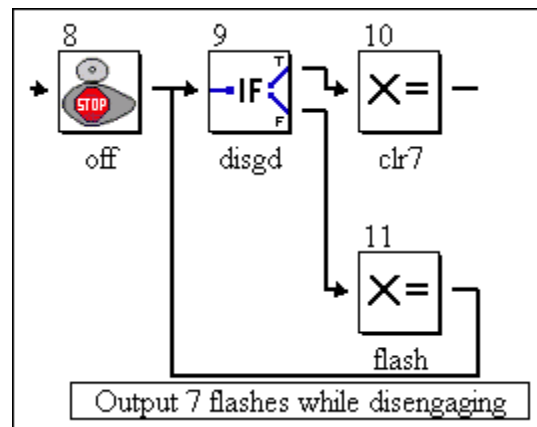
Engaging

The CAM engage consists of the blocks 3 – 5. The Cam engages at a particular Master position based on the value entered in block three, for this program 0 is the position used. Block 3 also sets the system variable mState_Camming = 1, this indicates that the system is waiting to engage. While waiting to engage (mState_Camming = 1) output 7 flashes. Once engaged (mState_Camming = 2) output 7 is turned on solid.



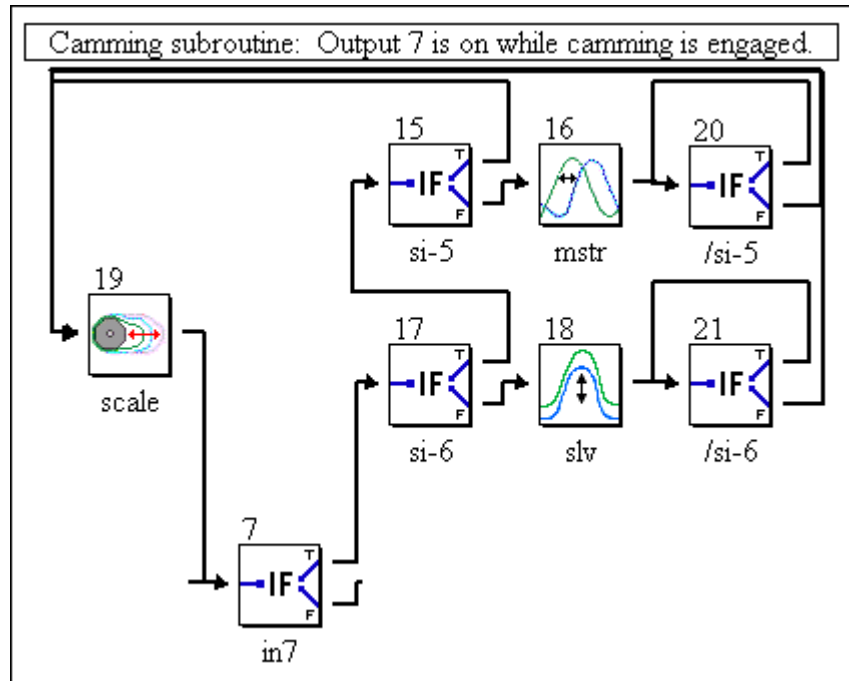
Disengaging

Disengagement begins after input 7 has been deactivated and consists of block 8 – 11. This works quite similarly to engaging, in Block 8 the disengage position is specified. Again, for this program 0 is the position used. Block 8 sets mState_Camming = 4, while disengaging output 7 again flashes. Once the disengage position has passed (mState_Camming = 0), output 7 is deactivated and the subroutine is exited.



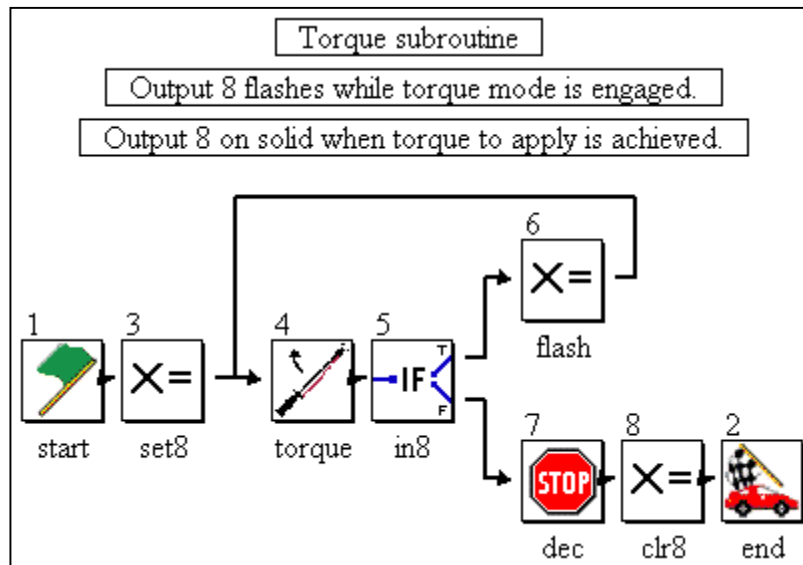
Running

The rest of the blocks are executed while the system is engaged in camming (or running). The user is able to scale the cam by modifying the variable (user variable [CamScale]) in the CAM SCALE block (19). The scale factor is a percentage of the original cam, where 100.00% represents the original cam size. A value larger than 100% equates to an expanded cam, less equals a contract cam. In addition, shifting of the Master or Slave position is possible by activating inputs SI-5 or SI-6, respectively.



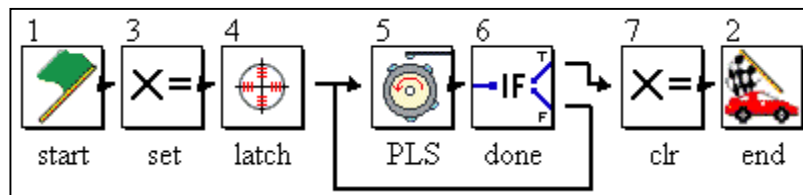
Amount of offset is relative (in user units, based on user variable [CamMasterShiftPosition] or [CamSlaveOffsetPosition]) to the current position of the respective axis. The offset can be accomplished over a given amount of time or distance (in user units, based on user variable [CamMasterShiftDuration] or [CamSlaveOffsetDuration]). System variable sSlaveOffset_Mode determines if the duration is time or position based. (sSlaveOffset_Mode = 0, time based offset; sSlaveOffset_Mode = 1, position based offset)

Torque Subroutine



The motor applies a variable amount of torque (based on a user variable [TorqueToApply]). It also limits the velocity of the motor (based on a user variable [Vel]). As long as input 8 is activated this is continued. Output 8 flashes while applying torque and goes on solid when motor is operating below the velocity limit. Once input 8 is deactivated the servo is decelerated to stop, output 8 is deactivated and the subroutine is exited.

Latch Target Subroutine

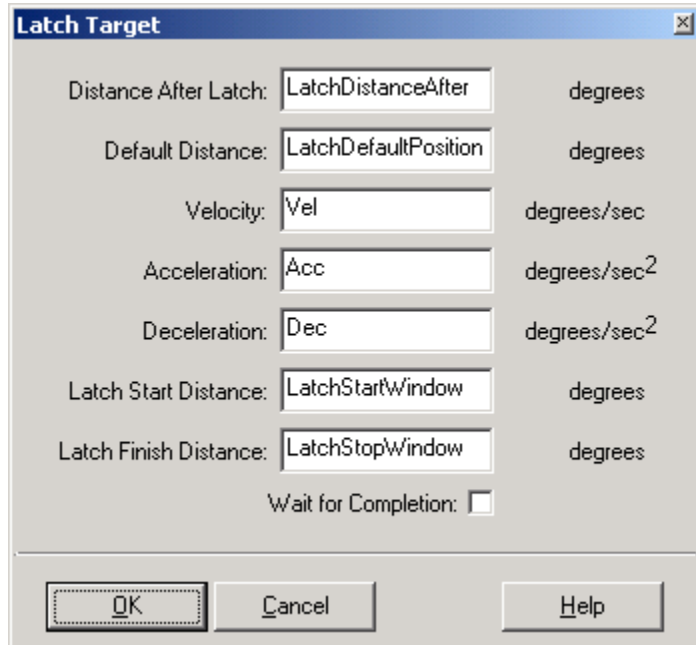


The latch subroutine works very similarly to the move subroutine. The main difference (besides the lack of timers in this subroutine) is the move block has been replaced with a LATCH TARGET block (4). Additionally, the values in PLS block (5) has modified to be more suitable for the Latch routine. It has been configured to activate output during the various states of latching.

LATCH TARGET Block

The LATCH TARGET block is a pre-configured block that allows the user to make use of the high speed (captures position in 30 microseconds or less) without knowing the ins and outs arming/disarming/windowing the latch signal. The block is much like a MOVE block. In fact, if a latch is not received, it will work identically to a MOVE AXIS block (Default Distance in LATCH TARGET block = Position in MOVE block). However, if a Latch signal is received, the servo will move the Distance After Latch from the point where the latch was detected.

Occasionally false latches can be a problem and cause the servo to move an incorrect distance. The Latch Start and Finish Distance parameters are useful in removing unwanted latch signals. They are used to set-up a window in which the latch is expected.



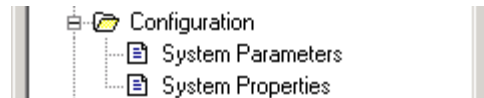
The screenshot shows a dialog box titled "Latch Target" with the following parameters:

Distance After Latch:	LatchDistanceAfter	degrees
Default Distance:	LatchDefaultPosition	degrees
Velocity:	Vel	degrees/sec
Acceleration:	Acc	degrees/sec ²
Deceleration:	Dec	degrees/sec ²
Latch Start Distance:	LatchStartWindow	degrees
Latch Finish Distance:	LatchStopWindow	degrees
Wait for Completion:	<input type="checkbox"/>	

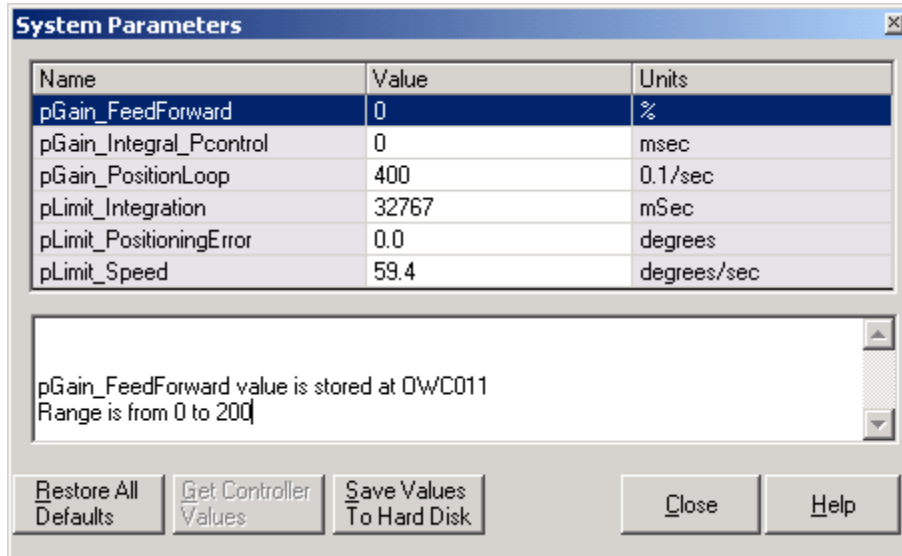
Buttons: OK, Cancel, Help

Configuration

The Configuration folder contains: System Parameters and System Properties configuration.

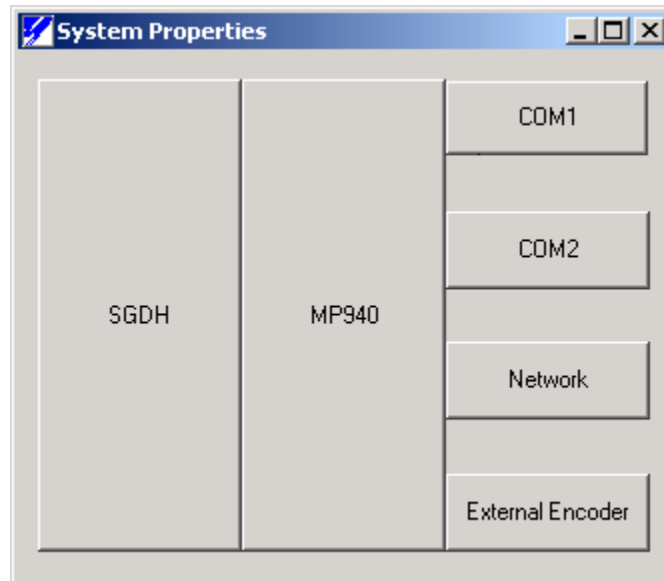


System Parameters



System Parameters have all been left at factory default. The user may need to modify some of these parameters when tuning the system.

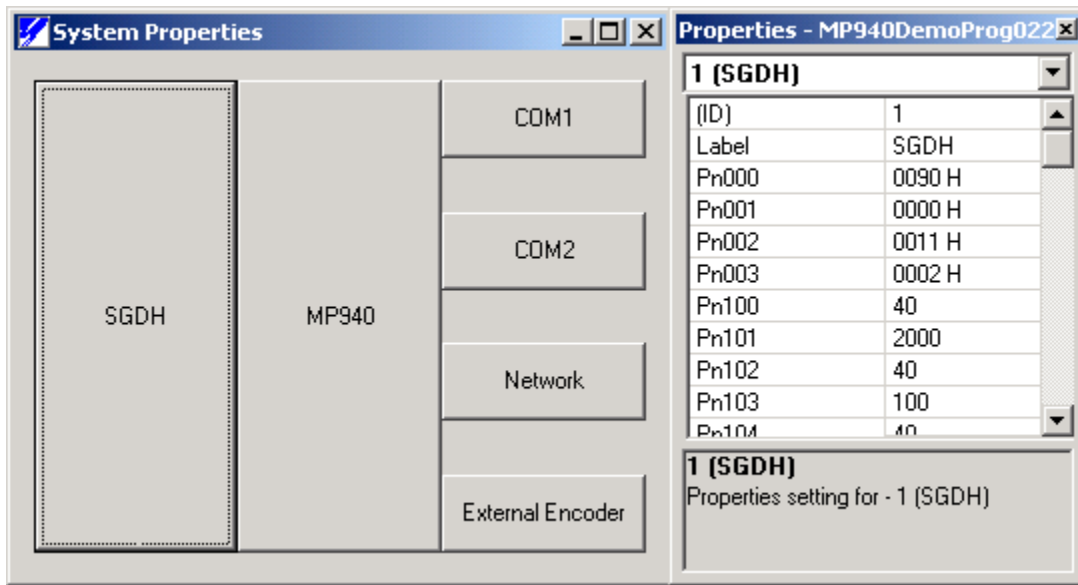
System Properties



The System Properties consist of: SGDH, MP-940, COM1, COM2, Network, and External Encoder configuration.

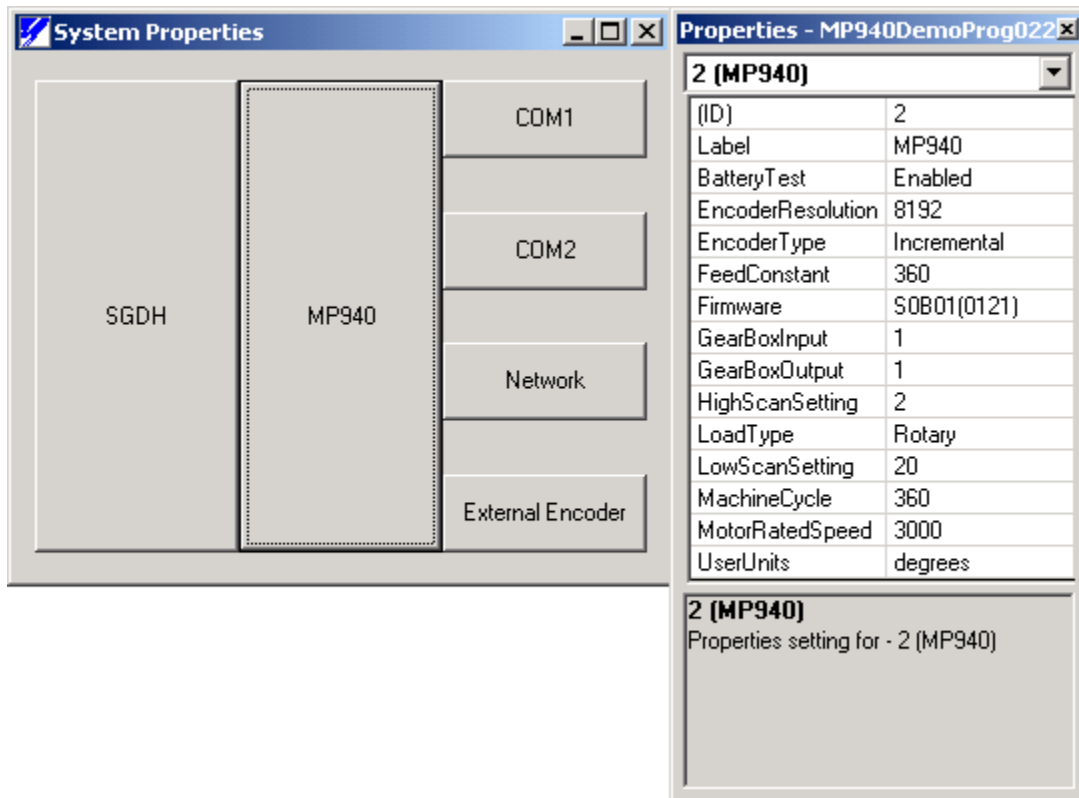
Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085
(800) YASKAWA - Fax (847) 887-7280

SGDH



All SGDH parameters are left at factory default for this example. The user may need to modify some user parameters when tuning the system.

MP940



Only five parameters were modified in the External Encoder properties to correspond with the demo unit. They are as follows:

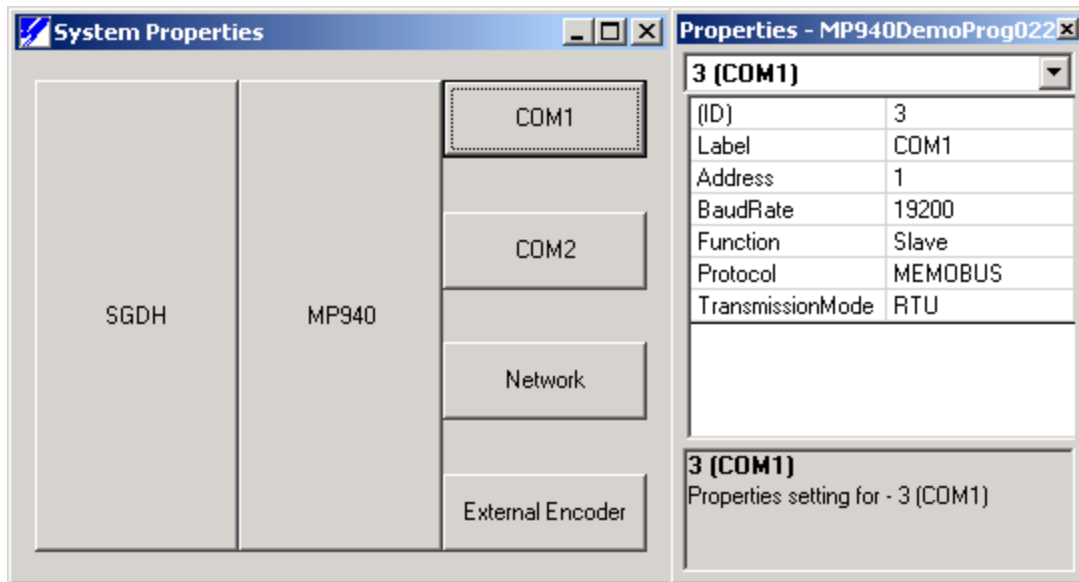
- BatteryTest: Enabled
- FeedConstant: 360 degrees
- Firmware: (Uploaded from Controller)
- HighScanSetting: 2 milliseconds
- LoadType: Rotary
- MachineCycle: 360 degrees
- UserUnits: degrees

These were modified to convert the SGMAH-01BAF41 of the demo to a rotary mode so that 8192 pulses, which is one revolution, equates to 360 degrees and so that it rolls over automatically at

Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085
(800) YASKAWA - Fax (847) 887-7280

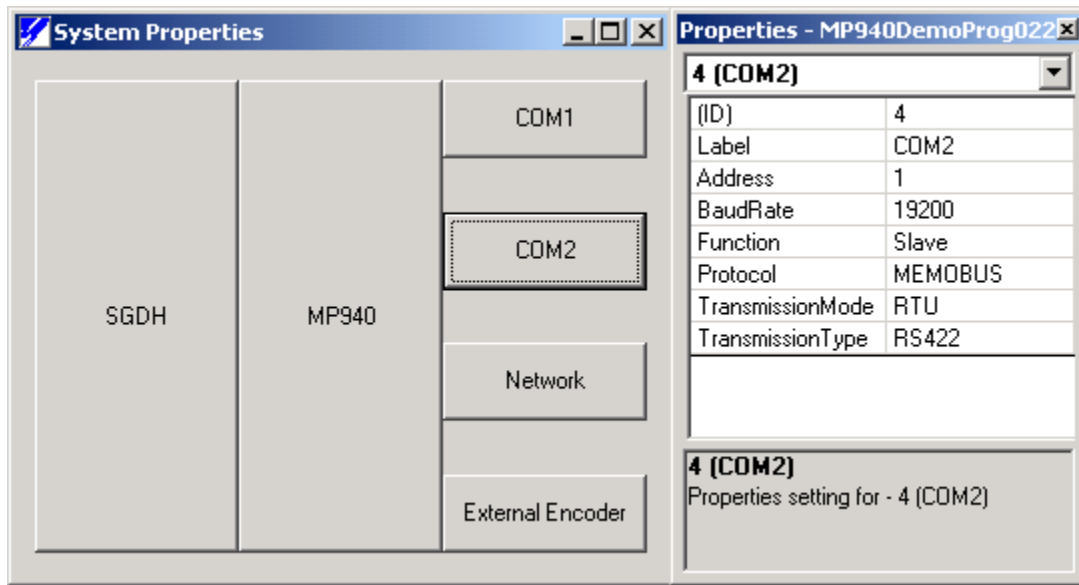
360 degrees. The Scan time was increased due to the large program size. This will eliminate nuisance over scan errors (A.E2 on SGDH Display). The battery test can be left at disabled if system does not include a battery backup. However, user should be concerned about data and program maintainability during extended power off time. The use of Flash memory is out of the scope of this document.

COM1



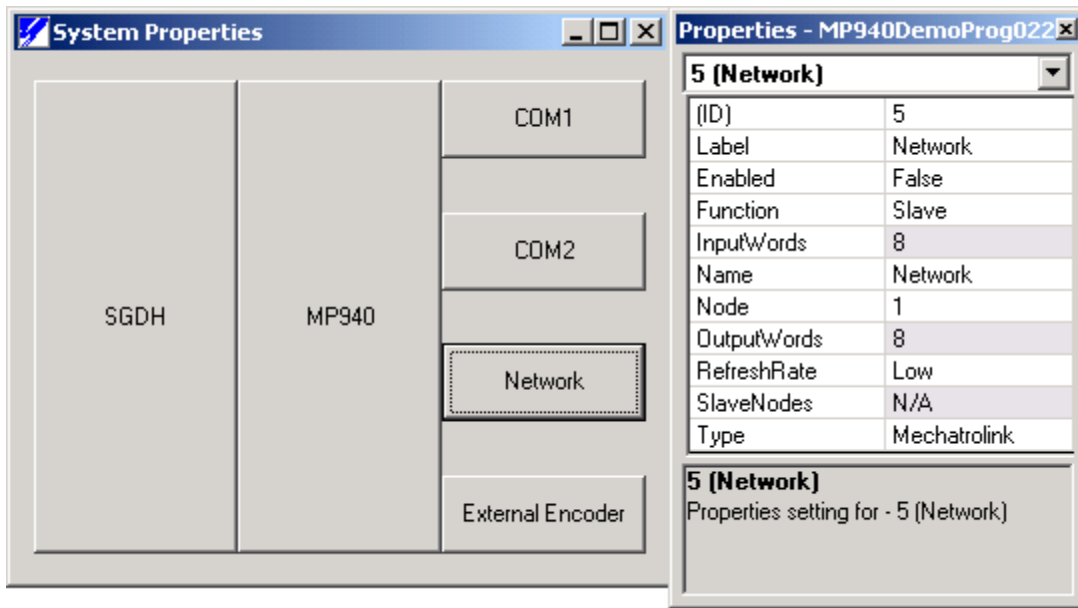
Com1 parameters were left at factory default. Leaving them in this configuration allows for correct connection to MotionWorks+ software.

COM2



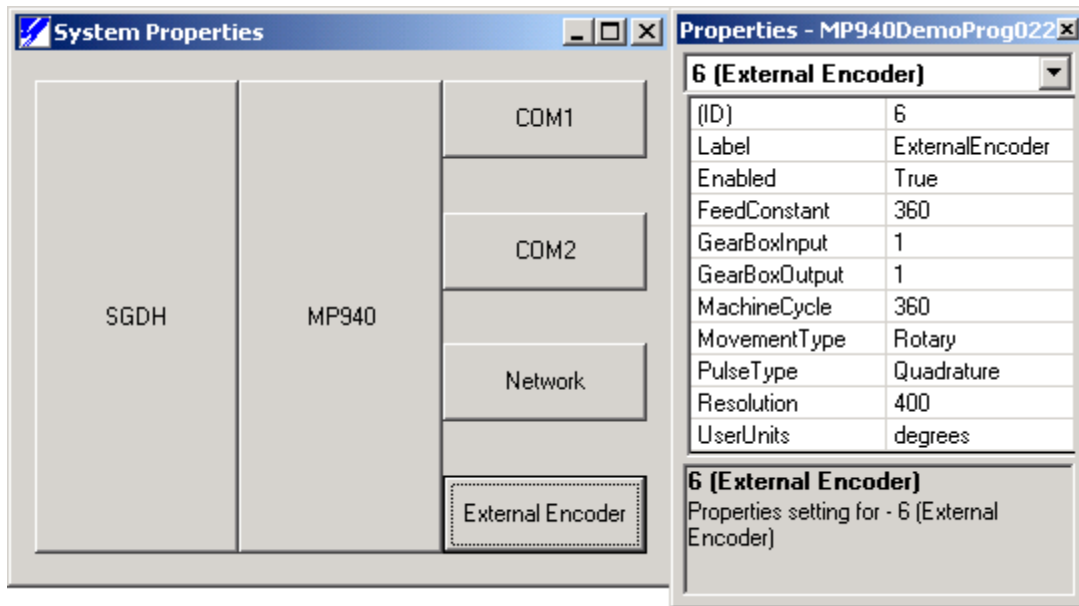
Com2 parameters were left at factory default. Leaving them in this configuration allows for easy connection to Yaskawa HMI. Connection to an HMI is beyond the scope of this document.

Network



The Network system properties were not modified from defaults, as no field bus (Mechatrolink or DeviceNet) was implemented in this project.

External Encoder



Only five parameters were modified in the External Encoder properties to correspond with the demo unit. They are as follows:

- Enabled: True
- FeedConstant: 360 degrees
- MachineCycle: 360 degrees
- Resolution: 400 post quadrature counts
- UserUnits: degrees

These were modified to convert the Pulse Generator of the demo to a rotary mode so that 400 pulses, which is one revolution, equates to 360 degrees and so that it rolls over automatically at 360 degrees.

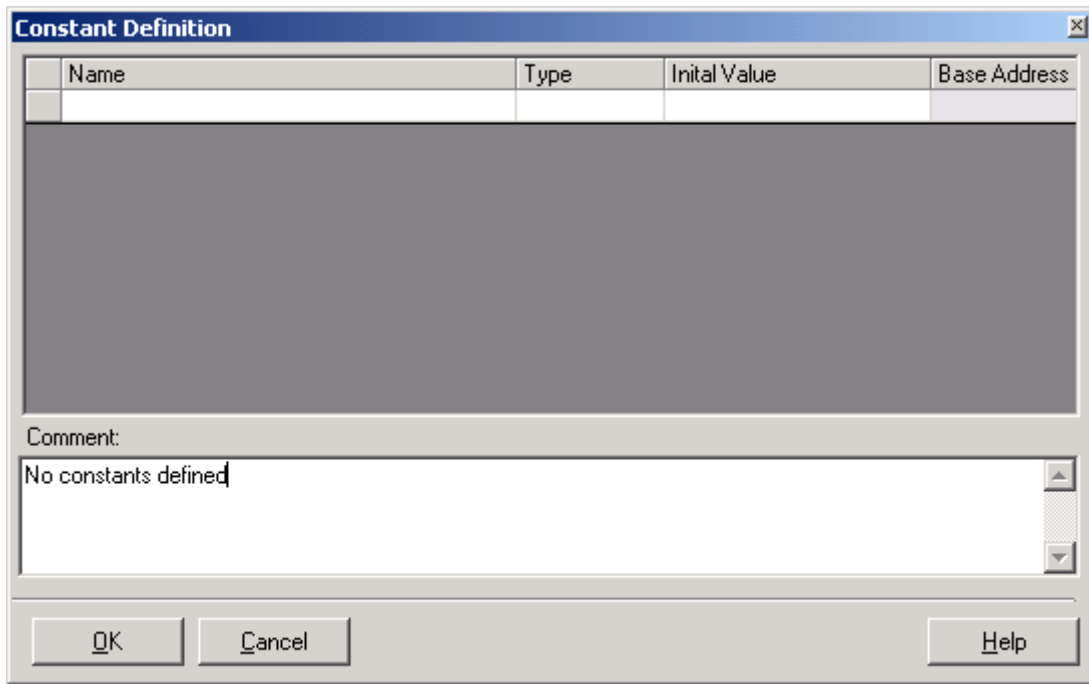
Data

The Data folder contains: Constants, Network, Tables, Variables, I/O, and System Variables configuration.



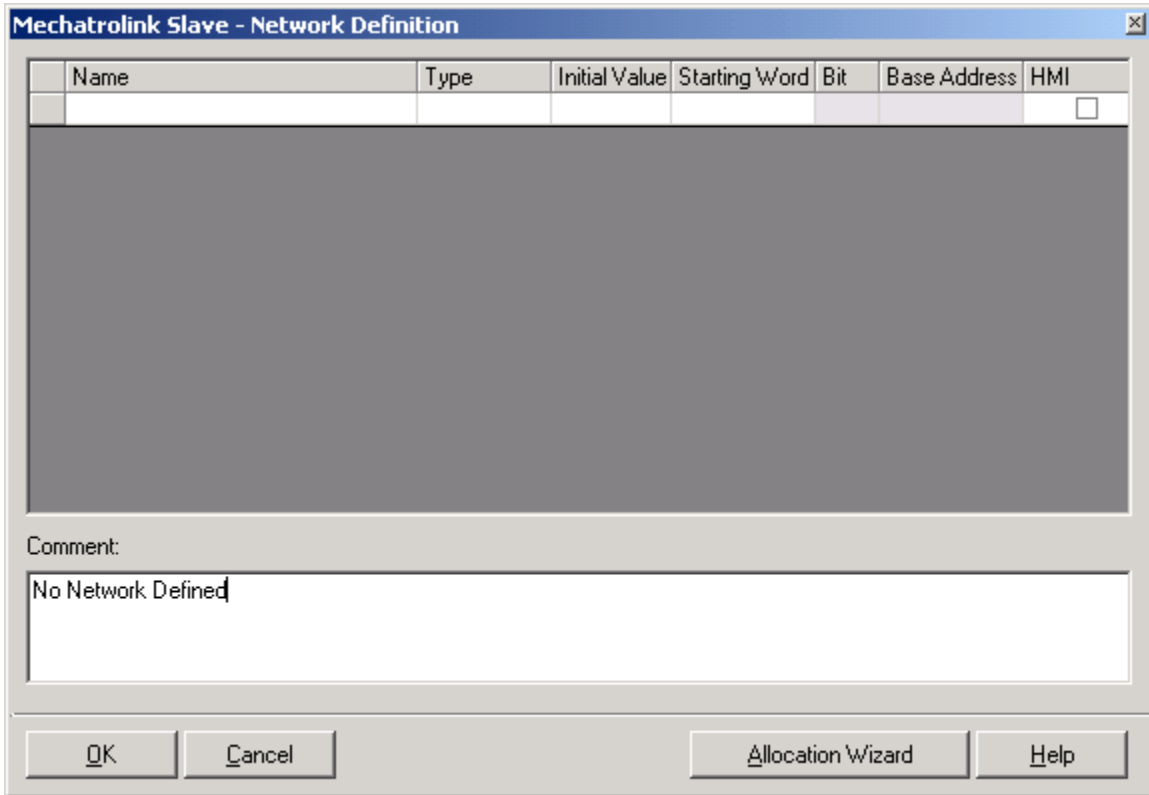
Yaskawa Electric America - 2121 Norman Drive South – Waukegan IL 60085
(800) YASKAWA - Fax (847) 887-7280

Constants



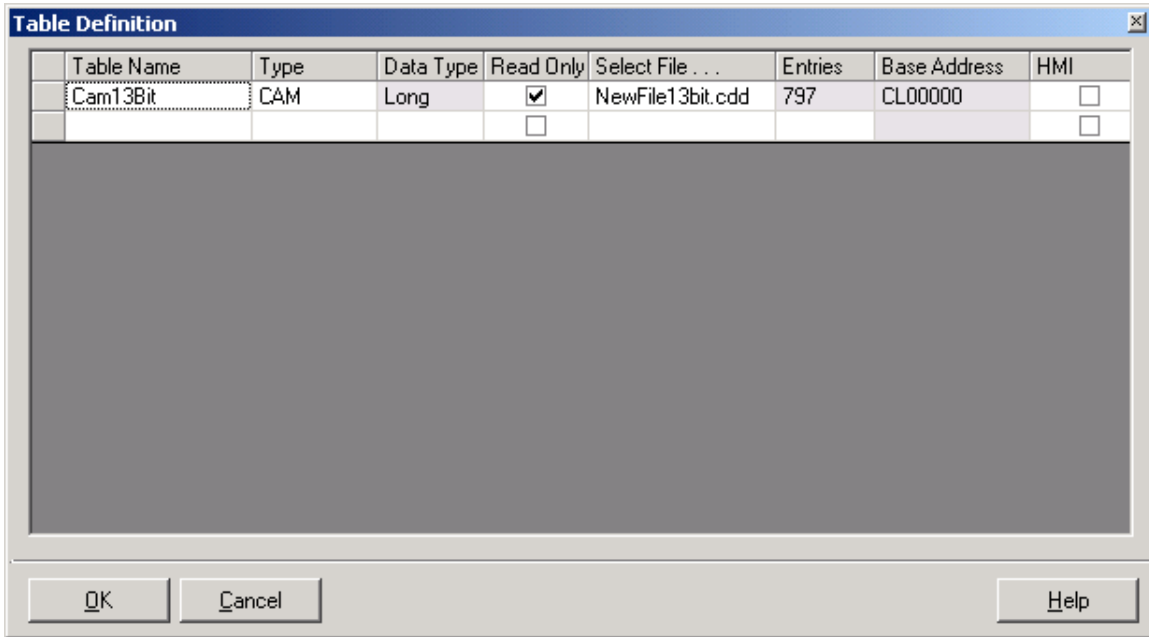
No User constants were defined for this project. This is where the user can define constants to be used in the program.

Network



This demo program is written without the field-bus configured. This is where the user can implement network variables (Mechatrolink or DeviceNet, hardware dependent).

Tables



One CAM table has been created with Yaskawa CamTool. Use of CamTool is out of the scope of this document. Additional CAM tables, CSV files, and empty arrays can be created in the table definition.

Variables

The image shows a 'Variable Definition' dialog box with a table of variables and a comment field. The table has columns for Name, Type, Initial Value, Assign Initial, Base Address, and HMI. The comment field contains the text 'Acceleration (degree/sec^2)'. Buttons for OK, Cancel, and Help are at the bottom.

Name	Type	Initial Value	Assign Initial	Base Address	HMI
Acc	Float	10800	<input checked="" type="checkbox"/>	MF20600	<input type="checkbox"/>
Dec	Float	10800	<input checked="" type="checkbox"/>	MF20602	<input type="checkbox"/>
Vel	Float	360.0	<input checked="" type="checkbox"/>	MF20604	<input type="checkbox"/>
VelJogLimit	Float	3600.0	<input checked="" type="checkbox"/>	MF20606	<input type="checkbox"/>
VelJogIncrement	Float	100.0	<input checked="" type="checkbox"/>	MF20608	<input type="checkbox"/>
VelJogIncrementTime	Integer	500	<input checked="" type="checkbox"/>	MW20610	<input type="checkbox"/>
VelJog	Float	0	<input checked="" type="checkbox"/>	MF20612	<input type="checkbox"/>
VelJogCurrentTime	Long	0	<input checked="" type="checkbox"/>	ML20614	<input type="checkbox"/>
Homed	Bit	false	<input checked="" type="checkbox"/>	MB206110	<input type="checkbox"/>
Homing	Bit	false	<input checked="" type="checkbox"/>	MB206111	<input type="checkbox"/>

Comment:
Acceleration (degree/sec^2)

OK Cancel Help

Several user variables have been created for this program. User can add or modify to suit application needs.

I/O

Name	Type	Initial Value	HMI
Local_Input1	Bit		<input type="checkbox"/>
Local_Input2	Bit		<input type="checkbox"/>
Local_Input3	Bit		<input type="checkbox"/>
Local_Input4	Bit		<input type="checkbox"/>
Local_Input5	Bit		<input type="checkbox"/>
Local_Input6	Bit		<input type="checkbox"/>
Local_Input7	Bit		<input type="checkbox"/>
Local_Input8	Bit		<input type="checkbox"/>
Local_Output1	Bit	false	<input type="checkbox"/>
Local_Output2	Bit	false	<input type="checkbox"/>

Cross Reference:
Local_Input1 (This value is stored at IB00000)

Comment:
No changes from defaults

OK Cancel Help

For ease of use, I/O settings were left at factory default naming conventions. The names could be modified to suit a particular application

Local Input functionality

Input	Functionality		Description
	Manual	Automatic	
MW+ Name (SGDH Ref)			
Local_Input1	Servo Enable		Enables Servo and Starts manual/auto programs
Local_Input2	Off*	On*	*Off Selects Manual, On Selects Automatic
Local_Input3	-	-	unused
Local_Input4	-	-	unused
Local_Input5	Jog Forward	Index	Mode Dependent
Local_Input6	Jog Reverse	Gear	Mode Dependent
Local_Input7	Home	Cam	Mode Dependent
Local_Input8	Torque	Latch	Mode Dependent
Sigma_ServoOn (SI-0)		Cam Shift*	*Only applicable while in Camming
Sigma_HomeInput (SI-1)	Home SW		Home Switch Input
Sigma_POT (SI-2)	P-OT		Positive Over Travel
Sigma_NOT (SI-3)	N-OT		Negative Over Travel
Sigma_EXT1 (SI-4)	Alarm Reset*		*Only applicable after alarm has occurred
Sigma_EXT1 (SI-5)		Slave Offset*	*Only applicable while in Gearing or Camming
Sigma_Latch_Input (SI-6)		Latch Input*	*Only applicable while in Latch routine

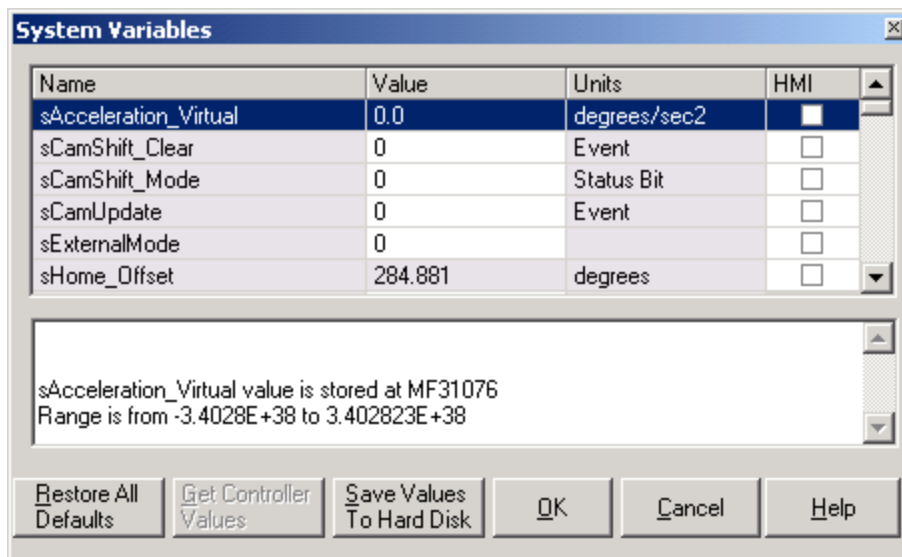
Local Output functionality

Output Functionality Description

MW+ Name All Modes

All Modes		
Local_Output1	Servo Enabled	On when Servo Enabled
Local_Output2	Manual Mode	On when in Manual Mode
Local_Output3	Automatic Mode	On when in Automatic Mode
Local_Output4	Homed / Homing	On when homed, flashing while homing, Off otherwise
Manual Mode		
Local_Output5	Jogging Fwd	Flashing when jogging forward, until max jog speed achieved, then on solid
Local_Output6	Jogging Rev	Flashing when jogging reverse, until max jog speed achieved, then on solid
Local_Output7	Homing	Flashing while homing, on solid when homed, off otherwise
Local_Output8	Applying Torque	Flashing while applying torque, on solid when applied torque=requested torque
Automatic Mode		
Local_Output5	Index	On while indexing
Local_Output6	Gear	On while Gearing
Local_Output7	Cam	On while Camming, Flashing when engaging/disengaging camming
Local_Output8	Latch	On during latch routine
While Indexing		
Local_Output6	PLS	On first third of move
Local_Output7	PLS	On second third of move
Local_Output8	PLS	On last third of move
While Latching		
Local_Output5	PLS	On before Latch Window Starts
Local_Output6	PLS	On during Latch Window
Local_Output7	PLS	On after Latch Window Ends until default distance reached

System Variables



Only four parameters were modified in the System Variables to correspond with the demo unit. They are as follows:

- sLimit_Speed_Negative: 30,000 degrees/sec
- sLimit_Speed_Positive: 30,000 degrees/sec
- sLimit_Torque: -30000 (.01% rated torque)
- sPosition_CompletionWindow: 1.0 degree

These were modified to allow full peak motor rpm (5000), and to allow full motor peak torque to be developed. Lastly, the completion window allows for an incompletely tuned system to be operated with consistency.