Motion with SoMachine

Training Manual

ACE University 2012

Schneider Electric

Contents

_Toc336438563

Revision History

Tab J Smith
Motion Application Specialist
OEM Business

Revision	**Date**	**Comments**
Version 1.0.0.1	09_06_2012	Initial release
Version 1.0.0.2	09_25_2012	Corrected error in Robot State machine. Repositioned State machine description section. Added SoMove configuration of axis address, baud rate, IO, and homing mode.

Chapter 1: Introduction

The following course material is provided to assist in the understanding and development of basic machine control in a motion-centric application. The training is based upon the fundamental concept of Hardware mapping and Functional Mapping to accomplish the operational requirements for simple and complex machines.

SoMachine software is the programming environment, and this course will make use of the SoftStruXure template program. Hardware is provided in the form of the SoMachine Motion training module as described in the following sections. If a comparable training module is not available, the course can be completed with an LMC058 motion controller, and 2 LXM32A axes, with appropriate network cables.

Course Overview

Historically, motion training has been product-focused. OEM machine builders and programmers are well-versed in the specific requirements of their machine, and often only required a fundamental knowledge of the product functionality. However, efficiency, flexibility, productivity and time-to-market pressure are forcing programmers to accommodate more flexibility, along with a larger information stream.

This course is designed to force the programmer to re-think the objectives of programming with the entire machine as the training focus. The software tools provided will help the programmer bypass administrative program tasks, minimize programming errors, and ultimately focus the development effort on the specific requirements of the machine…. as it should be!

Course Objectives

The objectives of this course are to provide the student with the tools and skills necessary to develop a working motion-centric machine as efficiently as possible.

In this course the student will:

- Apply the principles of basic motion axis and movement types
- Apply the PLCopen state diagram in the SoMachine program environment
- Develop re-useable code in the form of Structured Variables.
- Apply the concept of a State Machine
- Apply the SoftStruXure template to program and operate a multi-axis, pick-n-place robot.

It should be clearly understood that this course is does NOT overlook the basic comprehension that comes with years of experience and understanding of mechanical systems. Motion control is a highly complex electro-mechanical process, and proper configuration, programming, and tuning require a thorough understanding of the limitations imposed by control loop technology, mechanical response, and inertia.

Even the most experienced programmer may face an uphill challenge in the development of motion-centric machine functionality without this basic understanding.

Course Organization

The course is designed to accommodate two, 4 hour sessions:

Day 1 (4 hours)

- Fundamentals including Axis type, movement types, and referencing
- Introduction to the SoMachine Motion environment – Hardware Mapping
- Functional Mapping – PLCopen State Diagram
- Functional Mapping – Axis control with SoftMotion
- Creation and use of Interface Structures

Day 2 (4 hours)

- Machine Control – Overview of the SoftStruXure template
- Introduction to State Machines – Program a Pick-n-Place Robot
- Configuration and Application of FDR

Training Materials

Training materials used in this course include:

- SoMachine V3.1 software
- SoMove Commissioning software
- SoftStruXure template project archive
- SoMachine Training Module Hardware
- Ethernet patch cable for programming
- USB to RS485 (RJ45) programming cable for LXM32 commissioing

---

Related Documentation

Additional documentation may be useful for reference purposes:

- SoftStruXure Machine Template User Guide V1.0.0.1
- LXM32A (or M) User Manual

SoMachine Training module

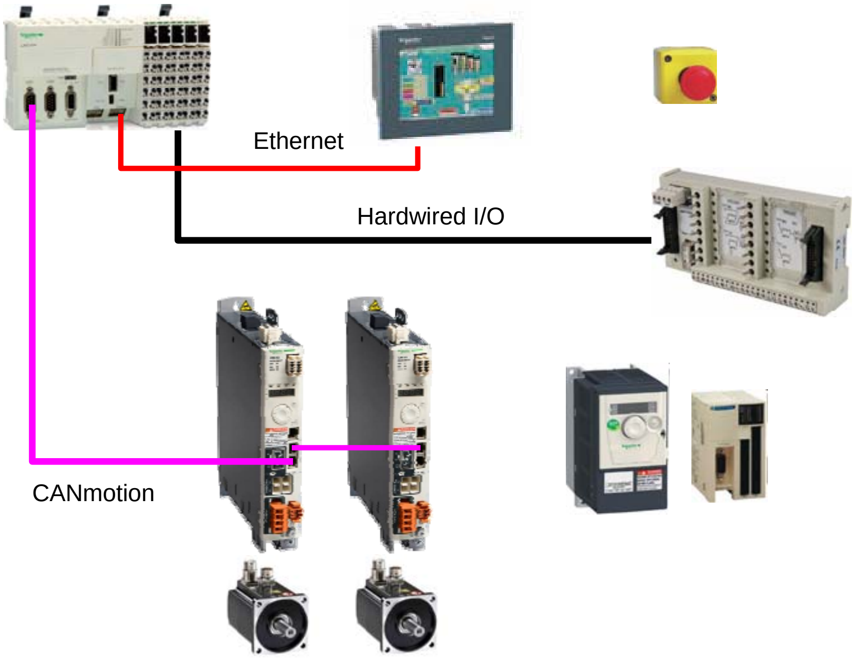
A Machine StruXure training module is provided for this course.

The training module comprises:

- LMC058 Machine StruXure (MSX) controller
- LXM32A servo drives (x2) with BMH motors on CANmotion
- Magelis XBTGC – series HMI
- ATV312 variable frequency drive on CANopen
- Input/Output control block
- Emergency Stop

Training Module

The Module is illustrated in Block diagram form as shown…

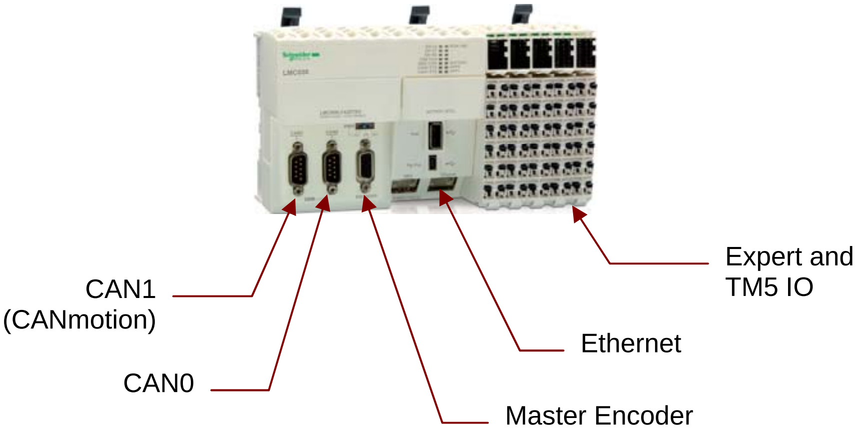


The module can easily be adapted for use with an M258 or other controller as the training content requires.

LMC058 Controller

At the heart of the training module is the LMC058 motion controller. The controller manages general machine tasks, communication and fieldbus networks, and I/O. In addition, it serves as the position path generator for multiple independent or synchronized axes on the CANmotion bus. The LMC058 supports a physical master encoder, as well as multiple virtual axes. These can be used as a pacing axis, or as a master axis in one or more master–slave follower sets.

A few of the relevant hardware connections are shown in the following illustration.



CAN ports 0 and 1 are provided on the controller. Both of these ports can be configured for use as a CANopen master. However, only CAN1 can be configure as a master port for CANmotion.

LXM32A Servo axes

This training focuses on the configuration and operation of a motion-centric machine. Two LXM32A servo drives are provided for use as physical servo axes. These are connected to the LMC058 via the CANmotion (CAN1) port as indicated.

Both Lexium32A and Lexium32M with CANopen fieldbus adapter are supported n the CANopen/CANmotion bus.

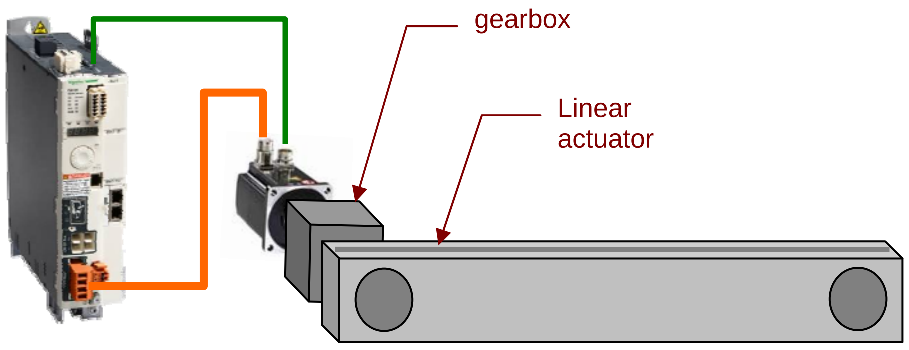
Chapter 2: Motion Fundamentals

The purpose of motion control is to precisely dictate the position and/or velocity of an object. The object is moved by mechanical connection to a servo motor, and the position and speed of the motor are controlled by a drive. Together, the load, power train, motor, and drive, form an axis. In many case, the motion path is critical, and may require the synchronized interaction of a collection of axes. In order to implement motion control in a machine, we have to understand the types of axes available, and the specific movement requirements.

In the following sections, we will take a rudimentary look at axis types, independent and multi-axis movement, and the concept of referencing for a typical motion control system.

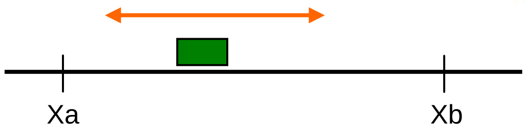
Axis Types

As mention above, an “axis” comprises servo motor, drive, and mechanical power train as illustrated. The response of the axis to a command is determined by the configured axis type



Finite

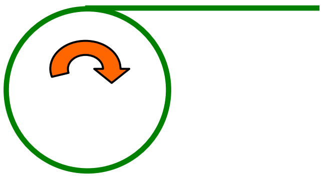
A Finite axis is characterized by a limited working range. Typically, the movement is bi-directional, and knowledge of the exact position within the working range is important for the application.



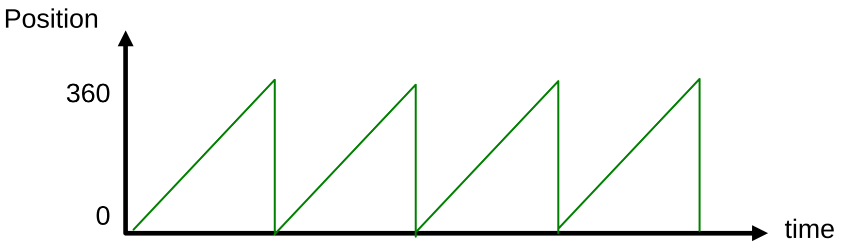
The position reported by a Finite axis will always be within the working range of the axis.

Modulo

A Modulo axis has an “infinite” working range. The position may not be important, indicative of a conveyor. Alternatively, the position may be critical, as in the case of a turntable or rotary knife, for which the position must be reported as a repeating measure of rotation angle.



The position reported by a Modulo axis increases from 0 to the Modulo value, then automatically resets to 0 as the Modulo position is crossed. The characteristic position profile for a rotary axis moving in one direction is a “sawtooth”. In this example, the Modulo value is set to 360 degrees.



Virtual

A Virtual Axis is one of 2 “special” axis types that may be defined as finite or modulo depending upon the implementation. What makes a Virtual axis “special” is that it exists only as a mathematical model. In software, the virtual axis exhibits the same characteristic movement behavior as a real axis.

A virtual axis is most often used as the Master axis in a Master-Slave pair, to set the “pacing” of the machine, or movement synchronization of the slave axes.

When used as a Master axis on a machine, the Virtual axis is often configured as a Modulo axis, for which one revolution (or sawtooth) represents one complete machine cycle.

Master Encoder

Though not a true axis, a machine encoder may be used as a Master axis in the same way as a virtual axis. The main difference is that the encoder is a read-only device that provides the same position and velocity information that would come from the feedback of a virtual (or real) axis.

Movement Paths

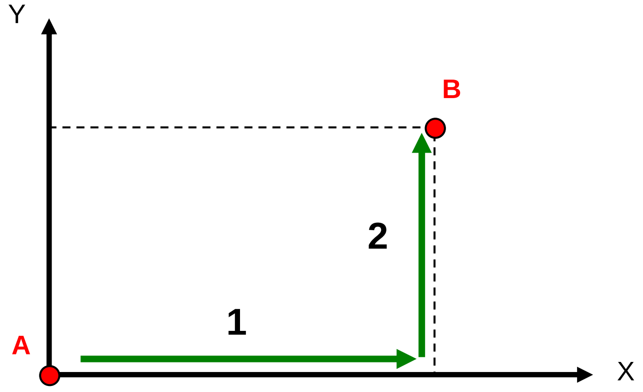
The path followed by an axis or a set of axes is often critical in a machine. For example, wasted movement must be eliminated to achieve high throughput.

It is convenient to classify the movement types for a positioning axis as Point-To-Point (PTP) or Synchronized. A single, positioning axis system is always PTP unless the axis is a slave…following a virtual or encoder master.

Point-to-Point Movement

Consider a pair of axes that move an object from A to B in an XY plane as shown. If the requirement is only to move the object, a simple PTP movement is adequate.

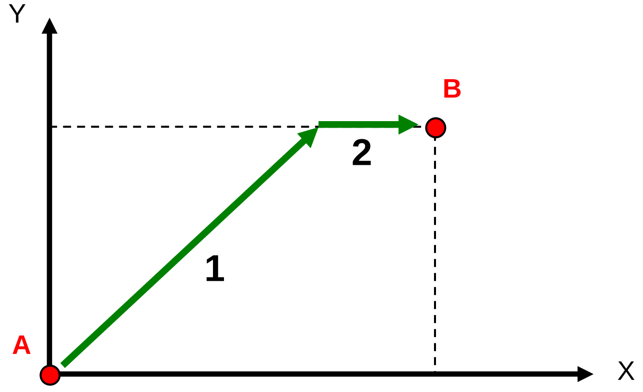
Example 1. The X axis moves first, followed by the Y axis.



This type of movement is simple to control and can be managed by a PLC such as the M258. However, the movement is inefficient because the time required perform each move is added together.

Example 2. The X and Y axes are commanded to move at the same time.

We can improve the efficiency by starting the movements at the same time as shown. Now the move is combined X and Y. Visually, Y completes its movement first, followed by the X axis, due to the slightly longer path length.



In each of these examples, the axes are commanded to move independently. Suppose now that the shortest possible path is required, or that a precise non-linear path must be used to avoid an obstacle. This leads to the requirement for axis synchronization.

Synchronized Movement

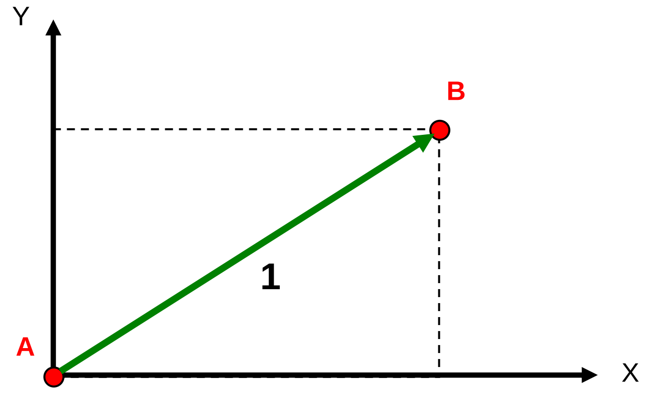
Unlike the independent PTP movement commands described above, synchronized axes behave as a “set”, and respond to a single movement command that manages the path for all of the included axes. Synchronization requires a path planner in a Motion controller such as the LMC058.

SoMachine provides 2 options for axis synchronization using the native SoftMotion functions. These are:

1. Interpolation, and
2. Master-Slave control

Example 3. The X and Y axes are interpolated.

A single command moves X and Y together along a straight line path given a target coordinate (X,Y) and a velocity. The motion controller determines the individual acceleration, deceleration, and velocities required to complete the path in one continuous move.



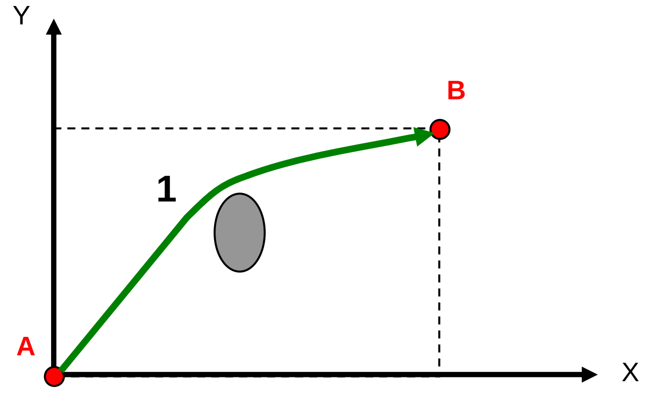
Interpolation is programmed in the SoMachine environment using G-Codes.

G-code programming is part of the CNC library functions, and is not covered in this training.

Note that the linear Example 3 above could also be processed using electronic gear-based Master Slave synchronization

Example 4. An obstacle is avoided using a Master Slave Cam profile.

A single command moves X and Y together along a predetermined path. The X and Y axes are each slaves to a common Master (virtual), and the master is given a single PTP move command. The slave positions are interpolated by a CAM profile.

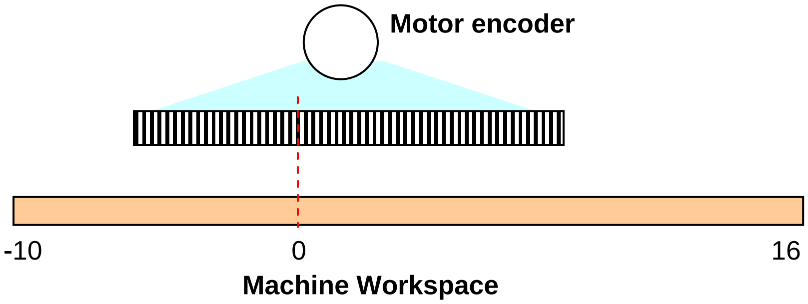


Referencing

How do we know *where* to command an axis to move?

When a machine moves a load, the actual target position is meaningless until the axis that moves the load is “referenced to the machine.

An axis is referenced by associating a specific position of the motor feedback to a corresponding location on the machine.

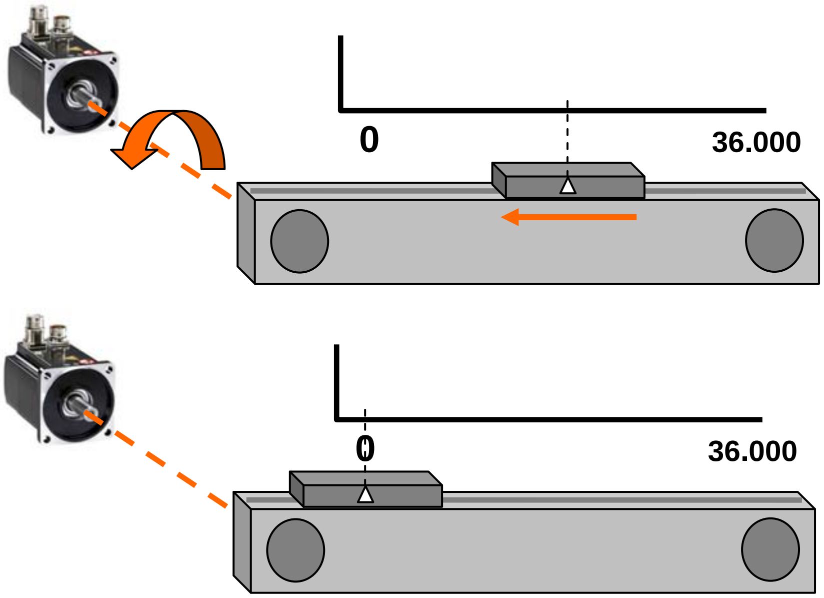


Referencing can be accomplished by:

- Aligning the axis to the Machine (Referencing move)
- Aligning the Machine to the axis (Set position)

Referencing Move

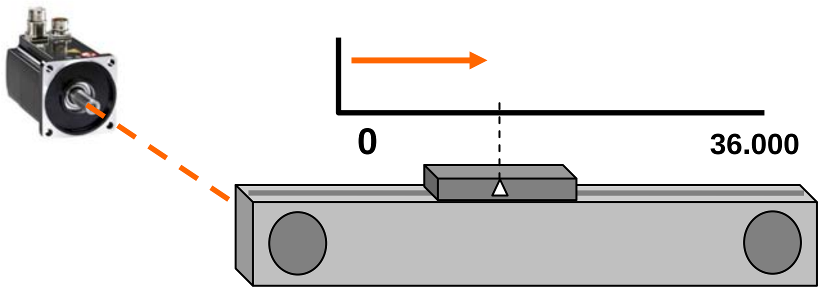
When a reference move is performed, the axis rotates the motor shaft until the load reaches a reference position (generally a switch) on the machine. The motor then stops, and the Axis position (encoder offset position) is set to the machine position.



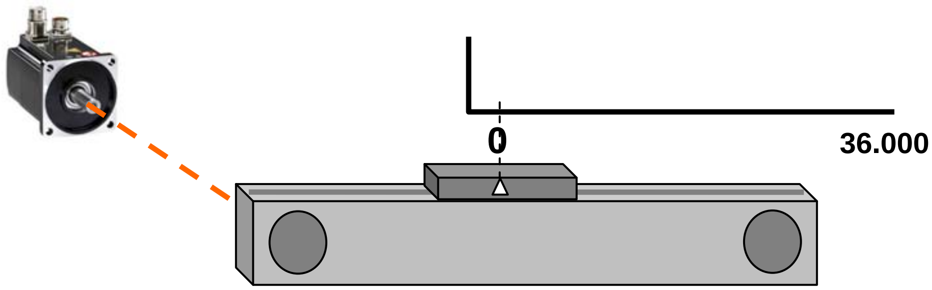
At the conclusion of the movement, the axis is assigned the mechanical position of the machine.

Set Position

In some cases, the machine mechanical position is arbitrary, and it is only necessary to align the motor and machine (without moving the axis) for subsequent relative movement. This can be done using a Set Position command.



Here, the machine reference is "moved" to accommodate the current position of the motor, and the axis is set to the reference position.



Multi-turn Absolute Encoders

The machine referencing process described above is required every time the machine is powered up. However, for some machines, particularly those with very high axis count, this is too inconvenient (or time consuming) to be practical. The reference process can be reduced to a one-time event by using multi-turn absolute encoders.

These encoders capture the shaft position as well as the motor revolution count. Once referenced, the multi-turn absolute encoder will provide the actual motor position on the machine on every power cycle.

Additional information

The Lexium32 supports a wide variety of Referencing movements, Please see the Lexium32A User Manual for a complete listing and description.

Next up

In the next section, we will explore the basic commissioning of the Lexium32 drive, and prepare the axis for use on the CANmotion (or CANopen) bus.

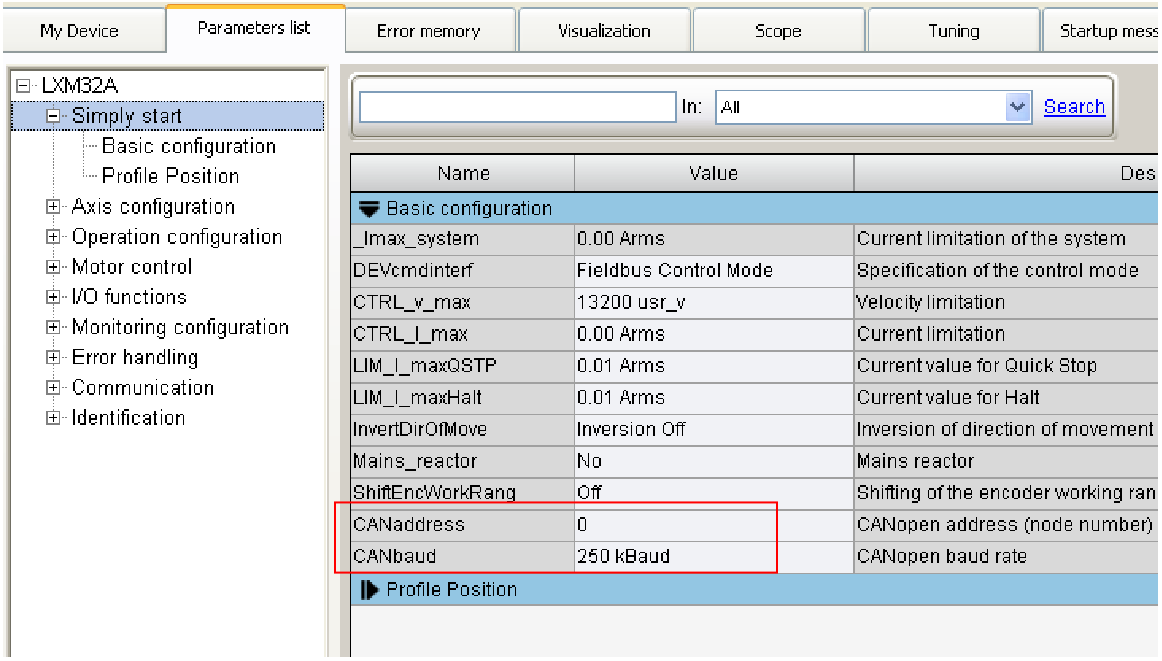
Chapter 3: Axis Commissioning

In order for the Lexium32 servo axis to operate as intended on the CANmotion (or CANopen) bus, the axis must be assigned a unique CANopen address (Node ID). In addition, the communication baud rate must be configured to match the baud rate of the CANopen master.

In the following sections, we will take a look at the configuration of the communication parameters for the Lexium 32 on CANopen using **SoMove lite** software. In addition, we will take some simple tests to verify proper feedback, commutation, and operation of the axis.

CANmotion / CANopen parameters

Access to the fieldbus communication parameters for a slave device on CANopen is provided in the Simply Start menu of SoMove.



Name	Value	Des
Basic configuration		
_Imax_system	0.00 Arms	Current limitation of the system
DEVcmdinterf	Fieldbus Control Mode	Specification of the control mode
CTRL_v_max	13200 usr_v	Velocity limitation
CTRL_I_max	0.00 Arms	Current limitation
LIM_I_maxQSTP	0.01 Arms	Current value for Quick Stop
LIM_I_maxHalt	0.01 Arms	Current value for Halt
InvertDirOfMove	Inversion Off	Inversion of direction of movement
Mains_reactor	No	Mains reactor
ShiftEncWorkRang	Off	Shifting of the encoder working ran
CANaddress	0	CANopen address (node number)
CANbaud	250 kBaud	CANopen baud rate
Profile Position		

Unless the bus length is unusually long (> 20m), the recommended baud rate for an axis on the CANmotion bus is **1000 kBaud**.

By convention, axes on the CANmotion bus are addressed in order from 1 to 8.

Axis Commissioning

Axis commissioning is generally the first step in the preparation of a servo axis on a physical machine. The commissioning process confirms:

- Proper electrical wiring
- Proper position feedback connection between the motor and drive
- Proper commutation of the motor
- Basic settings including input function and motor rotation direction
- Proper “Homing” behavior

For safety reasons, and as a general rule of caution, these steps are usually performed with the motor ***uncoupled*** *from the machine.*

Although this can be done directly from the SoMachine programming environment, it is usually more convenient, and more informative, to commission the axes using a dedicated tool. In this way, any potential problems associated with the PLC, motion controller, or programming are avoided.

The Lexium32 is an extremely user-friendly and informative device. The front panel LED display provides a wealth of information regarding basic electrical power wiring, feedback, motor connection, and axis status by way of LED status or fault codes.

Most of the important commissioning settings can be accessed, and edited from the front panel HMI. However, for the purposes of this training, we will use **SoMove lite** software to perform the basic commissioning steps as outlined above

---

Electrical Power Wiring

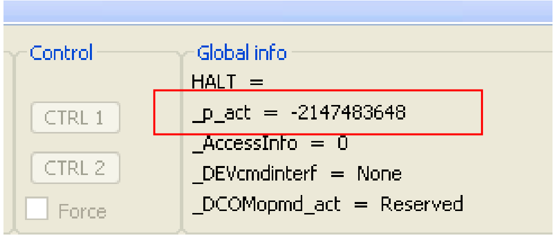
Depending upon the model number, the Lexium32 drive will operate over a wide range of AC input voltages including single phase 115 / 230 VAC, or 208 to 480 VAC three-phase. In order to safely monitor the DC bus voltage and AC Mains input power, the drive must be configured for the correct mains voltage.

The default setting is *Automatic Mains detection*, and generally no further intervention is required.

In some cases, it may be useful to modify this setting. This is generally required when the drive is powered from an external DC bus.

Encoder Feedback

Generally the front panel LED display will indicate any problems associated with encoder feedback by means of a fault code. Proper feedback can also be verified by monitoring the actual position on the SoMove **Command panel**. WIth 24 V logic power applied, and with the axis disabled, the motor shaft is turned manually. The actual position should increase smoothly for clockwise rotation as viewed from the shaft tip.



HALT =
_p_act = -2147483648
_AccessInfo = 0
_DEVcmdinterf = None
_DCOMopmd_act = Reserved

Commutation

Motor commutation can be confirmed by several methods. With AC input power and 24 VDC logic power applied, the Axis can be ***Enabled*** from the Command panel. The motor shaft should lock into position with a firm resistance to manual rotation. Any tendency of the shaft to “jump” from one pole position to another would be an indication of a commutation problem.

The most common cause of improper motor commutation is the reversal of two of the 3 motor leads at the drive-side motor connector during installation.

Commutation can also be verified by performing a simple jog movement in both directions. The motor shaft should rotate smoothly with no evidence of jerk or pole jump.

A Jog test will also confirm proper rotational direction of the motor shaft. If the default setting is inconvenient for the machine coordinate system, the motor rotation sense is easily edited within SoMove.

Application Settings

Remaining configuration settings are generally application specific, and may include:

- Homing methods and homing parameters
- IO function
- Motor rotation direction
- AC mains configuration

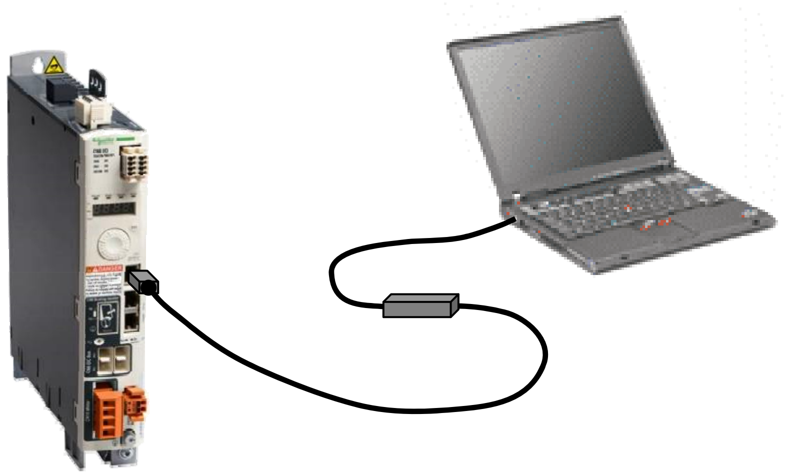
In the following exercise, we will configure the 2 physical axes in the training module for use on the CANmotion bus. This will require a unique CANopen address and baud rate settings, as well as IO function configuration, and homing method.

Exercise – Lexium32 Commissioning

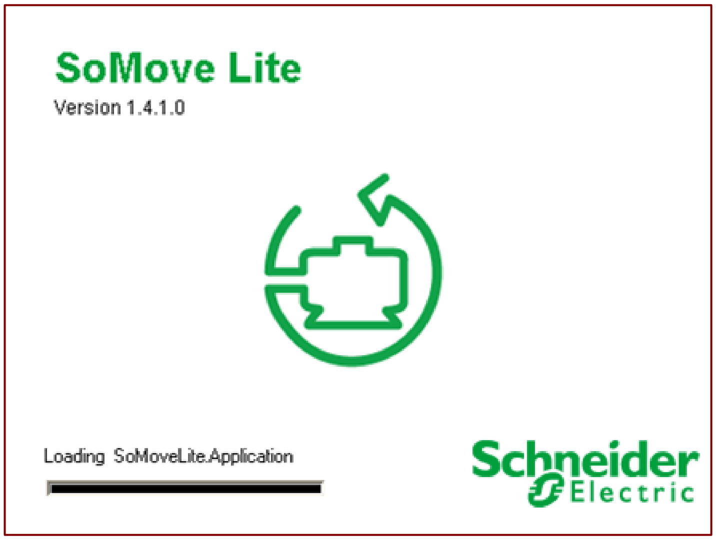
1. Connect to LXM32 drive using SoMove Lite

i. Make sure that the Lexium32 drive is supplied with 24Vdc logic power. A circuit break is provided on the training module to supply both AC mains and 24 V dc logic power. The front panel LED display should be illuminated as shown.

ii. Connect the USB/RS485 programming cable from the PC to the Modbus port on the LXM32 drive.



iii. Start **SoMove Lite** software…



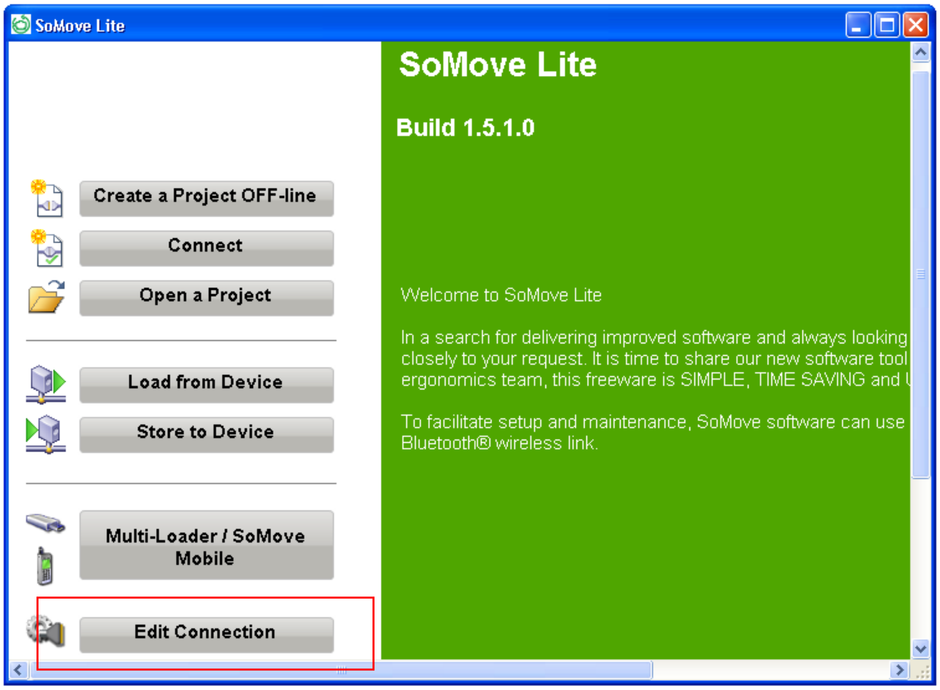
SoMove Lite

Version 1.4.1.0

Loading SoMoveLite.Application

Schneider Electric

iv. … and select **Edit Connection** to configure the connection settings.

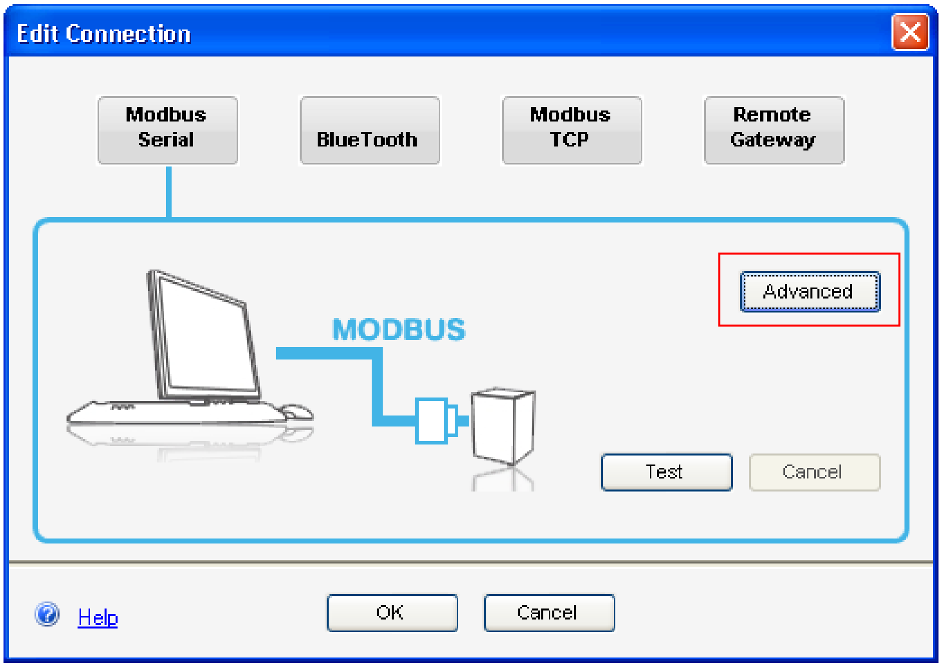


Welcome to SoMove Lite

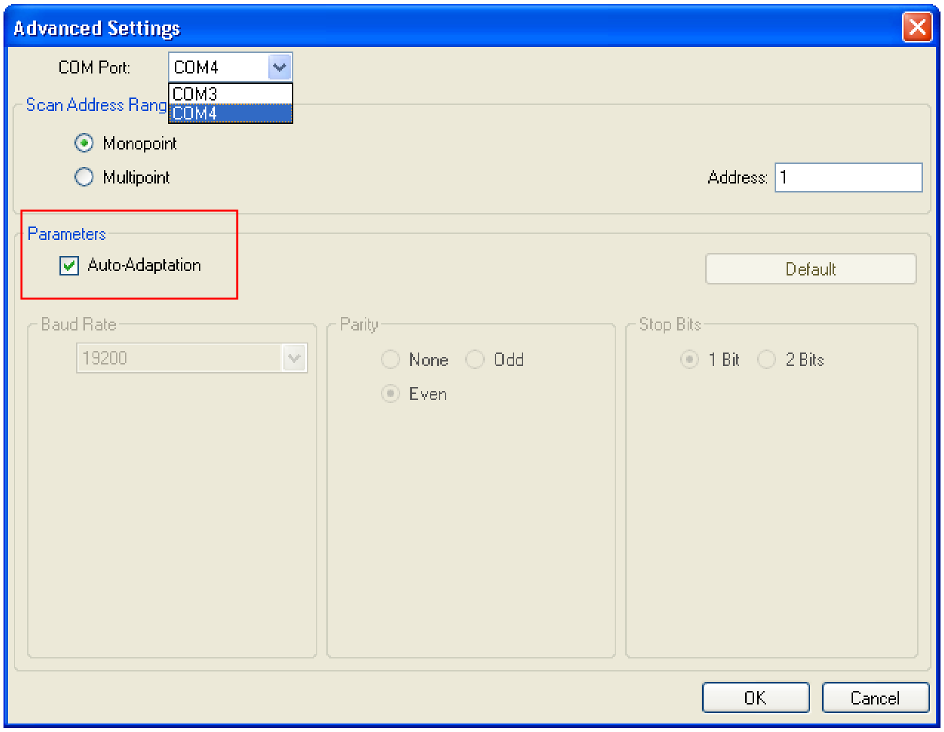
In a search for delivering improved software and always looking closely to your request. It is time to share our new software tool ergonomics team, this freeware is SIMPLE, TIME SAVING and

To facilitate setup and maintenance, SoMove software can use Bluetooth® wireless link.

v. Select **Modbus Serial** as indicated, and click **Advanced**.



vi. Choose the correct **COM** Port for your programming cable, and check **Auto-Adaptation** as shown. Select OK to continue.



Advanced Settings

COM Port: COM4

Scan Address Range: ● Monopoint ○ Multipoint Address: 1

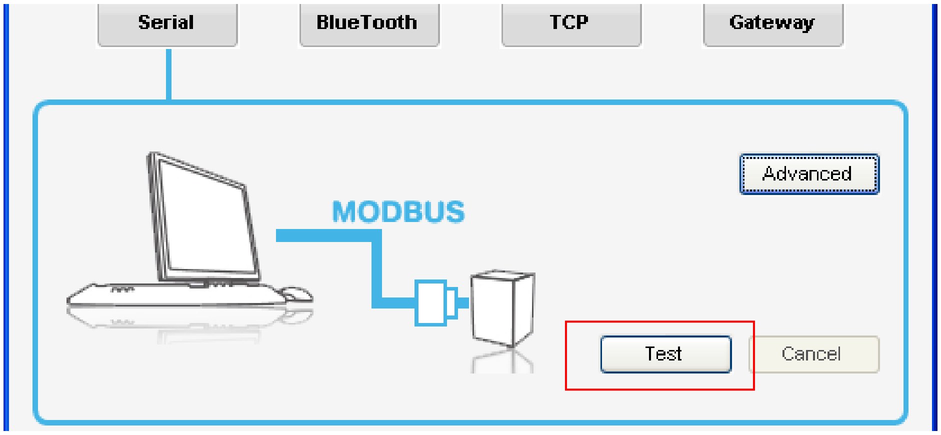
Parameters: ☑ Auto-Adaptation

Baud Rate: 19200

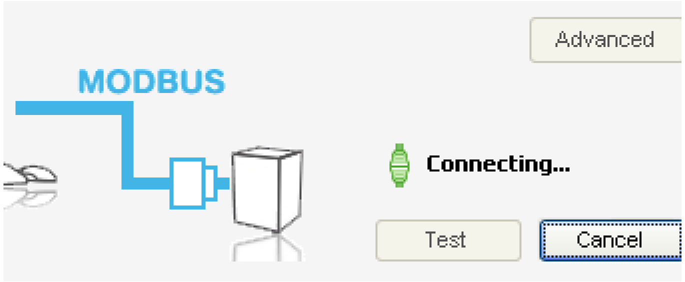
Parity: ○ None ○ Odd ● Even

Stop Bits: ● 1 Bit ○ 2 Bits

vii. Select **Test** from the Edit Connection screen.

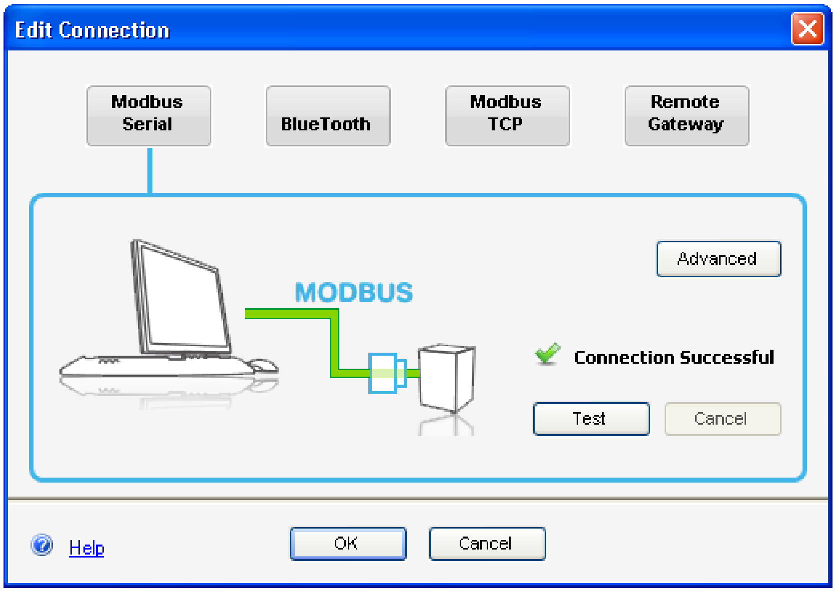


The screen will respond with the connection in progress.



Connecting...

viii. When the Connection has completed successfully, Select OK to adopt the settings.



✓ Connection Successful

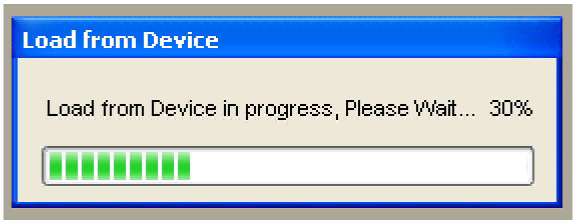
ix. From the Main screen, select **Connect** to connect to the drive.

Welcome to SoMove Lite

In a search for delivering improved software and always looking closely to your request. It is time to share our new software tool ergonomics team, this freeware is SIMPLE, TIME SAVING and

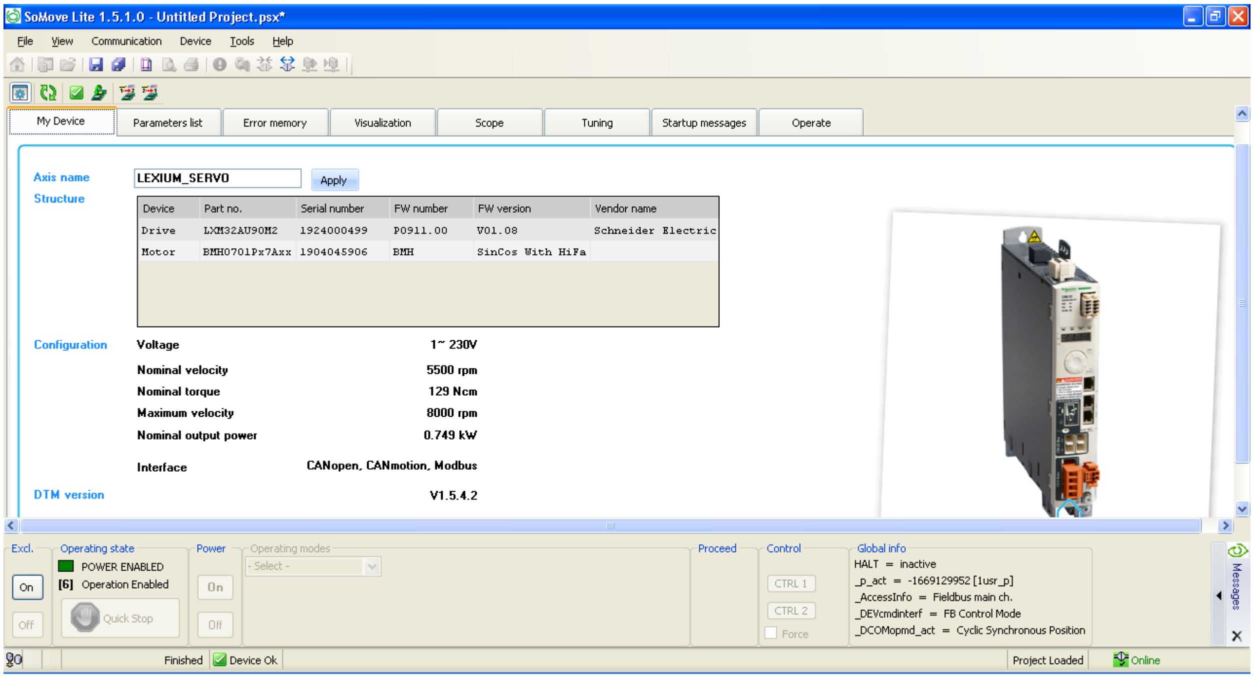
To facilitate setup and maintenance, SoMove software can use Bluetooth® wireless link.

This will connect SoMove to the device, and load current device parameters as shown.



Load from Device in progress, Please Wait... 30%

The Main screen will appear when the device parameter update has completed



Axis name: LEXIUM_SERVO

Structure

Device	Part no.	Serial number	FW number	FW version	Vendor name
Drive	LXM32AU90M2	1924000499	P0911.00	V01.08	Schneider Electric
Motor	BMH0701Px7Axx	1904045906	BMH	SinCos With HiFa	

Configuration

Voltage	1~ 230V
Nominal velocity	5500 rpm
Nominal torque	129 Ncm
Maximum velocity	8000 rpm
Nominal output power	0.749 kW
Interface	CANopen, CANmotion, Modbus

DTM version: V1.5.4.2

Operating state: POWER ENABLED
[6] Operation Enabled

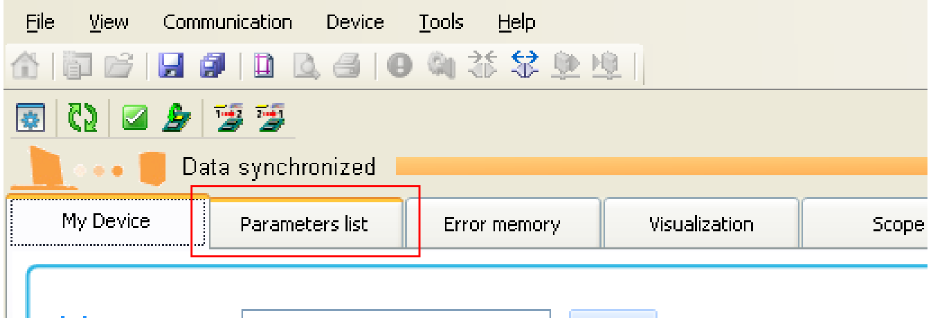
Global info:
HALT = inactive
_p_act = -1669129952 [1usr_p]
_AccessInfo = Fieldbus main ch.
_DEVcmdinterf = FB Control Mode
_DCOMopmd_act = Cyclic Synchronous Position

Finished ✓ Device Ok | Project Loaded | Online

---

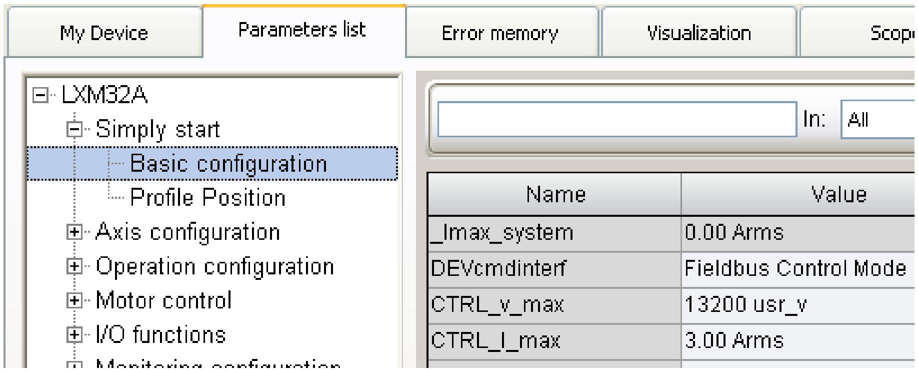
2. Set the CANopen Address and Baud rate

i. Select the **Parameters list**



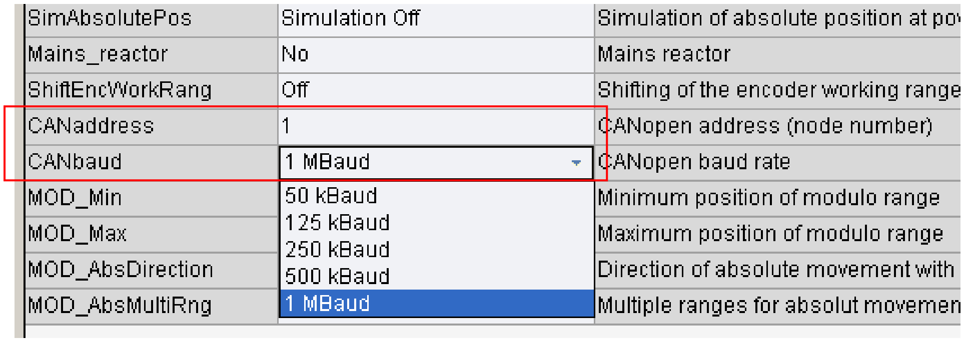
Data synchronized

ii. From the Parameters list, select **Simply start >> Basic Configuration**.



Name	Value
_Imax_system	0.00 Arms
DEVcmdinterf	Fieldbus Control Mode
CTRL_v_max	13200 usr_v
CTRL_I_max	3.00 Arms

iii. Enter the CANopen address 1, and a baud rate of 1000 kBd as shown.



SimAbsolutePos	Simulation Off	Simulation of absolute position at po
Mains_reactor	No	Mains reactor
ShiftEncWorkRang	Off	Shifting of the encoder working range
CANaddress	1	CANopen address (node number)
CANbaud	1 MBaud	CANopen baud rate
MOD_Min	50 kBaud	Minimum position of modulo range
MOD_Max	125 kBaud	Maximum position of modulo range
MOD_AbsDirection	250 kBaud 500 kBaud	Direction of absolute movement with
MOD_AbsMultiRng	1 MBaud	Multiple ranges for absolut movemen

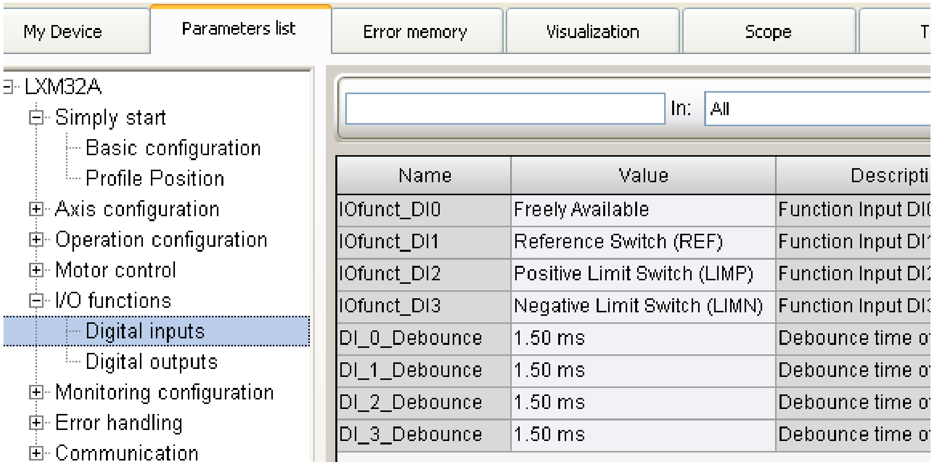
---

3. Configure the Input Functions

i. From the Parameters list, select **I/O Functions >> Digital inputs.** The default settings for digital inputs are:

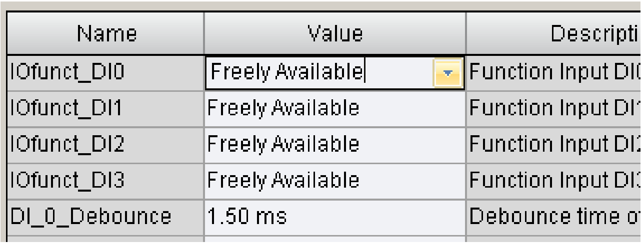
- DI0 – Freely Available
- DI1 – REF switch
- DI2 – LIMP switch
- DI3 – LIMN switch

Since there is no hardware input wiring to the drives in the training module, we will set these inputs all to be “Freely Available”. This will prevent a drive STOP error due to travel limit switch activation.



Name	Value	Descripti
IOfunct_DI0	Freely Available	Function Input DI
IOfunct_DI1	Reference Switch (REF)	Function Input DI
IOfunct_DI2	Positive Limit Switch (LIMP)	Function Input DI
IOfunct_DI3	Negative Limit Switch (LIMN)	Function Input DI
DI_0_Debounce	1.50 ms	Debounce time o
DI_1_Debounce	1.50 ms	Debounce time o
DI_2_Debounce	1.50 ms	Debounce time o
DI_3_Debounce	1.50 ms	Debounce time o

ii. Set each of the input functions to Freely Available as shown.

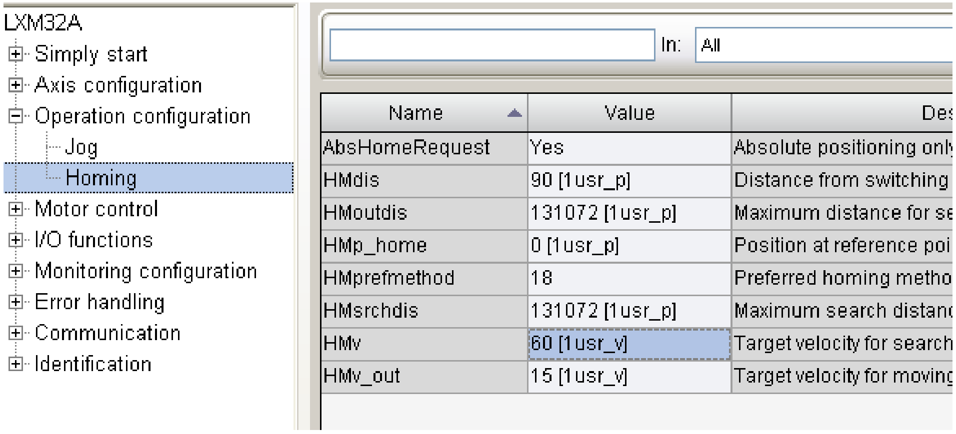


Name	Value	Descripti
IOfunct_DI0	Freely Available	Function Input DI
IOfunct_DI1	Freely Available	Function Input DI
IOfunct_DI2	Freely Available	Function Input DI
IOfunct_DI3	Freely Available	Function Input DI
DI_0_Debounce	1.50 ms	Debounce time o

4. Axis Referencing

Referencing by movement requires that the motor shaft move the load to a known position which may be a REFerence switch, Positive travel limit (LIMP), negative travel limit (LIMN), or to the motor index pulse. Reference movement is configured in the “Homing” parameter screen.

i. From the **Parameters list**, select **Operation configuration >> Homing**

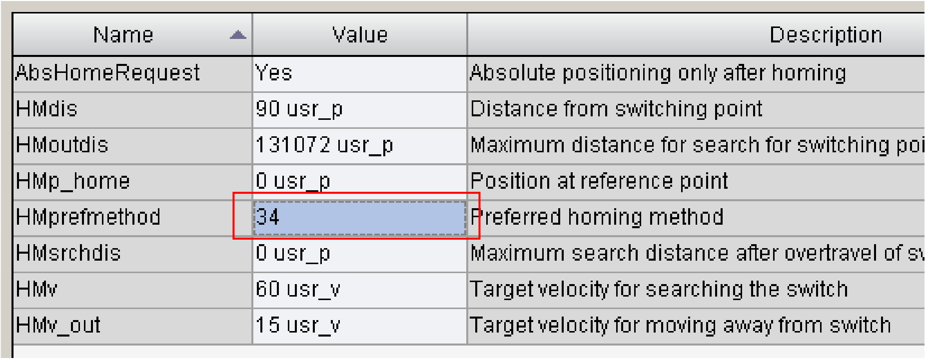


Name	Value	Des
AbsHomeRequest	Yes	Absolute positioning onl
HMdis	90 [1usr_p]	Distance from switching
HMoutdis	131072 [1usr_p]	Maximum distance for se
HMp_home	0 [1usr_p]	Position at reference poi
HMprefmethod	18	Preferred homing metho
HMsrchdis	131072 [1usr_p]	Maximum search distanc
HMv	60 [1usr_v]	Target velocity for search
HMv_out	15 [1usr_v]	Target velocity for moving

The three primary configuration parameters for homing are:

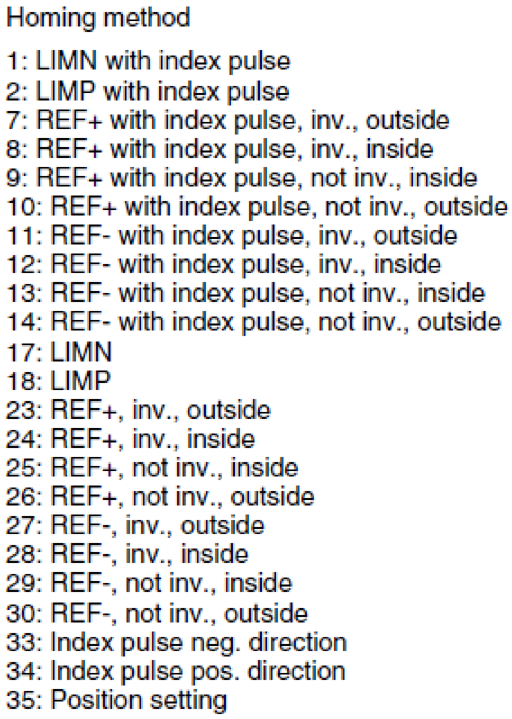
- HMprefmethod
- HMv (homing speed)
- HMp_home (reference position)

ii. Select the Homing method 34 [**Idx Positive]**. This will initiate a motor rotation in the positive direction to find the encoder index mark. The homing speed of 60 [usr_v] will produce a search speed of 60 RPM, or one motor rev per sec.



Name	Value	Description
AbsHomeRequest	Yes	Absolute positioning only after homing
HMdis	90 usr_p	Distance from switching point
HMoutdis	131072 usr_p	Maximum distance for search for switching poi
HMp_home	0 usr_p	Position at reference point
HMprefmethod	34	Preferred homing method
HMsrchdis	0 usr_p	Maximum search distance after overtravel of s\
HMv	60 usr_v	Target velocity for searching the switch
HMv_out	15 usr_v	Target velocity for moving away from switch

A complete description of the available Homing methods and parameters is available in the Lexium32 User Manual.



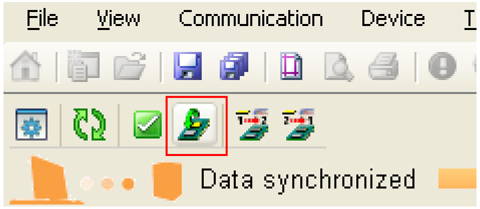
Homing method

1: LIMN with index pulse
2: LIMP with index pulse
7: REF+ with index pulse, inv., outside
8: REF+ with index pulse, inv., inside
9: REF+ with index pulse, not inv., inside
10: REF+ with index pulse, not inv., outside
11: REF- with index pulse, inv., outside
12: REF- with index pulse, inv., inside
13: REF- with index pulse, not inv., inside
14: REF- with index pulse, not inv., outside
17: LIMN
18: LIMP
23: REF+, inv., outside
24: REF+, inv., inside
25: REF+, not inv., inside
26: REF+, not inv., outside
27: REF-, inv., outside
28: REF-, inv., inside
29: REF-, not inv., inside
30: REF-, not inv., outside
33: Index pulse neg. direction
34: Index pulse pos. direction
35: Position setting

---

5. Save to EEPROM

i. Save the parameters to EEPROM so that the drive accepts the changes during the next boot cycle.



---

6. On your own… Repeat

i. Repeat this entire sequence for the second drive to provide a unique address (address 2) as required for the 2-axis exercises later in the training course.

This concludes the Exercise

Chapter 4: Motion with SoMachine

SoMachine software provides a powerful and convenient user interface for the mapping of machine hardware and functionality. This chapter provides an overview of the SoMachine project browser, the native hardware map for the LMC058, and the SoftMotion tools provided for single and multi-axis motion programming.

SoMachine Browser

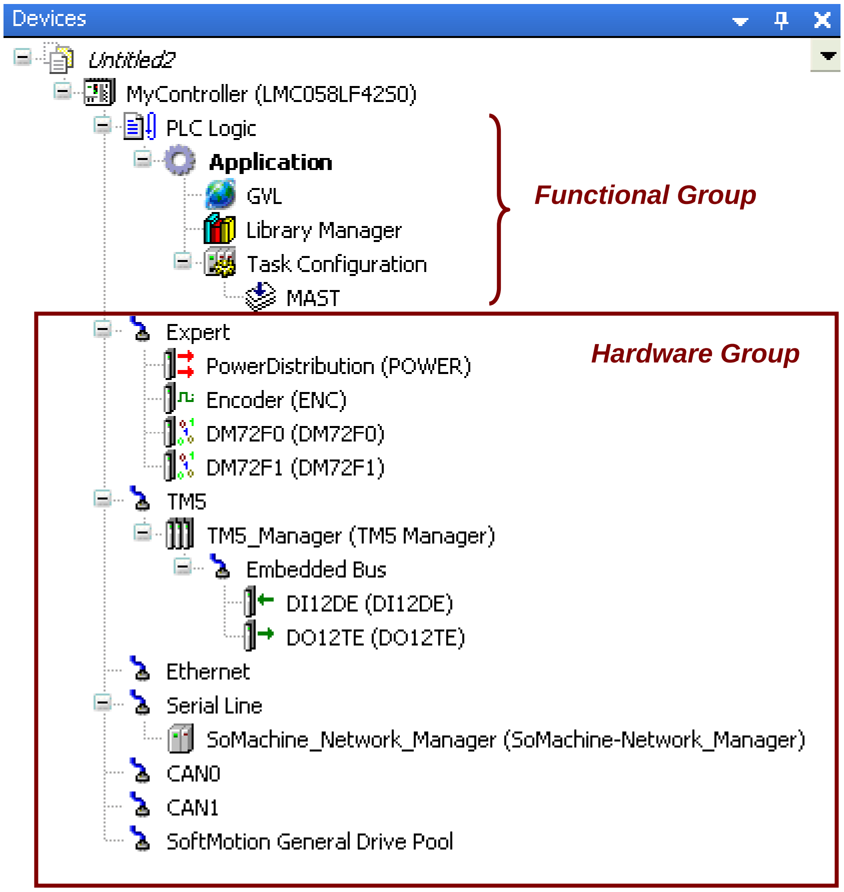
The SoMachine browser contains all of the necessary components for mapping the native hardware and functionality for the selected controller. In this section, we will take a quick look at the main browser objects required for a motion-centric machine application.

We will examine and perform the steps necessary to configure several axes of motion on the CANmotion bus.

Browser at a Glance

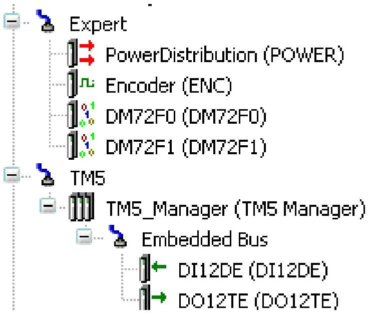
For mapping purposes, the browser provides a clear grouping of hardware and functional objects depending upon the controller.

These groups are defined for the LMC058 motion controller as shown.

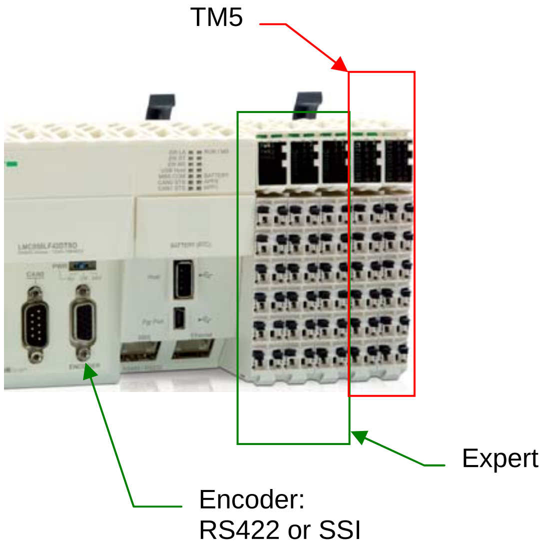


Expert and TM5 Standard IO

Expert IO includes multi-purpose fast IO that can be implemented as simple discrete points, or configured as specialized objects such as high-speed counters or motion encoders. A dedicated D-sub connector is provided for the SSI / RS422 SoftMotion Master Encoder interface.



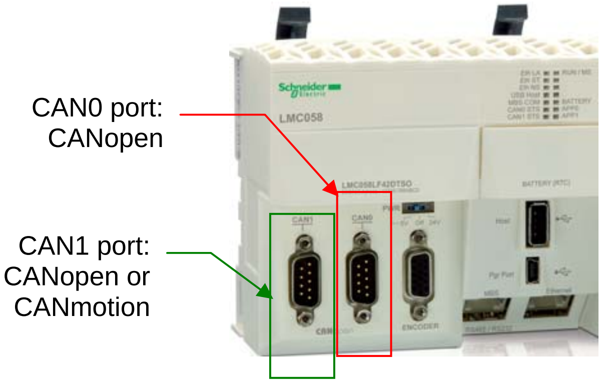
- Expert
  - PowerDistribution (POWER)
  - Encoder (ENC)
  - DM72F0 (DM72F0)
  - DM72F1 (DM72F1)
- TM5
  - TM5_Manager (TM5 Manager)
    - Embedded Bus
      - DI12DE (DI12DE)
      - DO12TE (DO12TE)



CAN ports

Two CAN hardware ports are provided on the LMC058 controller. These can be mapped as CANopen “Performance” or CANmotion ports as indicated.

- CAN0
- CAN1



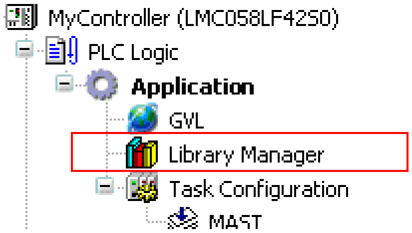
Mapping both of these ports as CANopen provides 2 independent CANopen masters. This can increase the number of CANopen devices that can be managed without excessive “burden” on the network.

Mapping CAN0 as CANopen, and CAN1 as CANmotion, provides the ability to mix CANopen devices with synchronous servo axes in the machine application.

Functional Group Libraries

SoMachine software retains much of the basic interface of a typical “C program” including the declaration of variables, included libraries, program build and compilation to create an executable program for download to the target device.

All of the native libraries required to manage Machine StruXure devices or functionality, are maintained in a Repository. Application-specific libraries are added to (included in) the application automatically, as devices or ports are configured. Included libraries are located in the **Library Manager**.



SoftMotion is the library structure for independent and multi-axis motion control functions. SoftMotion includes the PLCopen libraries as previously discussed, as well as drive interface (Motion bus) functions, the Cam editor, CNC editors, and error management.

POUs

Program Organization Units (or POUs) are the program code sections of a SoMachine application. Program sections are created using IEC 6-1131 compliant programming languages and tools including Function Block, Structured Text, Sequential Function Chart, Instruction List, Ladder, and Continuous Function Chart.

POUs are attached to (or called by) Tasks. A POU that has been created but not assigned to a task *will not execute* (solve) in RUN mode

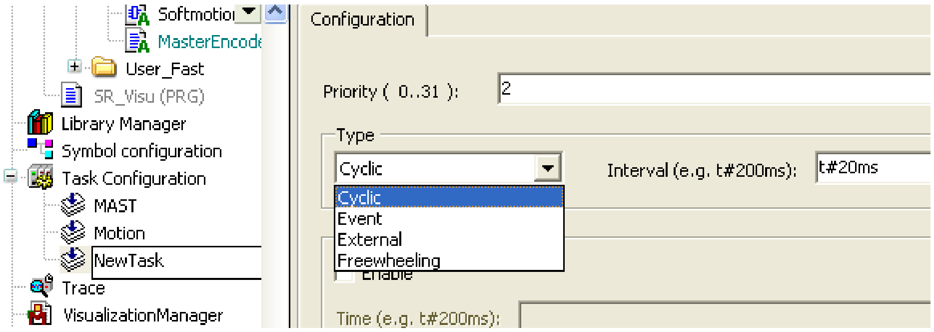
---

Tasks

A SoMachine application makes use of task-based programming. A task is configured to run cyclically in accordance with one of the following:

- a defined cycle time,
- in response to a periodic trigger,
- an event input, or
- Freewheeling

The task execution is also governed by priority as shown in the configuration options for the task, “NewTask”.



Priority (0..31): 2

Type: Cyclic

Interval (e.g. t#200ms): t#20ms

Typically, all tasks are configured as cyclic in order to provide a repeatable time base. Multiple tasks can be configured in the application, each with its own cycle time, and execution priority.

SoMachine creates the **Mast** (master) task as a default setting. A **Motion** task is created automatically with the addition of the CANmotion master.

The names or priority of the Mast and Motion task must NOT be changed.

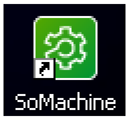
In the following example, an optional HMI_task has been created to process data for a relatively slow, periodic HMI update.

Task	**Motion**	**Mast**	**HMI_task**
Cycle time	4 msec (default setting)	20 msec (default setting)	250 msec (user-defined)
Program calls: (POUs)	SR_SoftMotion	SR_Main	SR_HMI

Exercise – Create a SoMachine Application

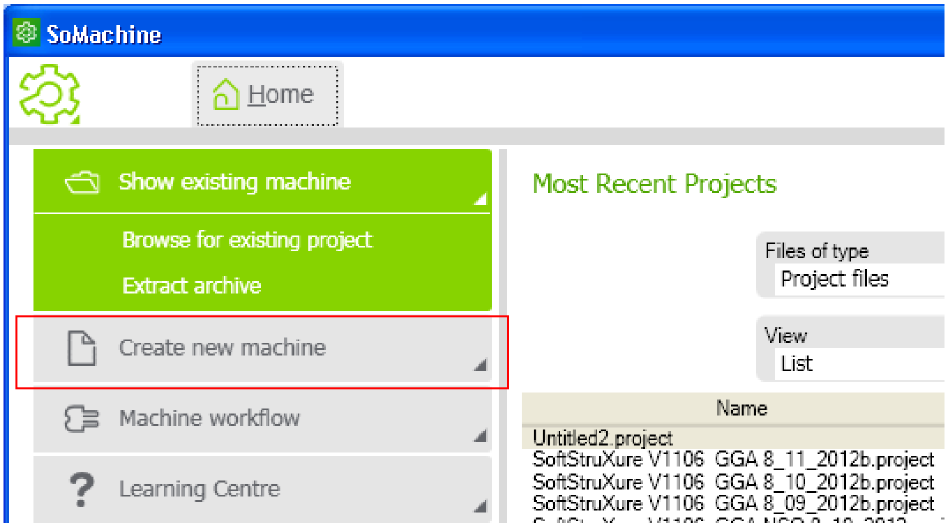
1. Launch SoMachine

i. From the Desktop Icon, launch the SoMachine application.



Alternately, select **All Programs >> Schneider Electric >> SoMachine** from the Start Menu

ii. At the SoMachine Home screen, Select **Create new machine**…

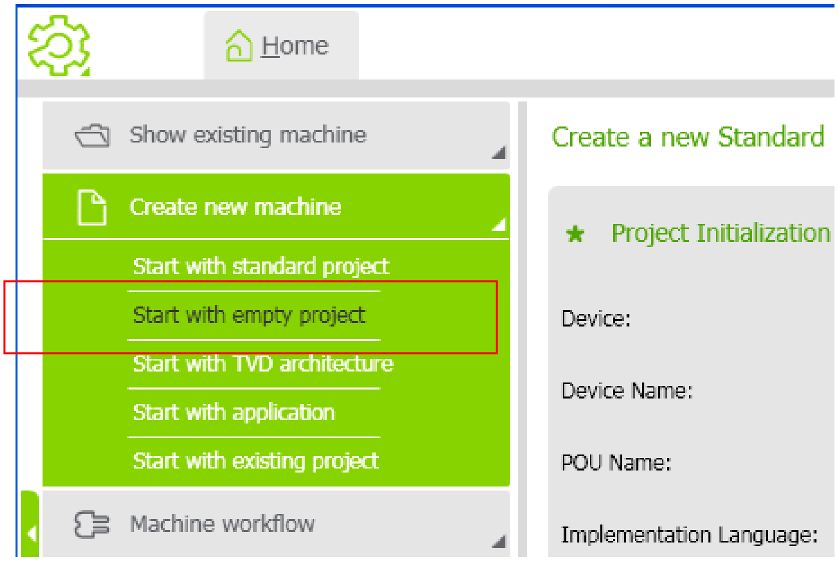


Files of type: Project files

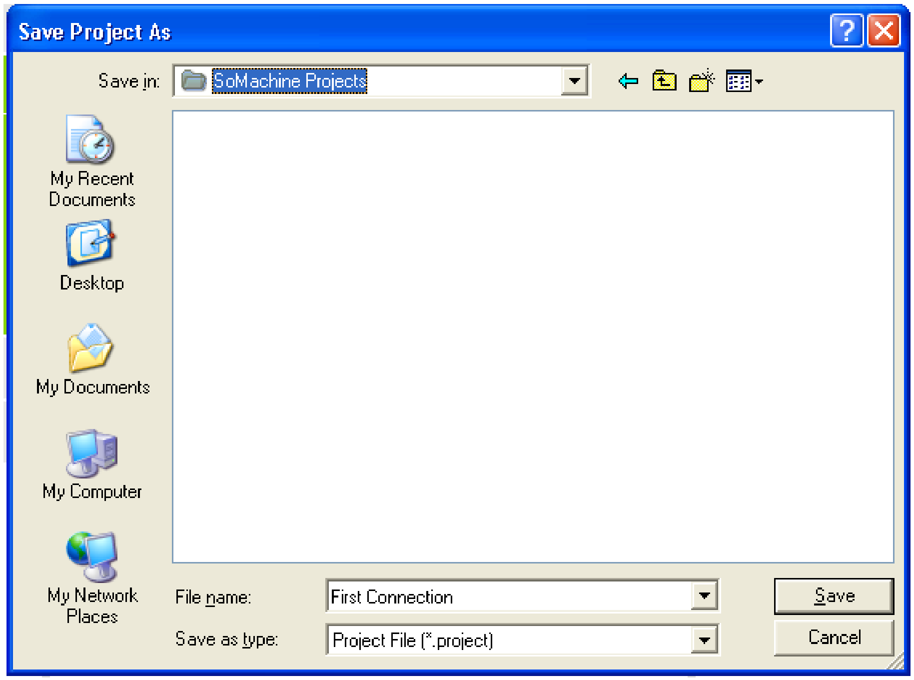
View: List

Untitled2.project
SoftStruXure V1106 GGA 8_11_2012b.project
SoftStruXure V1106 GGA 8_10_2012b.project
SoftStruXure V1106 GGA 8_09_2012b.project

iii. … then select **Start with empty project.**



iv. At the prompt, browse to the Desktop folder, **ACE University Motion with SoMachine >> MyProjects**, and save the project as **First Connection**.



Save in: SoMachine Projects

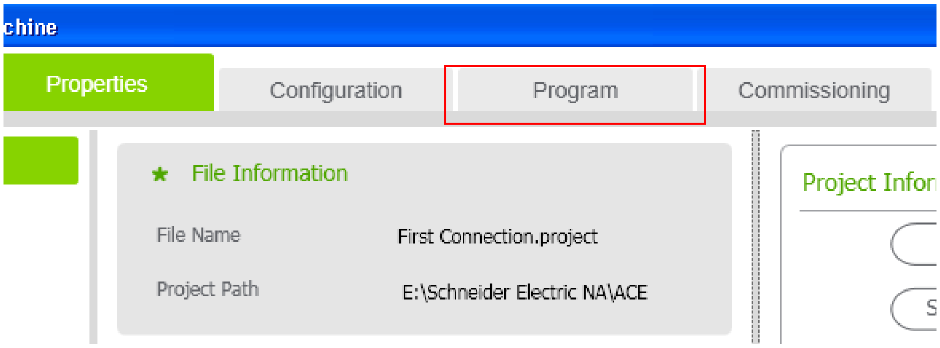
File name: First Connection

Save as type: Project File (*.project)

---

2. Add the Controller

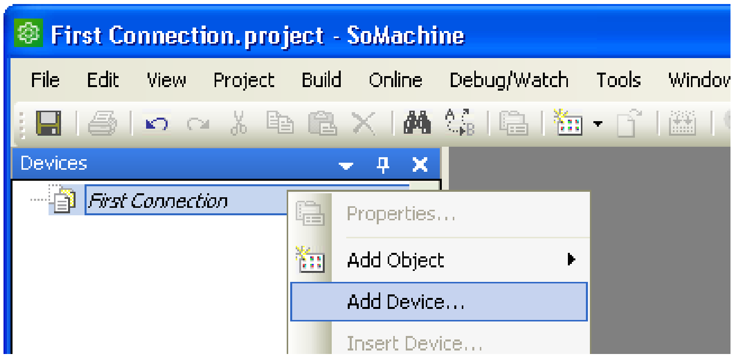
i. From the Project Navigator view, select the **Program** tab.



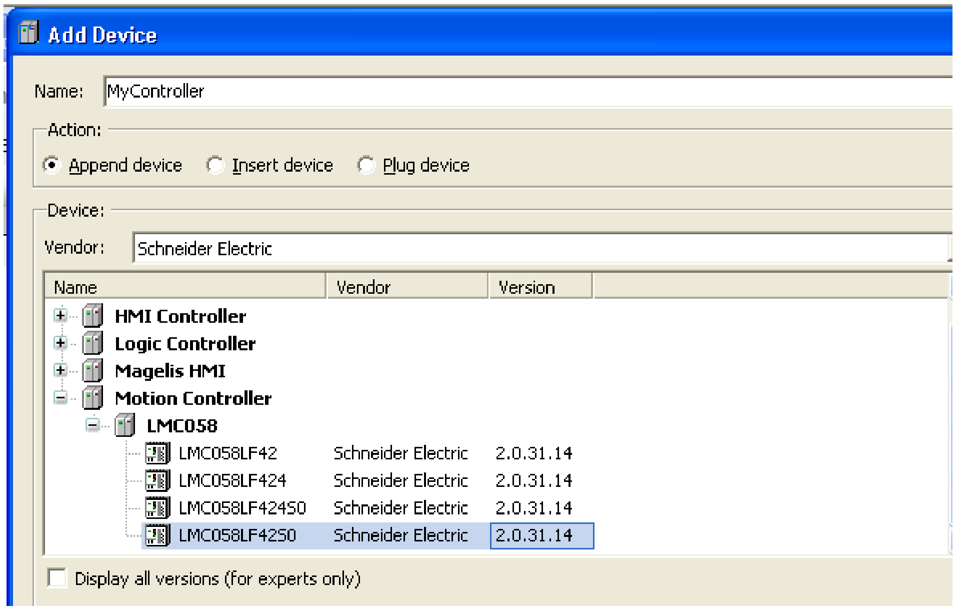
File Name First Connection.project

Project Path E:\Schneider Electric NA\ACE

ii. From the Project view, right-click on the Project name (*First Connection*), and select **Add Device…**



iii. From the device menu, select **Motion Controller >> LMC058F42S0**



Name: MyController

Action:

● Append device ○ Insert device ○ Plug device

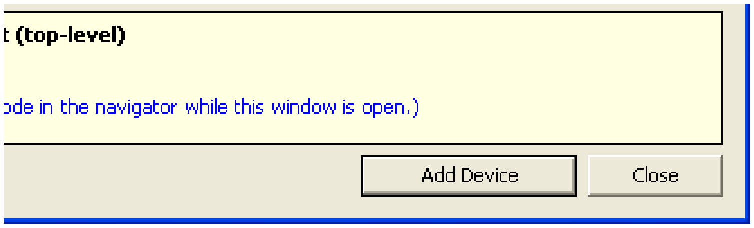
Device:

Vendor: Schneider Electric

Name	Vendor	Version
HMI Controller		
Logic Controller		
Magelis HMI		
Motion Controller		
LMC058		
LMC058LF42	Schneider Electric	2.0.31.14
LMC058LF424	Schneider Electric	2.0.31.14
LMC058LF424S0	Schneider Electric	2.0.31.14
LMC058LF42S0	Schneider Electric	2.0.31.14

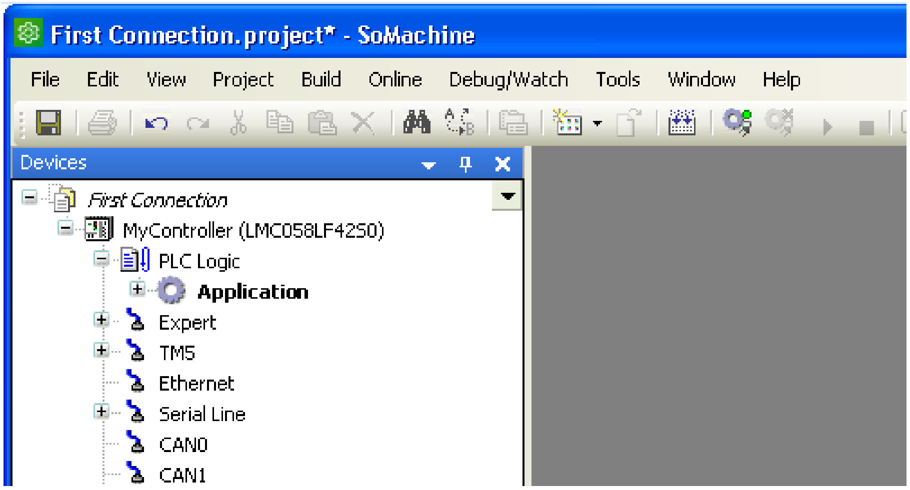
☐ Display all versions (for experts only)

iv. Click **Add Device** to continue.


t (top-level)

ode in the navigator while this window is open.)

At this time, SoMachine will create the application browser with the hardware and base libraries for the LMC058 controller.



- *First Connection*
  - MyController (LMC058LF42S0)
    - PLC Logic
      - **Application**
    - Expert
    - TM5
    - Ethernet
    - Serial Line
    - CAN0
    - CAN1

v. Close the Add Device menu at any time, and **Save** the project.

This concludes the Exercise

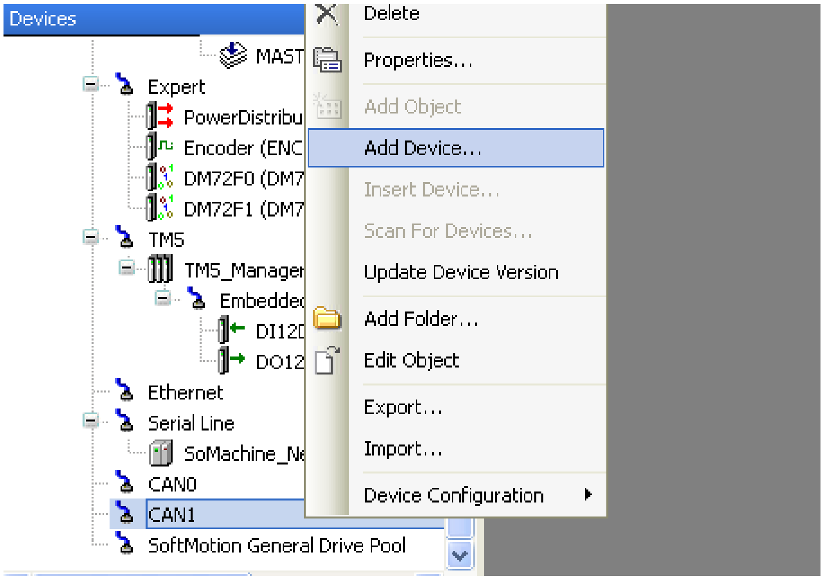
SoftMotion Axis – Mapping the Hardware

Adding a CANmotion device creates both a CANopen object and a subordinate SoftMotion object. The CANopen object manages basic CANopen communication properties including the device address and network health. The SoftMotion object manages the CANmotion-specific properties including axis type and scaling units for the path planner.

Configuration steps include:

1. Add a CANmotion master to the CAN1 port.
2. Add a CANmotion device to the CANmotion master.
3. Configure the CANopen device object
4. Configure the SoftMotion device object

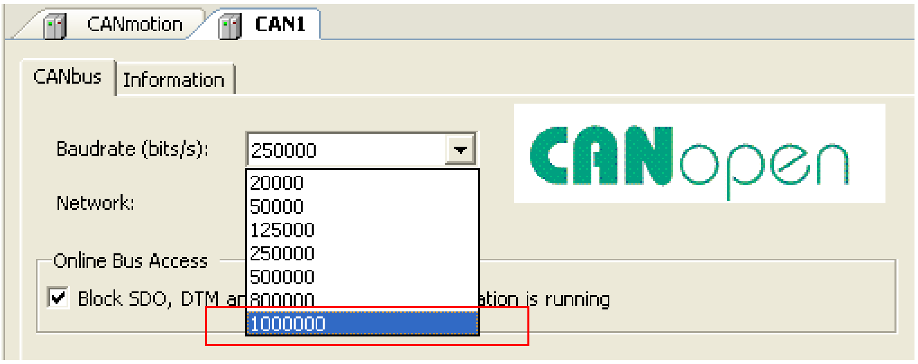
Devices are created in the hardware map by right-clicking on the browser object, and selecting **Add Device…** from the menu.



CAN1 Port

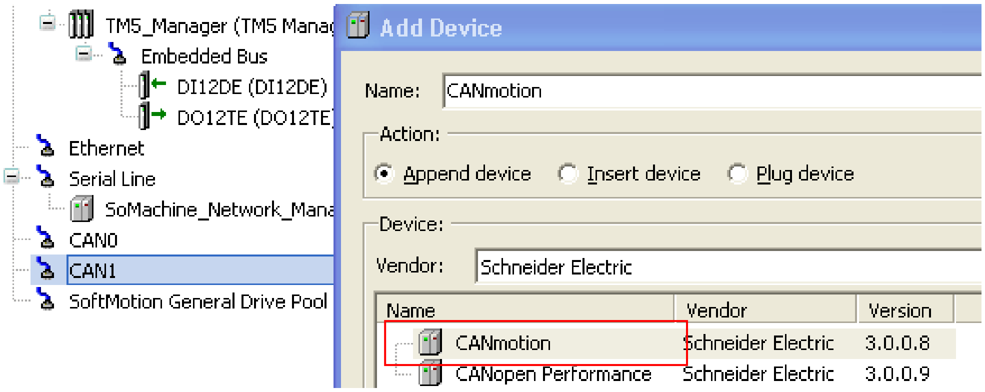
The communication baud rate for all slave devices is set at the CAN1 port. The configuration screen is accessed by double-clicking on the browser object.

Unless there is an unusually long network length, the baud-rate should always be set to 1Mb as shown.



CANmotion Master

A CANmotion master must be created at the CAN1 port.



Name: CANmotion

Action: ● Append device ○ Insert device ○ Plug device

Device:

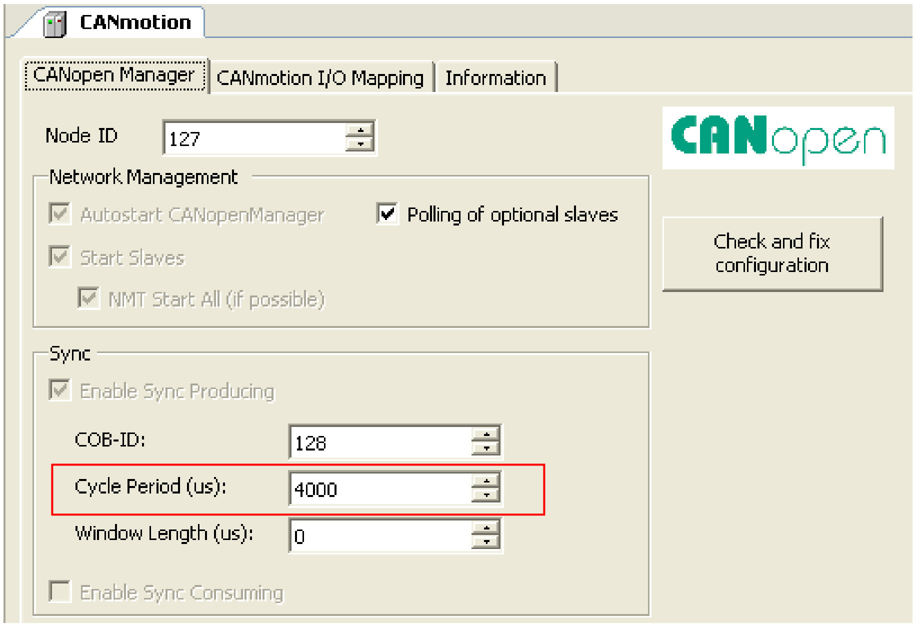
Vendor: Schneider Electric

Name	Vendor	Version
CANmotion	Schneider Electric	3.0.0.8
CANopen Performance	Schneider Electric	3.0.0.9

This object manages the CANmotion bus cycle time, RX PDO exchange, and NMT properties for the master.

CANmotion Cycle TIme

The configuration screen is accessed by double-clicking on the browser object. Typically, the only user interaction required for configuration is to set the CANmotion cycle time as shown



Node ID: 127

Network Management

☑ Autostart CANopenManager ☑ Polling of optional slaves

☑ Start Slaves

☑ NMT Start All (if possible)

Sync

☑ Enable Sync Producing

COB-ID: 128

Cycle Period (us): 4000

Window Length (us): 0

☐ Enable Sync Consuming

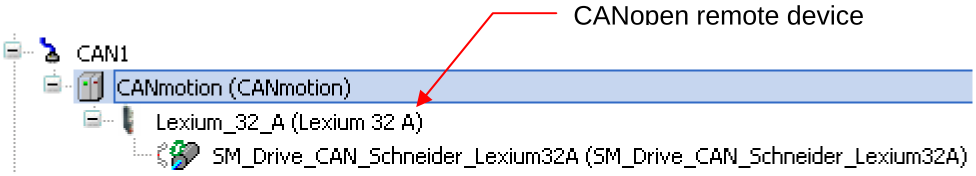
The default setting of 4000 usec can be used for many applications. However, the cycle time may have to be reduced, or extended based upon the number of devices

The cycle time should always be configured in multiples of 1000 usec.

CANmotion Axis - CANopen Device

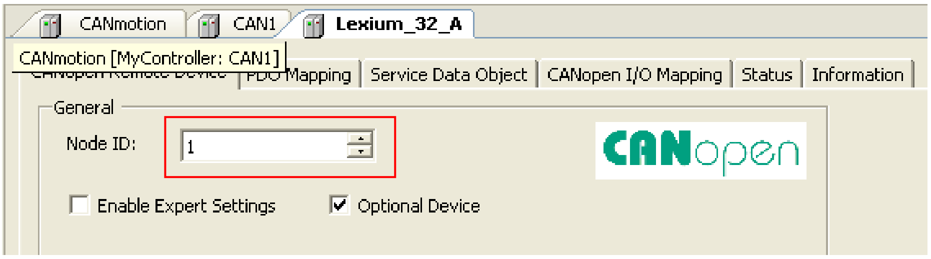
Adding a motion axis (Lexium32A) to the CANmotion master creates 2 objects:

- CANopen remote device (slave)
- SoftMotion Axis


CANopen remote device

CAN1
CANmotion (CANmotion)
Lexium_32_A (Lexium 32 A)
SM_Drive_CAN_Schneider_Lexium32A (SM_Drive_CAN_Schneider_Lexium32A)

The primary user requirement for the CANopen object is to set the device address.

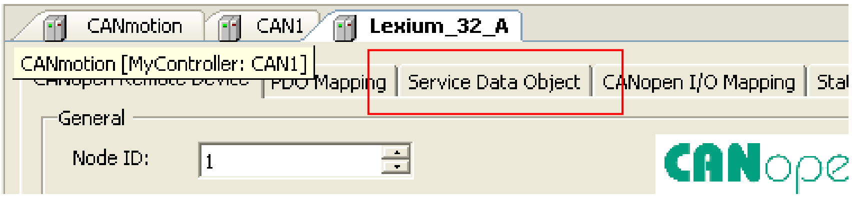


Node ID: 1

☐ Enable Expert Settings ☑ Optional Device

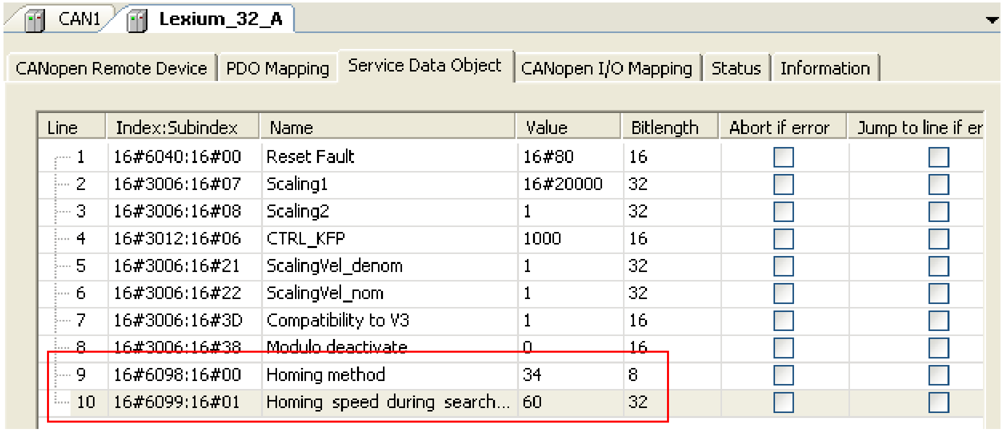
Service Data Objects

SoMachine provides a useful utility for device parameterization in the form of the **Service Data Object** (SDO) list.



Node ID: 1

CANopen parameters that are included in this list are automatically written to the slave device on startup. The SDO list contains a pre-configured set of parameters that are required to manage the CANmotion axis. Additional parameters, such as Homing method and search speed, can be included in the list to “commission” the axis for the application requirements



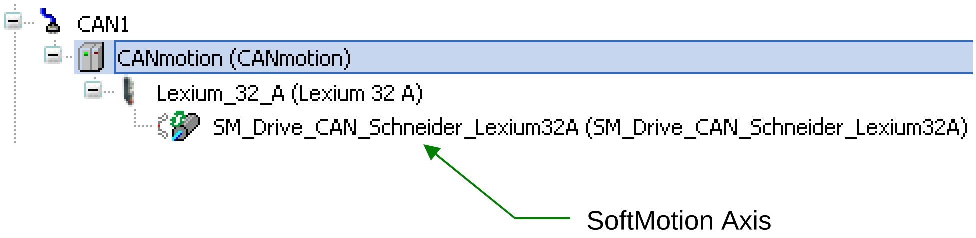
Line	Index:Subindex	Name	Value	Bitlength	Abort if error	Jump to line if er
1	16#6040:16#00	Reset Fault	16#80	16	☐	☐
2	16#3006:16#07	Scaling1	16#20000	32	☐	☐
3	16#3006:16#08	Scaling2	1	32	☐	☐
4	16#3012:16#06	CTRL_KFP	1000	16	☐	☐
5	16#3006:16#21	ScalingVel_denom	1	32	☐	☐
6	16#3006:16#22	ScalingVel_nom	1	32	☐	☐
7	16#3006:16#3D	Compatibility to V3	1	16	☐	☐
8	16#3006:16#38	Modulo deactivate	0	16	☐	☐
9	16#6098:16#00	Homing method	34	8	☐	☐
10	16#6099:16#01	Homing speed during search...	60	32	☐	☐

By adding all of the modified axis parameters (loop gains, hardware IO function, bus voltage, motor direction, EEPROM save, etc.) to the default list, it is possible to fully parameterize the drive without any use of commissioning software !!!

The functionality provided by the SDO list is particularly useful for configuring and demonstrating Fast Device Replacement (FDR).

SoftMotion Axis Object

A SoftMotion axis object is automatically created with any new CANmotion axis.



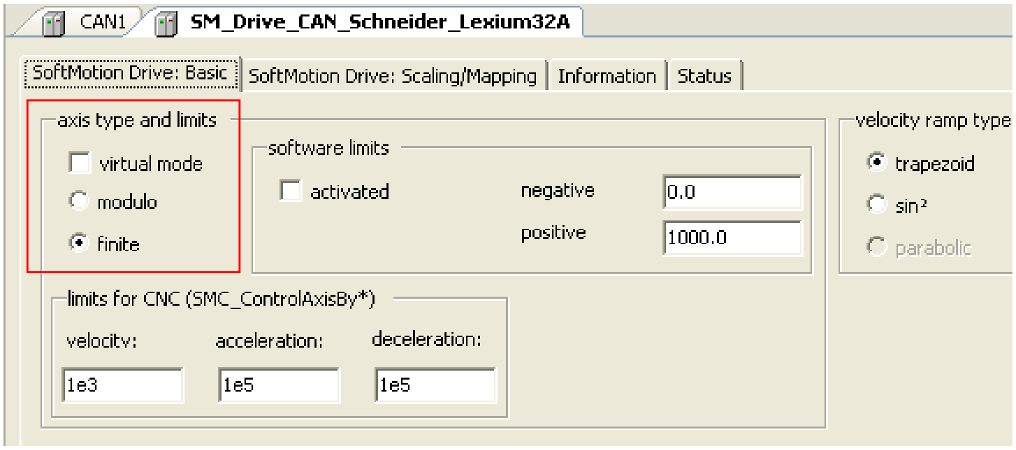
CAN1
CANmotion (CANmotion)
Lexium_32_A (Lexium 32 A)
SM_Drive_CAN_Schneider_Lexium32A (SM_Drive_CAN_Schneider_Lexium32A)

SoftMotion Axis

The primary user interaction for the SoftMotion object is to select the Axis type, and User engineering units for the axis. Double-click the SoftMotion browser object to access the configuration screen.

Axis Type

The Axis type is configured from the **SoftMotion Drive: Basic** tab. Here, software limits can also be activated, and defined if necessary. By selecting virtual mode, the axis becomes a mathematical model that does not exist on the CANmotion bus, although the remaining configuration parameters (modulo or finite, software limits, axis scaling, etc) still apply.



axis type and limits
☐ virtual mode
○ modulo
● finite

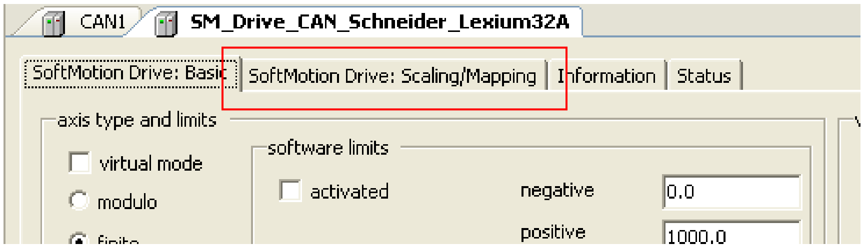
software limits
☐ activated
negative 0.0
positive 1000.0

velocity ramp type
● trapezoid
○ sin²
○ parabolic

limits for CNC (SMC_ControlAxisBy*)
velocity: 1e3
acceleration: 1e5
deceleration: 1e5

Scaling User Units

User units for the axis are configured within the Scaling and Mapping screen.

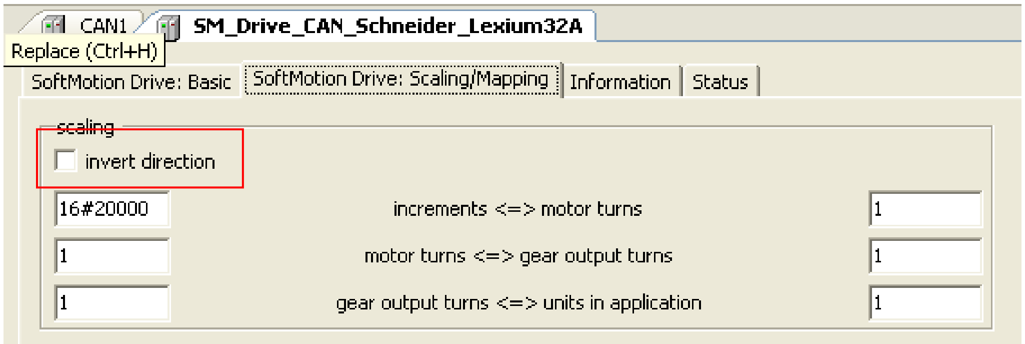


axis type and limits
☐ virtual mode
○ modulo
● finite

software limits
☐ activated
negative 0.0
positive 1000.0

Here, base drive units (increments) are converted into engineering units as specified by the user. Also, note the checkbox to invert the direction of motor rotation.

By standard convention, “positive” rotation is defined as clockwise when looking at the motor shaft.



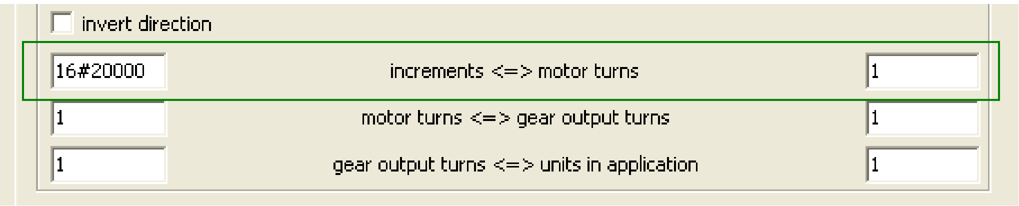
☐ invert direction

16#20000	increments <=> motor turns	1
1	motor turns <=> gear output turns	1
1	gear output turns <=> units in application	1

To understand the **Scaling/Mapping** panel shown above, it is convenient to think about the entry fields as three rows of data… namely, Top, Middle, and Bottom.

The Top row defines the number of drive increments that correspond to a single motor shaft rotation.

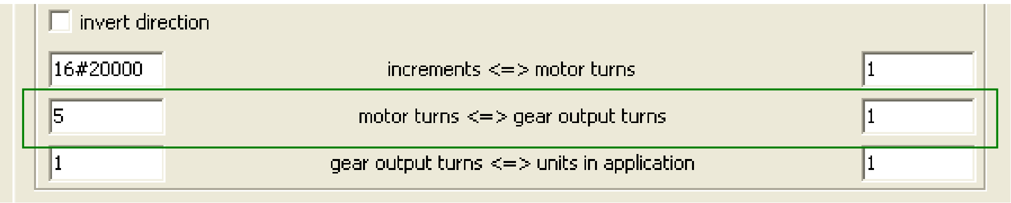
> *The parameters in this row are entered automatically, and should not be changed !*



☐ invert direction

16#20000	increments <=> motor turns	1
1	motor turns <=> gear output turns	1
1	gear output turns <=> units in application	1

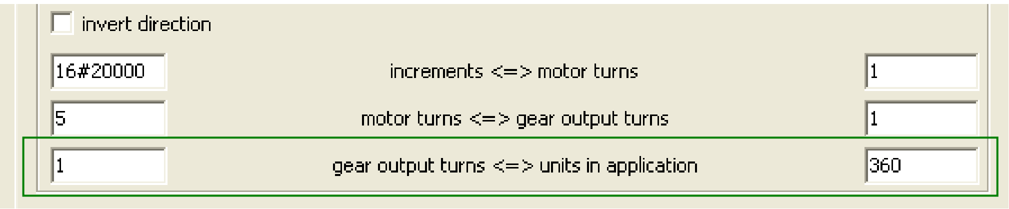
The middle row is designed to accommodate a gear reducer such as planetary gearbox, or timing pulleys. For example, if the power train makes use of a 5:1 gearbox, the corresponding entry is this row is shown as follows.



☐ invert direction

16#20000	increments <=> motor turns	1
5	motor turns <=> gear output turns	1
1	gear output turns <=> units in application	1

Finally, the bottom provides the input for the load mechanism and the conversion to user (engineering) units. In the example shown, the gearbox output is connected to a rotary load with user units of degrees.



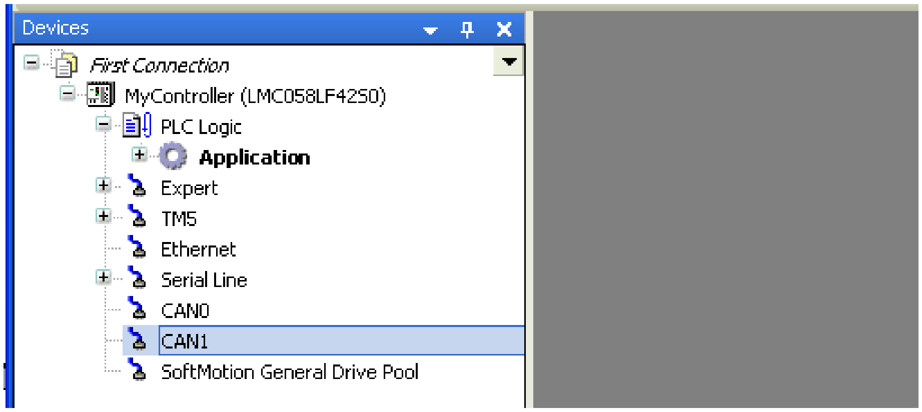
☐ invert direction

16#20000	increments <=> motor turns	1
5	motor turns <=> gear output turns	1
1	gear output turns <=> units in application	360

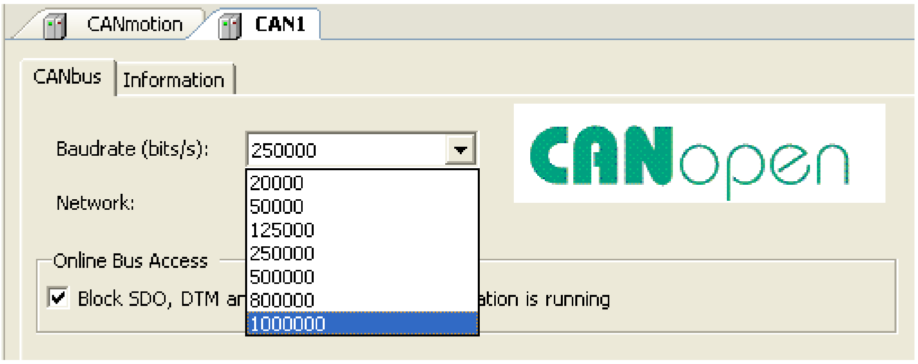
Exercise – Mapping an Axis on CANmotion

1. Configure the CAN1 port

i. From the First Connection browser, Double-click the **CAN1** port to open the port configuration panel



ii. In the configuration panel, set the CANbus baud rate to 1Mb (1000000).



Baudrate (bits/s): 250000

Network:

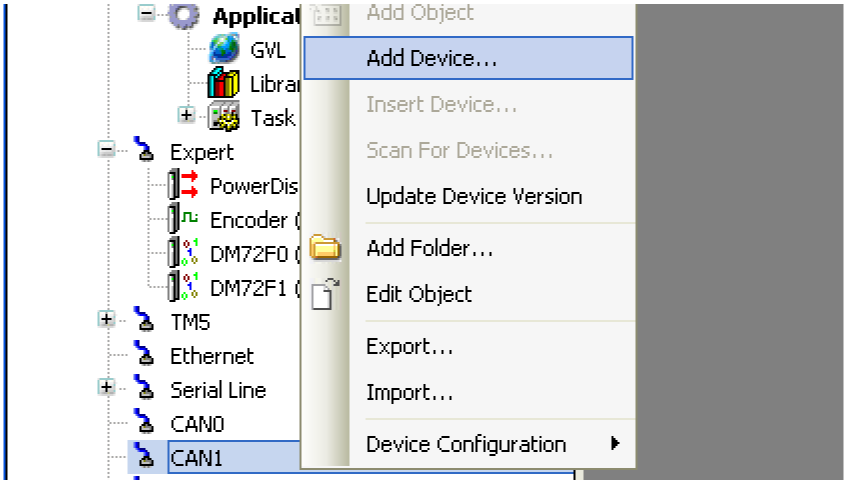
Online Bus Access

☑ Block SDO, DTM an… …ation is running

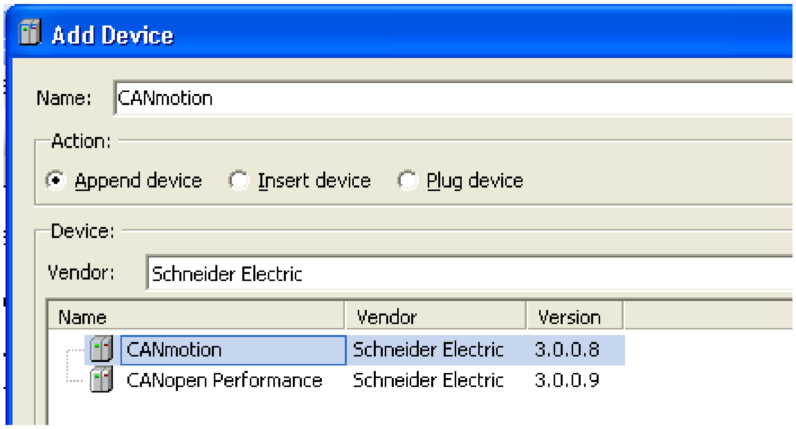
---

2. Add the CANmotion master

i. Right-click on the CAN1 port and select **Add device**.



ii. Select the **CANmotion** master. Double-click, or select Add device as before, to add the master object.



Name: CANmotion

Action:

● Append device ○ Insert device ○ Plug device

Device:

Vendor: Schneider Electric

Name	Vendor	Version
CANmotion	Schneider Electric	3.0.0.8
CANopen Performance	Schneider Electric	3.0.0.9

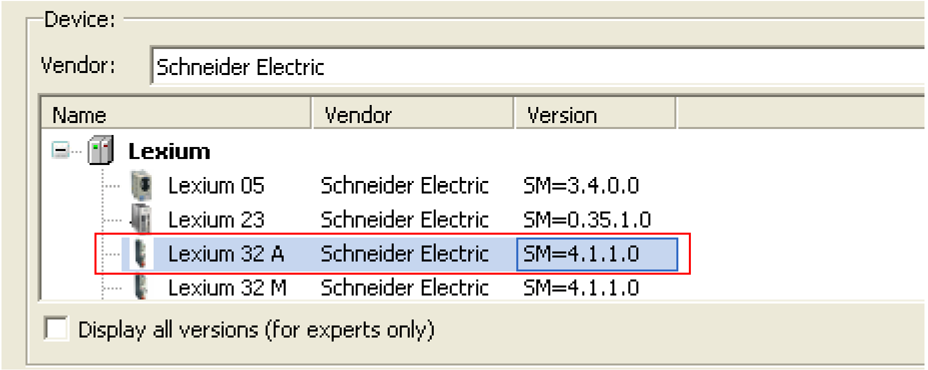
This will create a CANmotion master object within the CAN1 port.

---

3. *Add the Lexium32 Axis*

i. Right-click the CANmotion master, and select Add device as before.

ii. From the device list, select Lexium32A,.



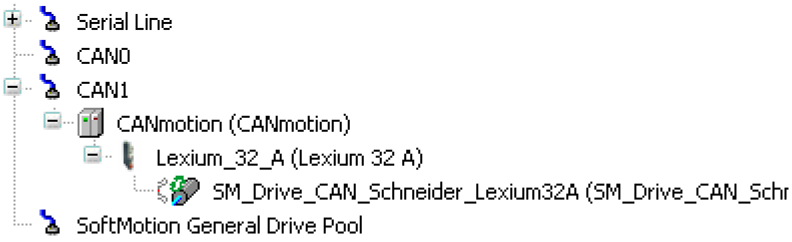
Device:

Vendor: Schneider Electric

Name	Vendor	Version
Lexium		
Lexium 05	Schneider Electric	SM=3.4.0.0
Lexium 23	Schneider Electric	SM=0.35.1.0
Lexium 32 A	Schneider Electric	SM=4.1.1.0
Lexium 32 M	Schneider Electric	SM=4.1.1.0

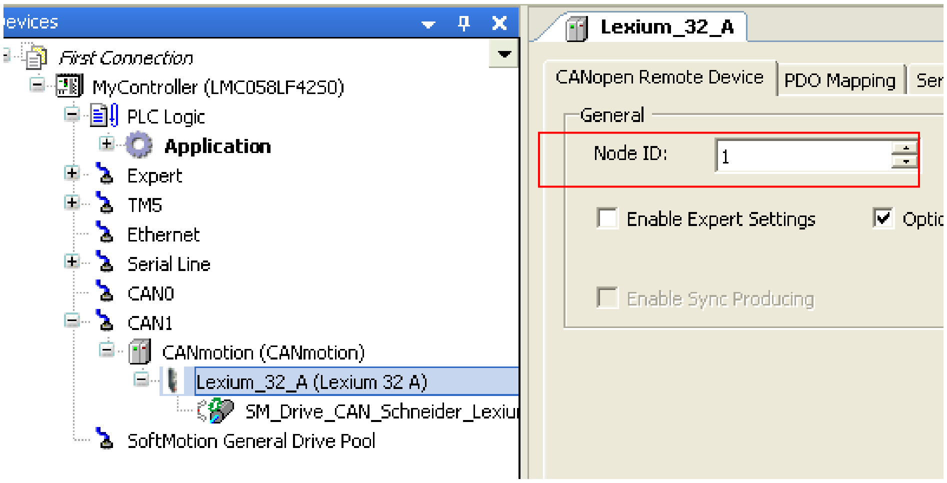
☐ Display all versions (for experts only)

A CANopen remote device, and SoftMotion axis are added to the browser as shown



```
Serial Line
CAN0
CAN1
    CANmotion (CANmotion)
        Lexium_32_A (Lexium 32 A)
            SM_Drive_CAN_Schneider_Lexium32A (SM_Drive_CAN_Schr
SoftMotion General Drive Pool
```

iii. Double-click the CANopen slave object, and select address (Node ID) 1 for the drive.



Node ID: 1

☐ Enable Expert Settings ☑ Optic

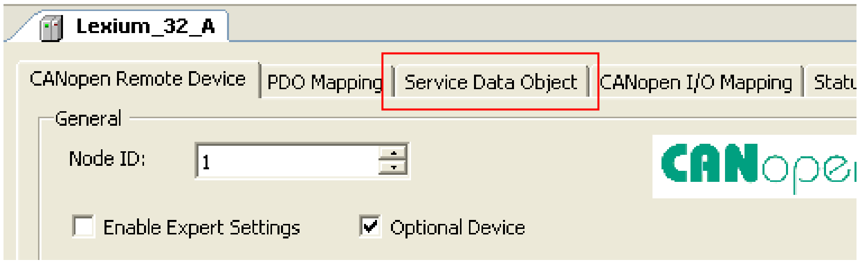
☐ Enable Sync Producing

---

4. Configure the CANopen SDO list

We will use the SDO list to pre-configure the axis with a specific homing type and IO hardware configuration.

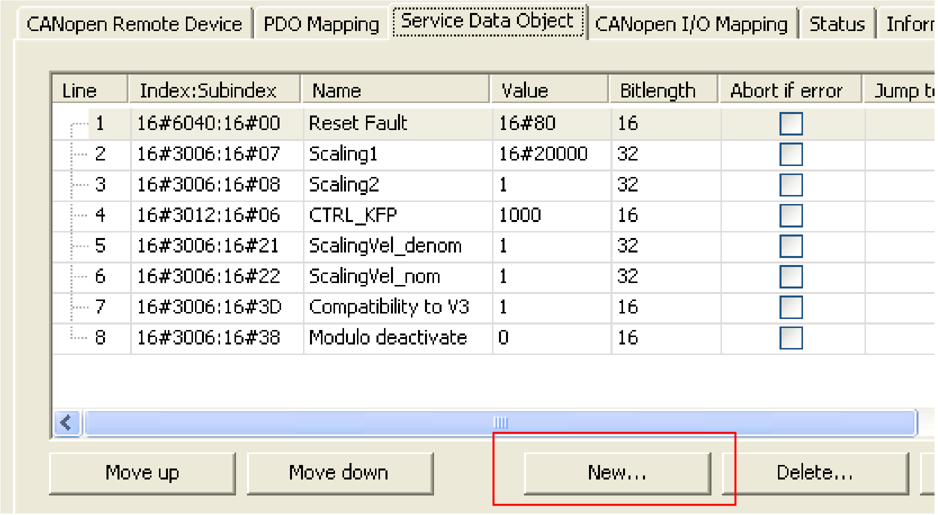
i. From the CANopen device configuration menu, select the **Service Data Object** tab.



Node ID: 1

☐ Enable Expert Settings ☑ Optional Device

ii. From the SDO menu, select **New…**



Line	Index:Subindex	Name	Value	Bitlength	Abort if error	Jump t
1	16#6040:16#00	Reset Fault	16#80	16	☐	
2	16#3006:16#07	Scaling1	16#20000	32	☐	
3	16#3006:16#08	Scaling2	1	32	☐	
4	16#3012:16#06	CTRL_KFP	1000	16	☐	
5	16#3006:16#21	ScalingVel_denom	1	32	☐	
6	16#3006:16#22	ScalingVel_nom	1	32	☐	
7	16#3006:16#3D	Compatibility to V3	1	16	☐	
8	16#3006:16#38	Modulo deactivate	0	16	☐	

iii. From the picklist, add the following parameters:

CANopen index [hex]	Subindex [hex]	Parameter	Value [dec]
6098	00	Homing method	34
6099	00	Homing speed	60
3077	01	IOfunction_DI0	1
3077	02	IOfunction_DI1	1
3077	03	IOfunction_DI2	1
3077	04	IOfunction_DI3	1

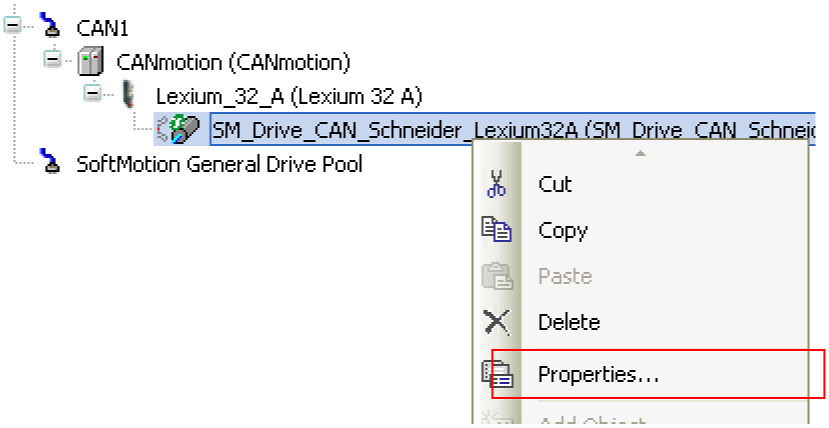
These parameters will set the :

- homing method to "positive search for index"
- homing search speed 60 RPM
- IO function "freely available" for all inputs

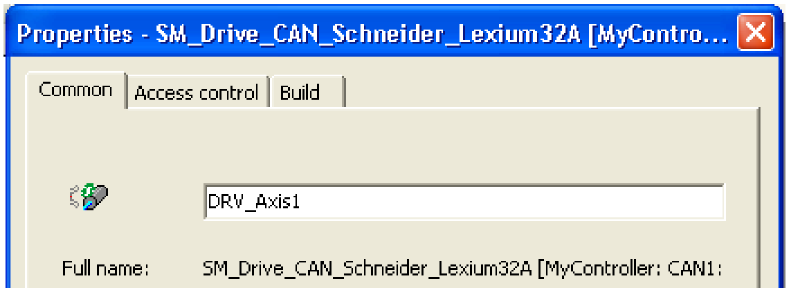
---

5. Configure Axis SoftMotion Parameters

i. Right-click on the SoftMotion object, select **Properties**…



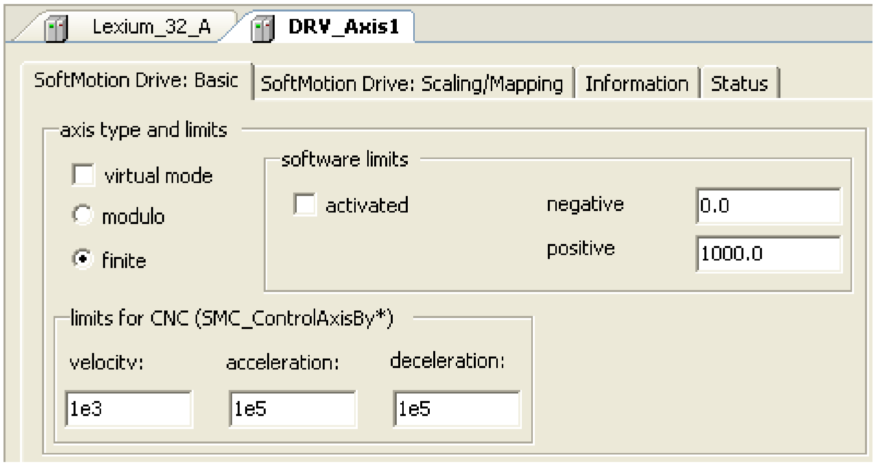
ii. … and rename the axis **DRV_Axis1**. Select **OK** to continue.



DRV_Axis1

Full name: SM_Drive_CAN_Schneider_Lexium32A [MyController: CAN1:

iii. Double-click the SoftMotion Axis to open **the SoftMotion Drive : Basic** screen. Accept the default settings for the Axis type as shown.



axis type and limits

☐ virtual mode
○ modulo
● finite

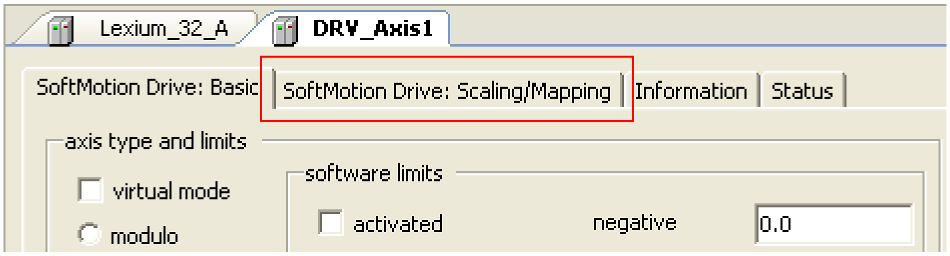
software limits

☐ activated
negative 0.0
positive 1000.0

limits for CNC (SMC_ControlAxisBy*)

velocity:	acceleration:	deceleration:
1e3	1e5	1e5

iv. Select the **Scaling/Mapping** tab.



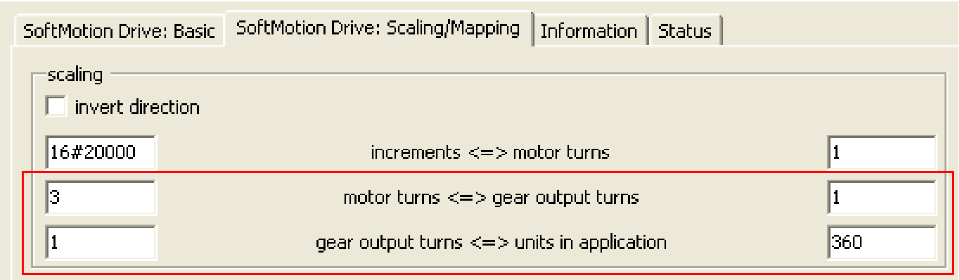
axis type and limits

☐ virtual mode
○ modulo

software limits

☐ activated
negative 0.0

v. Modify the Scaling fields to create an axis with 3:1 gearbox using degrees as the user units.



scaling

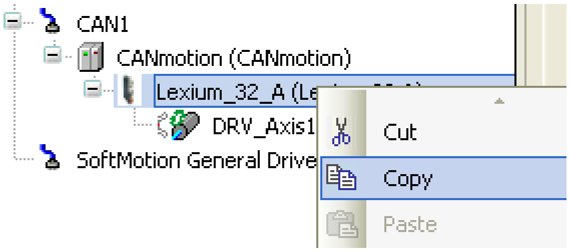
☐ invert direction

16#20000	increments <=> motor turns	1
3	motor turns <=> gear output turns	1
1	gear output turns <=> units in application	360

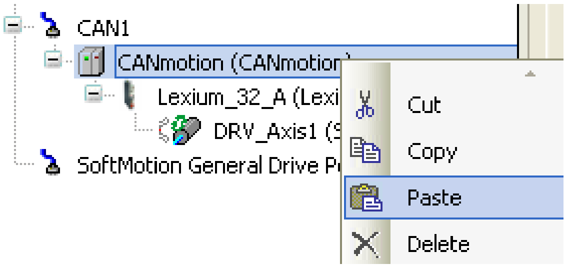
---

6. Copy and Paste an additional Axis

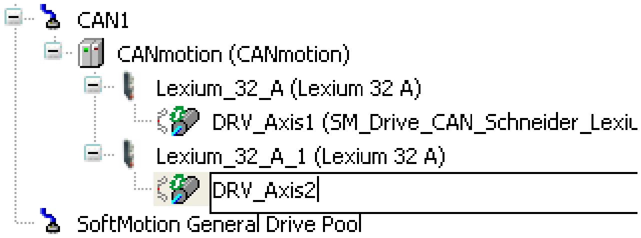
i. Create an identical axis by copying the CANopen drive object from the application browser.



ii. Paste onto the CANmotion master to create a copy.



iii. Change the name of the new axis to **DRV_Axis2**.

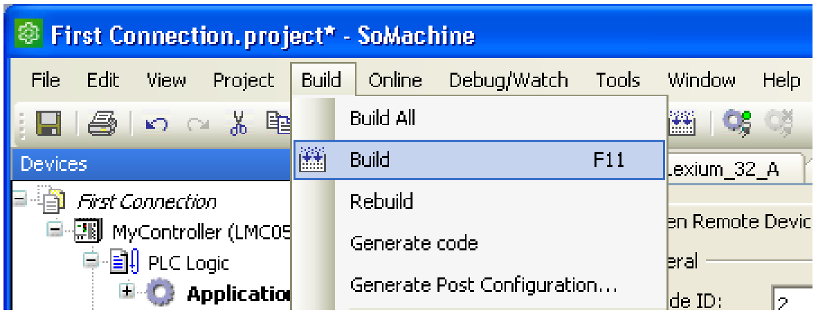


iv Finally, Double-click the CANopen object “Lexium_32_A_1”, and change the device address (Node ID) to 2.

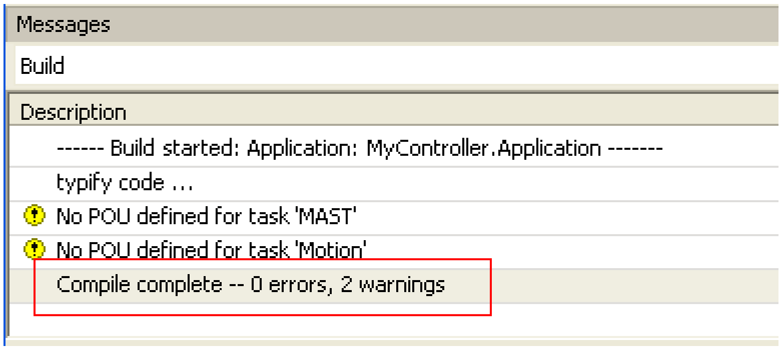
---

7. Build and Save

i. Build the Application using the top level menu



ii. Make sure there are no build errors….



```
------ Build started: Application: MyController.Application -------
typify code ...
No POU defined for task 'MAST'
No POU defined for task 'Motion'
Compile complete -- 0 errors, 2 warnings
```

iii. And **Save** the project as ***First Hardware Map***.

---

8. On your own …

i. Copy and Paste to create a Virtual Master axis.

ii. Assign the node address **9**

iii. Rename the SoftMotion axis **DRV_Master**.

iv. Scale the axis for 360 degrees to correspond to a single motor rotation.

v. Set the Axis type to **Modulo**.

vi. **Build** and **Save** as before.

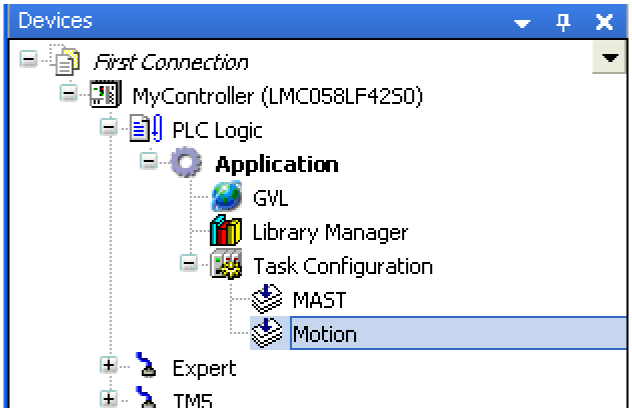
This concludes the exercise

Motion Control – Mapping the Functionality

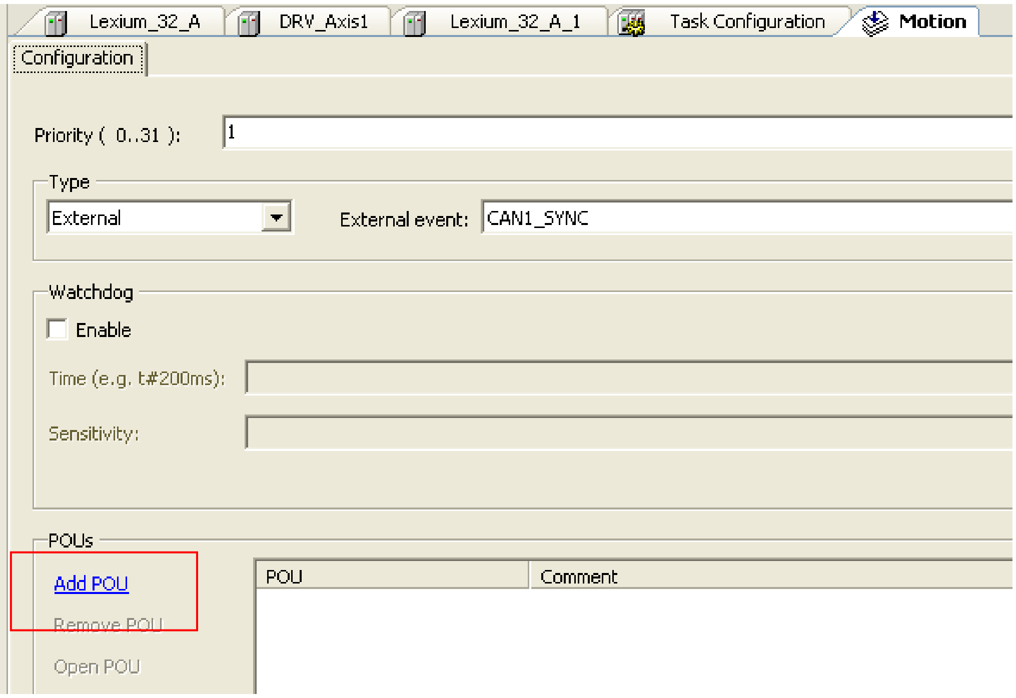
CANmotion provides an efficient means of managing independent and synchronized axis functionality in conformance with the PLCopen standard. SoMachine software implements the standard using an extensive library running on the CoDeSys SoftMotion engine. In this chapter, we will review the basic requirements of the SofMotion and the PLCopen standard.

Task Calls

The SoftMotion library and Motion task are automatically created with the addition of a CAN motion master. A requirement of SoftMotion is that the associated PLCopen Function blocks are instantiated within a Program Organizational Unit (POU) called from the Motion task.



Once created, a POU can be associated with (added to) a task call from the configuration screen for the specific task, as shown below for the Motion task.



Priority (0..31): 1

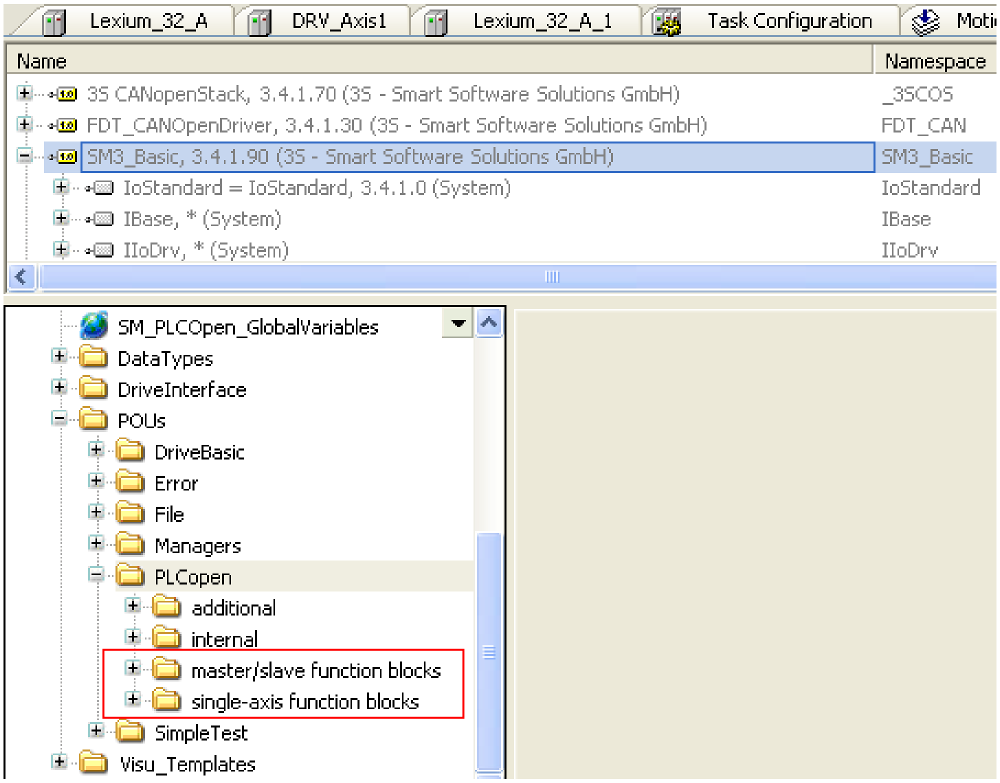
Type: External

External event: CAN1_SYNC

☐ Enable

SoftMotion

SoftMotion is the run-time engine for motion control in the SoMachine environment. Located within the **SM3_Basic** library as shown, SoftMotion provides an extensive collection of administrative and movement control function blocks that confirm to the international PLCopen standard.



Name	Namespace
3S CANopenStack, 3.4.1.70 (3S - Smart Software Solutions GmbH)	_3SCOS
FDT_CANOpenDriver, 3.4.1.30 (3S - Smart Software Solutions GmbH)	FDT_CAN
SM3_Basic, 3.4.1.90 (3S - Smart Software Solutions GmbH)	SM3_Basic
IoStandard = IoStandard, 3.4.1.0 (System)	IoStandard
IBase, * (System)	IBase
IIoDrv, * (System)	IIoDrv

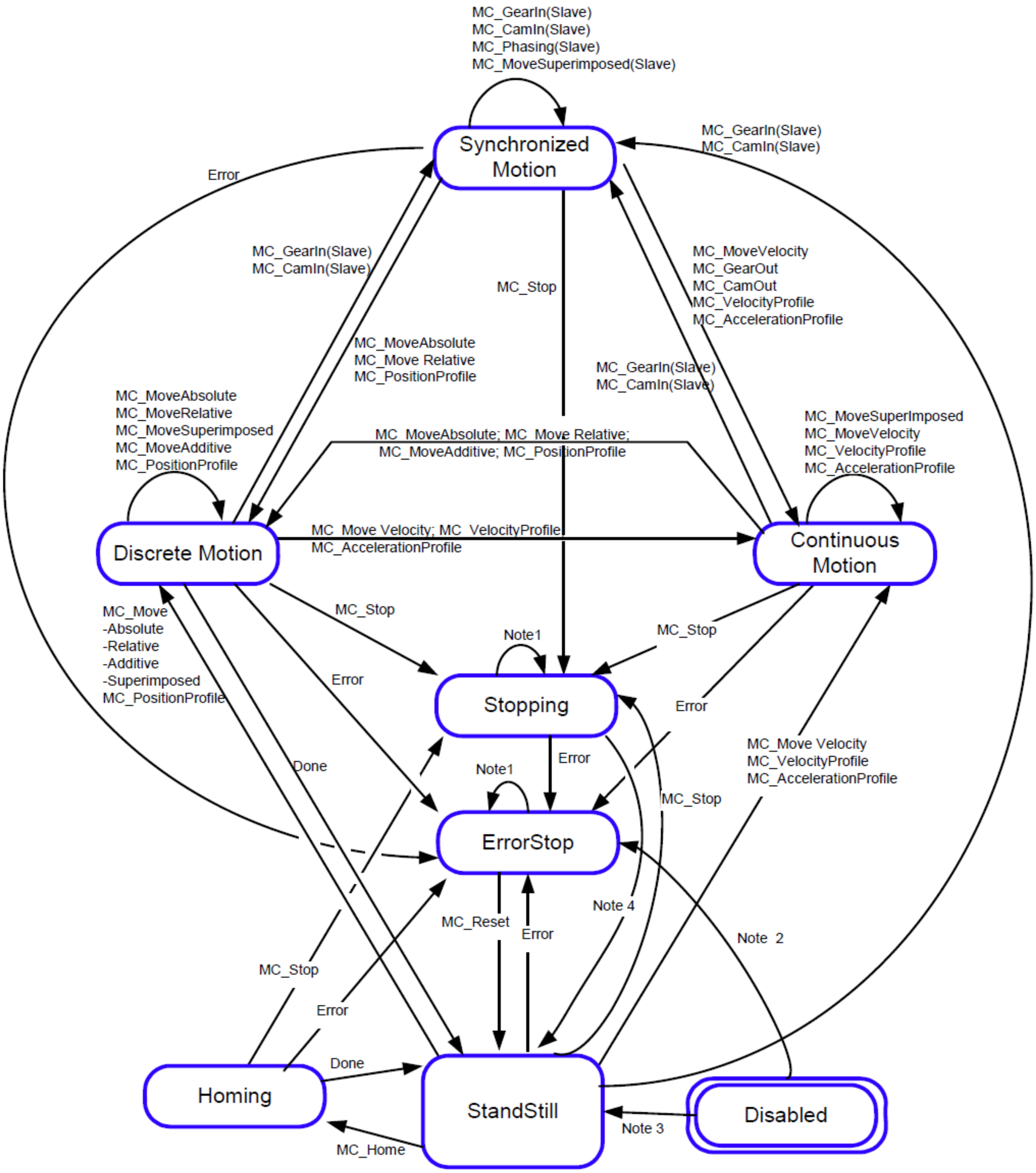
- SM_PLCOpen_GlobalVariables
- DataTypes
- DriveInterface
- POUs
  - DriveBasic
  - Error
  - File
  - Managers
  - PLCopen
    - additional
    - internal
    - master/slave function blocks
    - single-axis function blocks
  - SimpleTest
- Visu_Templates

A few of the function blocks required for nearly any motion application include:

Function Block	Description
MC_Power	Enables the Axis position control loop
MC_Reset	Resets an active asynchronous or synchronous alarm
MC_ReadAxisError	Retrieves an active asynchronous drive alarm
MC_SetPosition	Performs a machine reference by moving the machine origin
MC_Home	Performs a reference by moving the axis
MC_MoveVelocity	Initiates a continuous movement at command velocity
MC_MoveAbsolute	Initiates an absolute PTP movement
MC_MoveRelative	Initiate a relative PTP movement
MC_Jog	Performs an axis Jog (forward or reverse)
MC_Stop	Stops all active movement with a predefined deceleration rate

The PLCopen State Diagram

Effective and efficient use of these functions requires strict adherence to the PLCopen state diagram. In accordance with the PLCopen standard, a servo axis always exists on one of 8 possible states. The State Diagram provides a graphical map of the Axis states and the possible transitions between them.



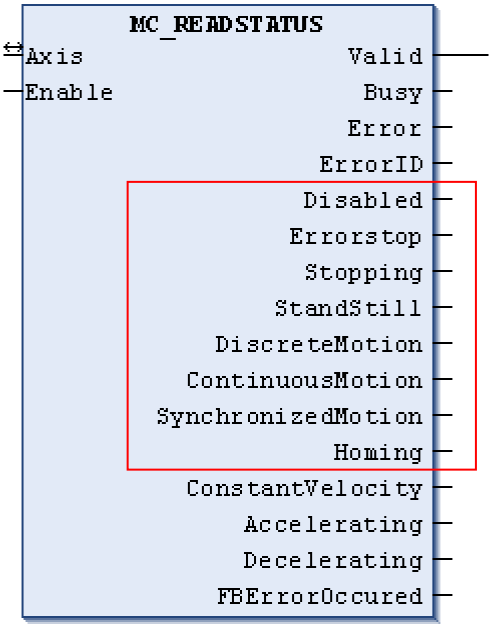
A basic rule of thumb in the development of any motion-centric program is to “Confirm… then Command”

- ➢ Confirm that the axis is in the appropriate PLCopen state.
- ➢ Command the axis to initiate the required motion function.

MC_ ReadAxisStatus

A SoftMotion function, **MC_ReadAxisState**, to retrieve the current PLCopen state of an axis. MC_ReadAxisState should ALWAYS be instantiated as the primary SoftMotion function block for any configured axis in the application.

The PLCopen state outputs for MC_ReadStatus are indicated below. The FB also provides additional movement information as obtained from the Device statusword.



Two of the most important axis states are :

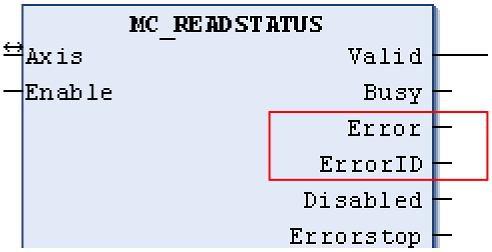
- **Standstill**, and
- **ErrorStop**

Standstill is generally the starting point state for any movement control. It indicates that the Axis has an applied DC bus, STO inputs are active, and the drive is enabled and ready for movement.

Errorstop indicates either an asynchronous drive alarm, or a synchronous FB alarm. In the event of a synchronous FB alarm, there may be no indication on the drive itself that there is an alarm condition.

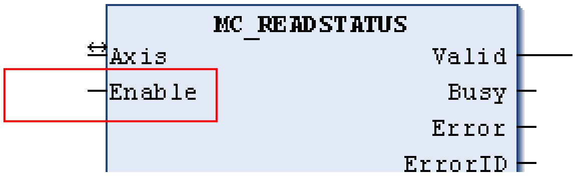
PLCopen - General Characteristics

The PLCopen standard defines the behavior of the function as well as the required administrative inputs and outputs. Every PLCopen function block includes some form of synchronous message status as an output. This error is related only to the FB message, not the axis itself.



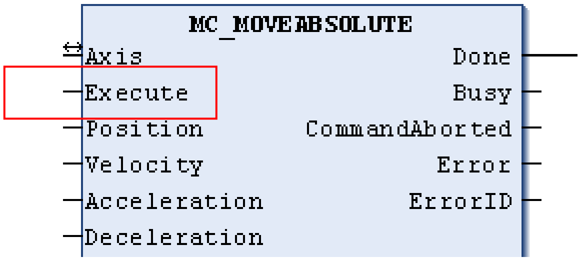
Input Execution Types

Most, if not all, PLCopen FBs will have either an **Enable** or **Execute** input to “trigger” the functionality. An Enable input is “level-based”, which continues the FB action as long as the Input is applied.



Care must be taken to avoid a continuous active Enable input on certain functions, such as ***MC_ReadAxisError****. In this case, the input will continuously perform an SDO read parameter message, and significantly burden the CAN network.*

Movement-based functions are generally “rising edge-triggered” and are identified by an Execute input.

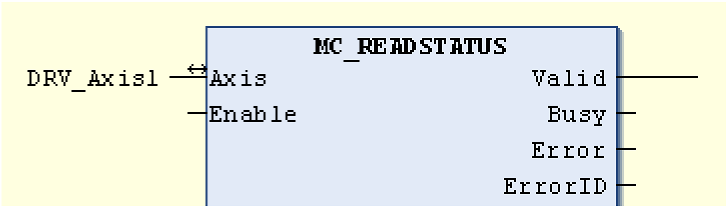


A rising edge input is required to update parameters on-the-fly, or to trigger a secondary movement upon completion of the first.

For a detailed listing of available PLCopen standard function blocks and behavior, please reference documentation from the PLCopen organization.

Axis_Ref

The Input Output object “Axis” establishes the communication path to the correct Axis, and must be assigned an **Axis_Ref_SM3** data type. The Axis_Ref_SM3 assignment can either be the Softmotion axis created during the CANmotion axis configuration, or a pointer to the SoftMotion Axis.



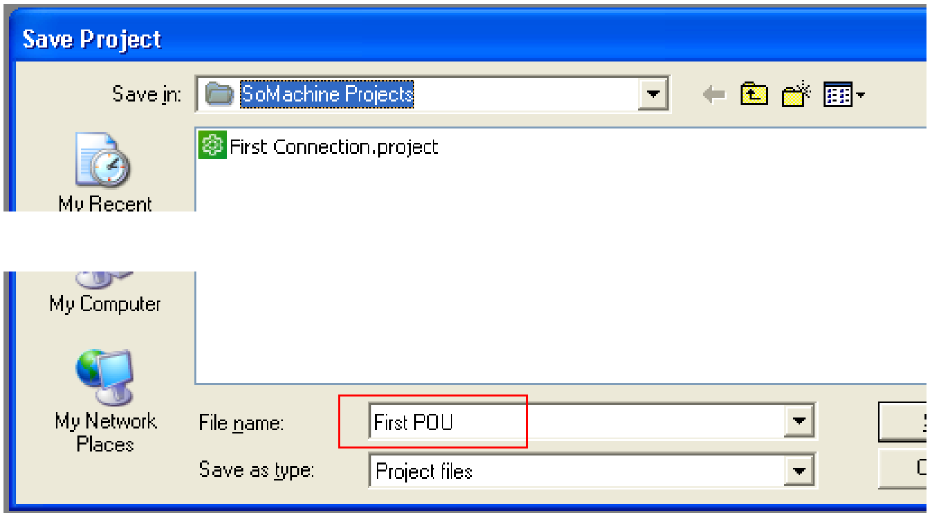
In the next Exercise, we will :

- Create a POU for Motion control
- Instantiate PLCopen FBs for Axis Control
- Associate the POU with the Motion task
- Monitor and Control the axis using online system variables

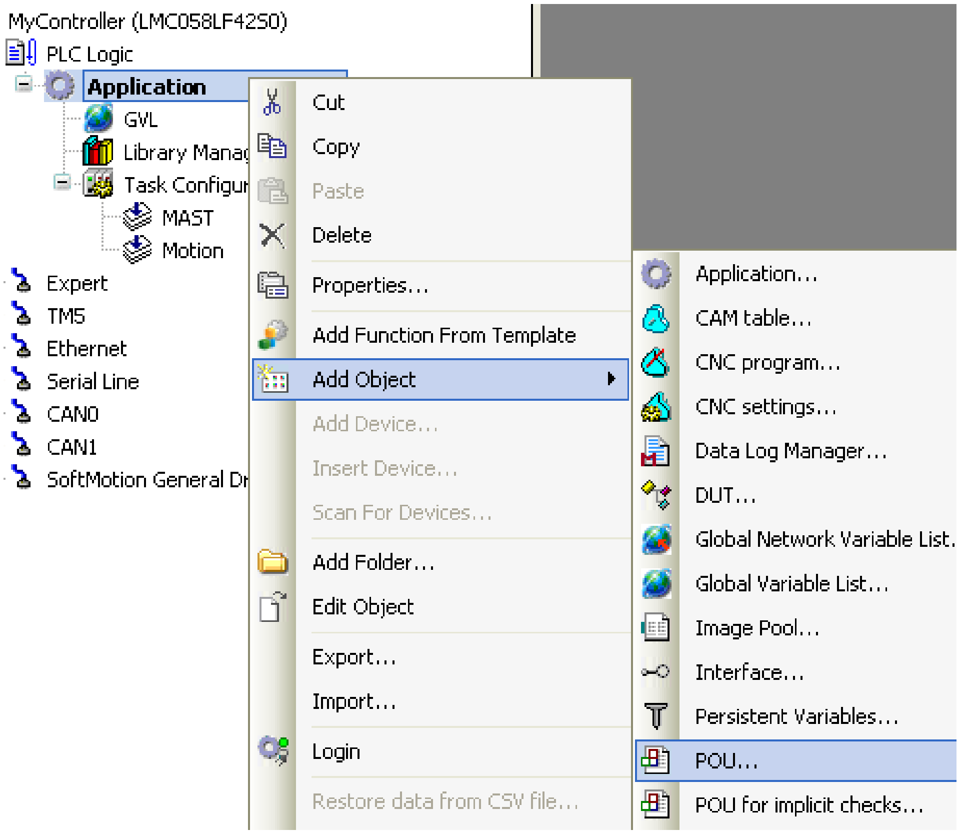
Exercise – Create an Axis Control POU

1. Create a Motion POU called SR_SoftMotion

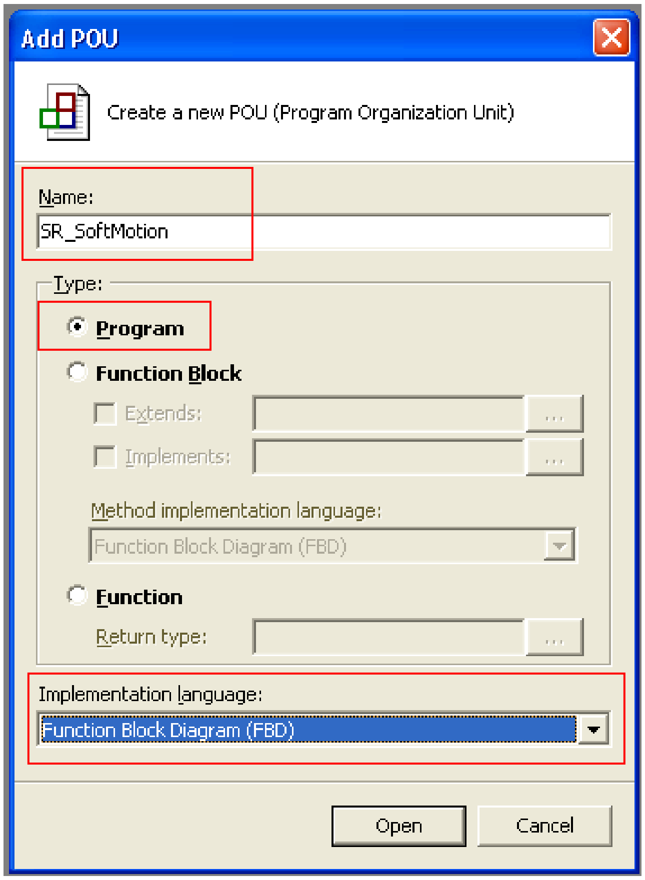
i. Open the existing project (First Hardware Map), and **Save** the project in the training folder as "**First POU**".



ii. From the SoMachine browser, right-click on the **Application** object, and select **Add Object >> POU…**



iii. Configure the POU type as ***Program***, using the language **FBD**. Name the POU **SR_SoftMotion** as shown.



Add POU

Create a new POU (Program Organization Unit)

Name: SR_SoftMotion

Type:

● Program

○ Function Block

☐ Extends:

☐ Implements:

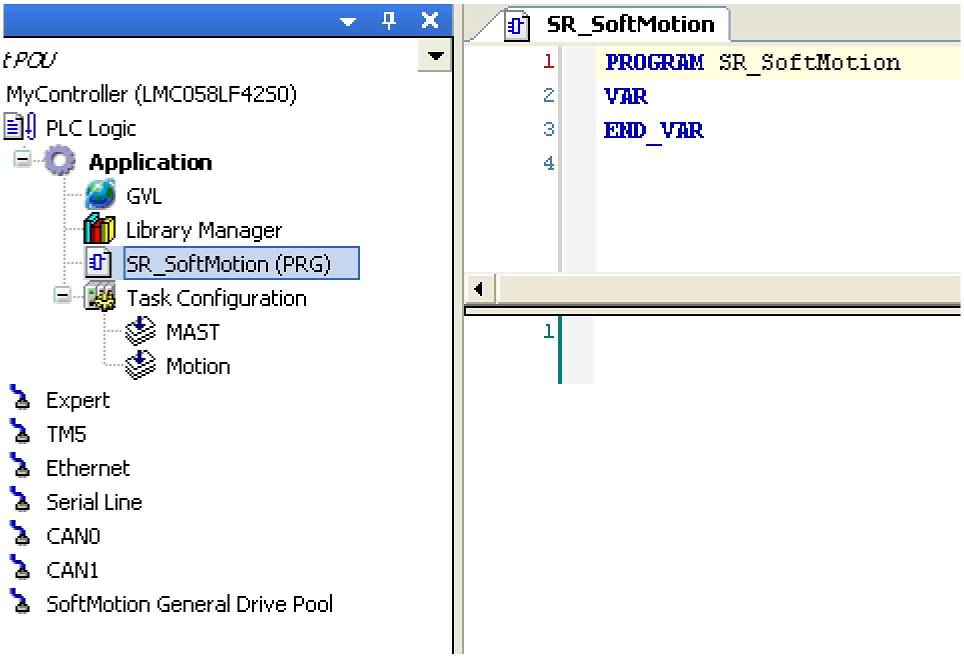
Method implementation language: Function Block Diagram (FBD)

○ Function

Return type:

Implementation language: Function Block Diagram (FBD)

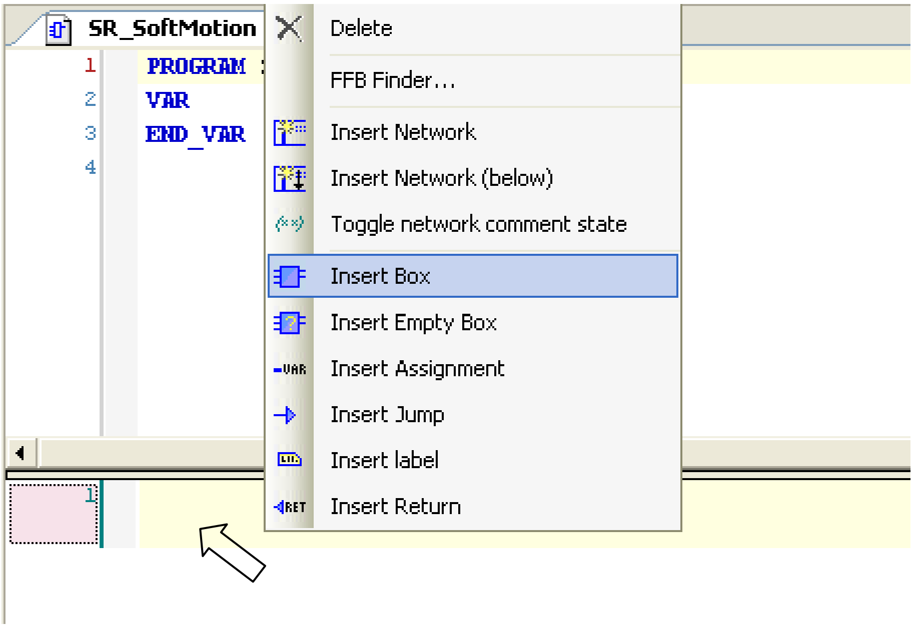
iv. Click **Open** to accept the configuration, and display the Function Block Diagram editor



```
PROGRAM SR_SoftMotion
VAR
END_VAR
```

2. Instantiate SoftMotion FBs

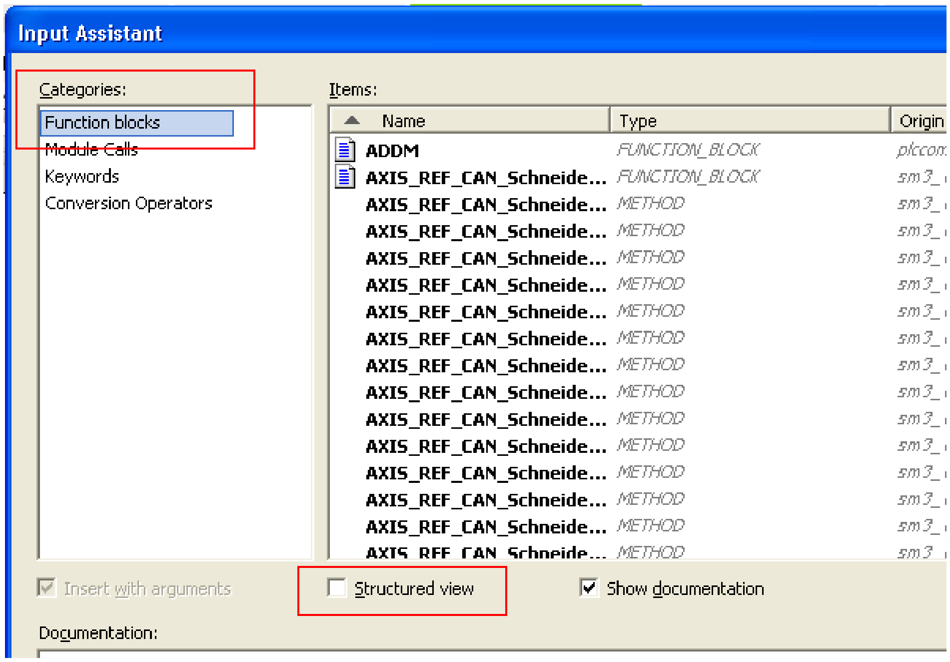
i. Right-click within the first FBD network rung, and select **Insert Box**.



```
1  PROGRAM :
2  VAR
3  END_VAR
4
```

ii. From the Input Assistant menu, make sure that the **Function blocks** category is selected. Also, uncheck **Structured view**.

Available Function blocks are listed in alphabetical order.

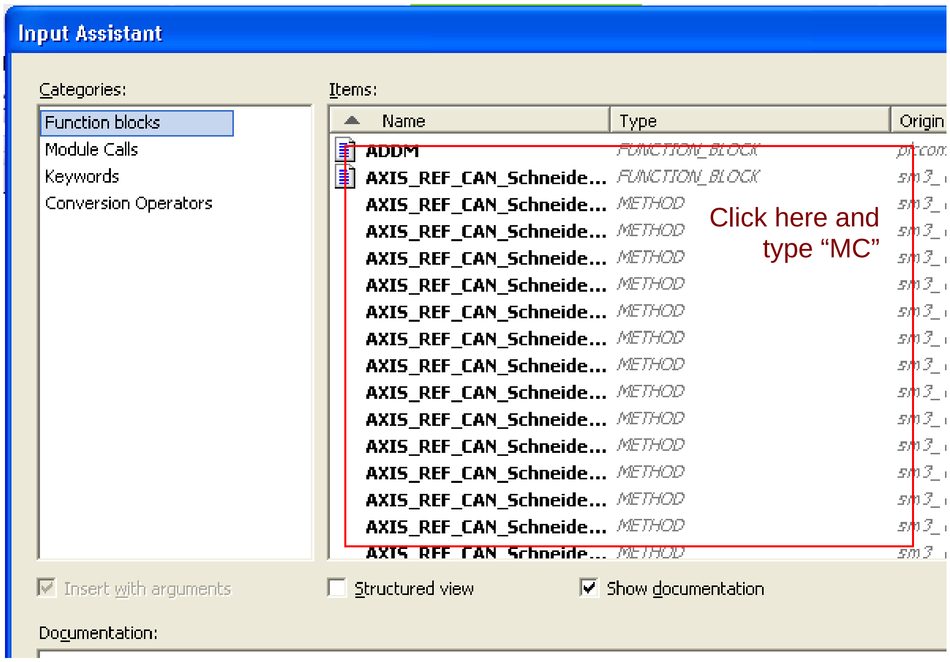


Name	Type	Origin
ADDM	FUNCTION_BLOCK	plccon
AXIS_REF_CAN_Schneide…	FUNCTION_BLOCK	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_
AXIS_REF_CAN_Schneide…	METHOD	sm3_

☑ Insert with arguments ☐ Structured view ☑ Show documentation

Documentation:

iii. The PLCopen standard motion function blocks are indicated by the “MC_” prefix, Click anywhere on the right-hand panel, and type “MC” to navigate quickly to the PLCopen Function blocks.

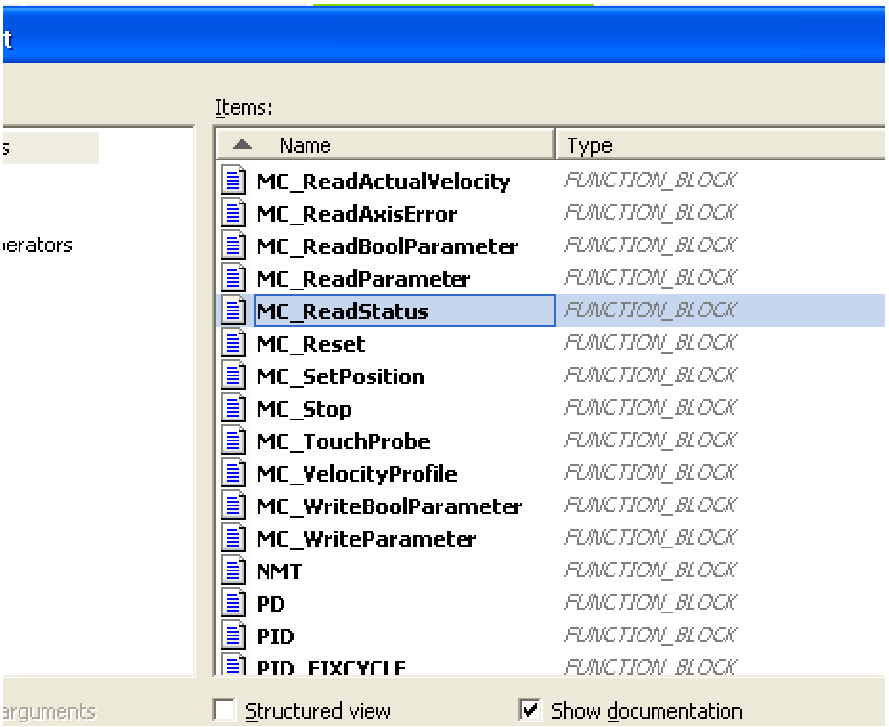


Name	Type	Origin
ADDM	FUNCTION_BLOCK	plccon
AXIS_REF_CAN_Schneide...	FUNCTION_BLOCK	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_
AXIS_REF_CAN_Schneide...	METHOD	sm3_

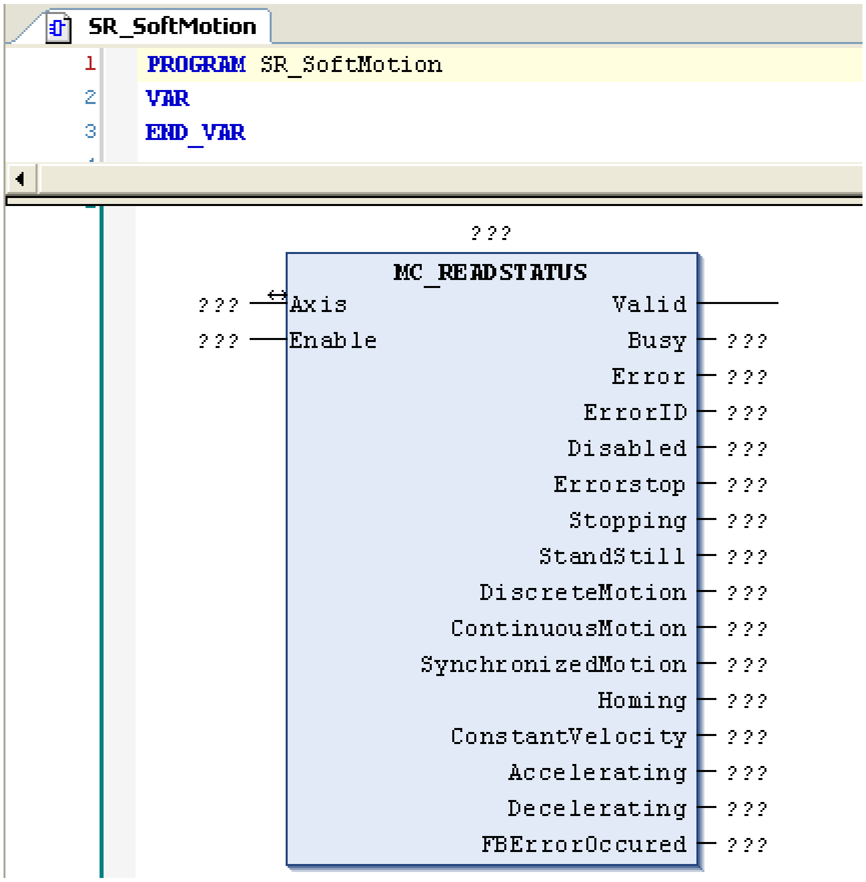
Click here and type “MC”

This will bring up the list of PLCopen motion control FBs starting with “MC_”.

iv. Scroll to **MC_ReadStatus**, and double-click (or select **OK)** to instantiate the FB into the editor.



Name	Type
MC_ReadActualVelocity	FUNCTION_BLOCK
MC_ReadAxisError	FUNCTION_BLOCK
MC_ReadBoolParameter	FUNCTION_BLOCK
MC_ReadParameter	FUNCTION_BLOCK
MC_ReadStatus	FUNCTION_BLOCK
MC_Reset	FUNCTION_BLOCK
MC_SetPosition	FUNCTION_BLOCK
MC_Stop	FUNCTION_BLOCK
MC_TouchProbe	FUNCTION_BLOCK
MC_VelocityProfile	FUNCTION_BLOCK
MC_WriteBoolParameter	FUNCTION_BLOCK
MC_WriteParameter	FUNCTION_BLOCK
NMT	FUNCTION_BLOCK
PD	FUNCTION_BLOCK
PID	FUNCTION_BLOCK
PID_FIXCYCLE	FUNCTION_BLOCK

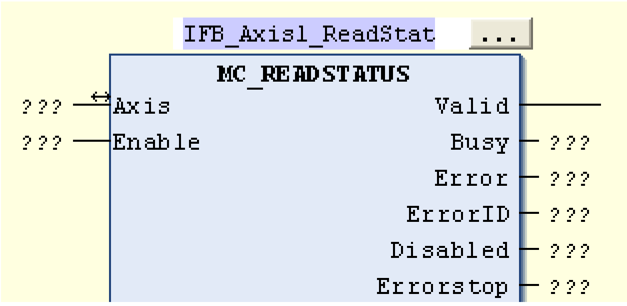


```
PROGRAM SR_SoftMotion
VAR
END_VAR
```

In order to complete the instantiation, the function block instance must be given a unique name, and any **Input Output** variables, indicated by a ↔ connector, must be assigned.

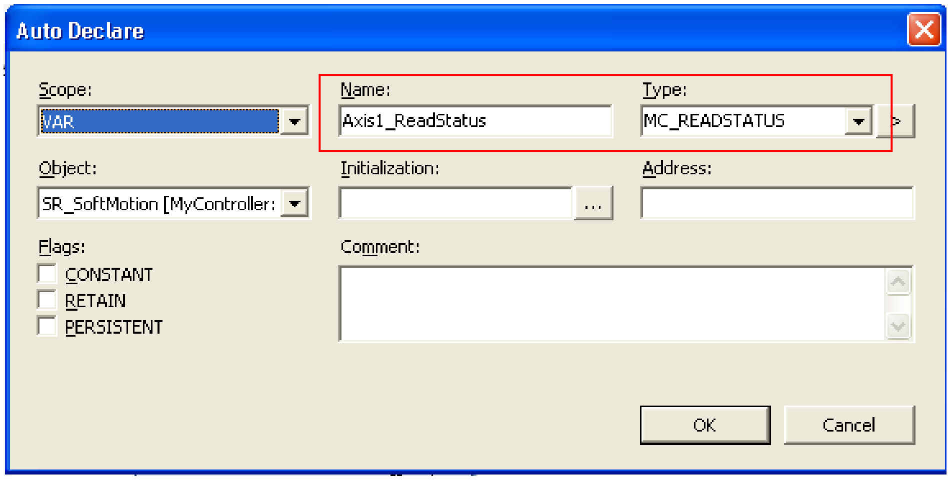
A standard **Input Output** object for all SoftMotion function blocks is the Axis_Ref_SM3 object as indicated by the “Axis” pin description.

v. Click on the **???** symbol at the top of the function block, and assign the instance name **IFB_Axis1_ReadStatus**.



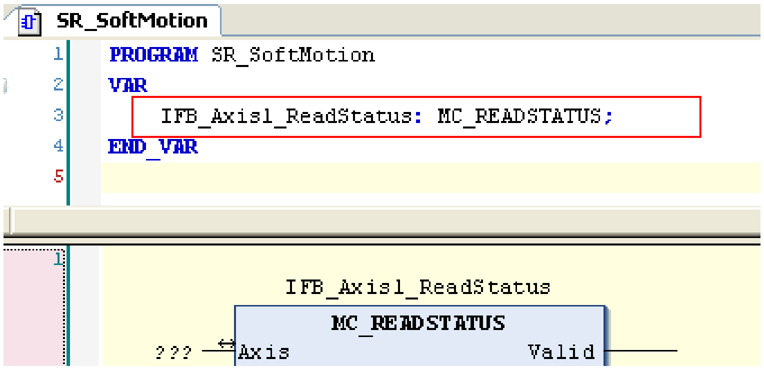
The notation “IFB_” indicates that this is an “Instance of a Function Block”, and is consistent with PLCopen variable naming convention.

vi. Make sure that the Auto Declaration screen showns the correct data type. Click OK to accept the declaration as shown.



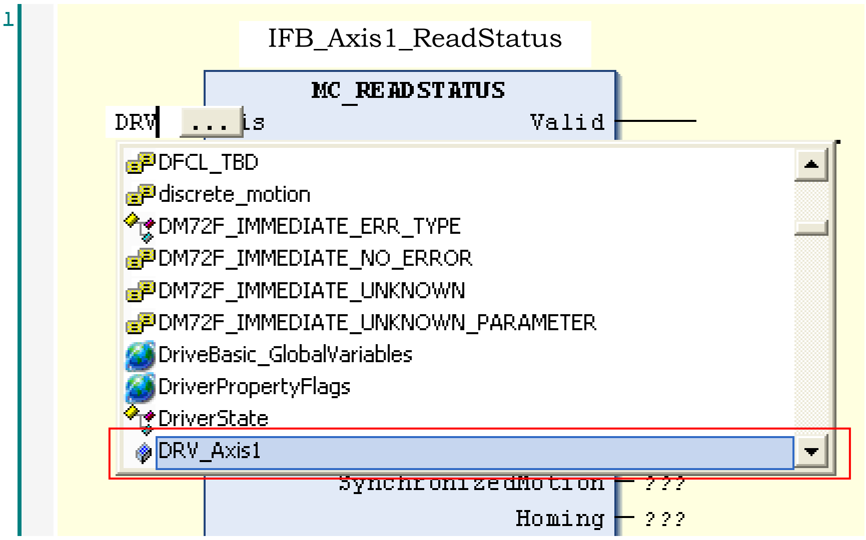
Scope: VAR
Name: Axis1_ReadStatus
Type: MC_READSTATUS
Object: SR_SoftMotion [MyController:

Note the new variable declaration in the Declaration window of the FBD editor

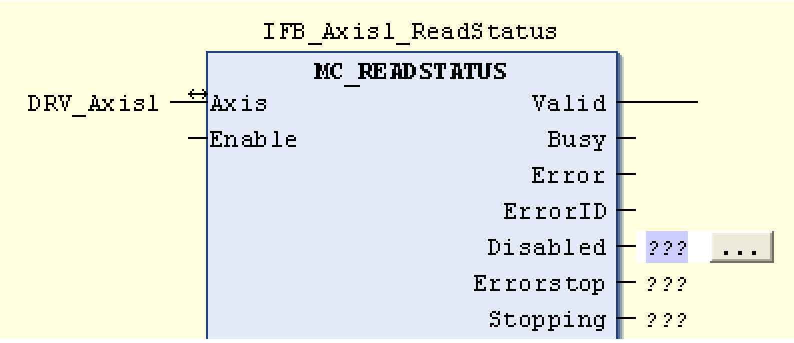


```
PROGRAM SR_SoftMotion
VAR
    IFB_Axis1_ReadStatus: MC_READSTATUS;
END_VAR
```

vii. Click on the “axis” input pin and Type “DRV_Axis1” to assign the Axis_Ref_SM3 object. The variable should appear in the “smart code” pick list as shown.



viii. Finally, delete all of the remaining ??? input and output symbols by double-clicking the symbol (cursor mode) followed by the <delete> key.



---

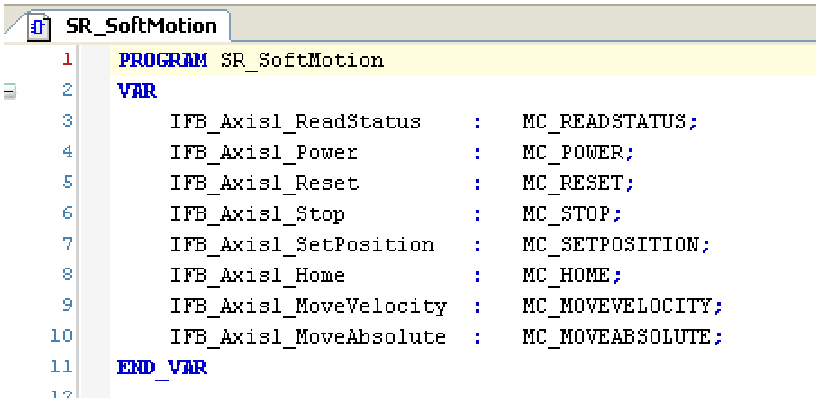
3. Add the remaining PLCopen FBs to the POU.

i. Following the steps as before, instantiate each of the following FBs and instance names.

Hint… Copy and Paste the MC_ReadStatus network rung, and edit the function block and instance name!

MC_Reset	IFB_Axis1_Reset
MC_Power	IFB_Axis1_Power
MC_Stop	IFB_Axis1_Stop
MC_SetPosition	IFB_Axis1_SetPosition
MC_Home	IFB_Axis1_Home
MC_MoveVelocity	IFB_Axis1_MoveVelocity
MC_MoveAbsolute	IFB_Axis1_MoveAbsolute

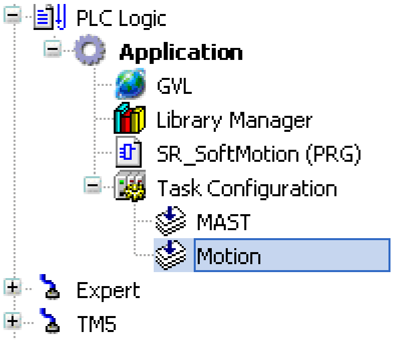
ii. Confirm the completed variable declaration pane as shown when finished.



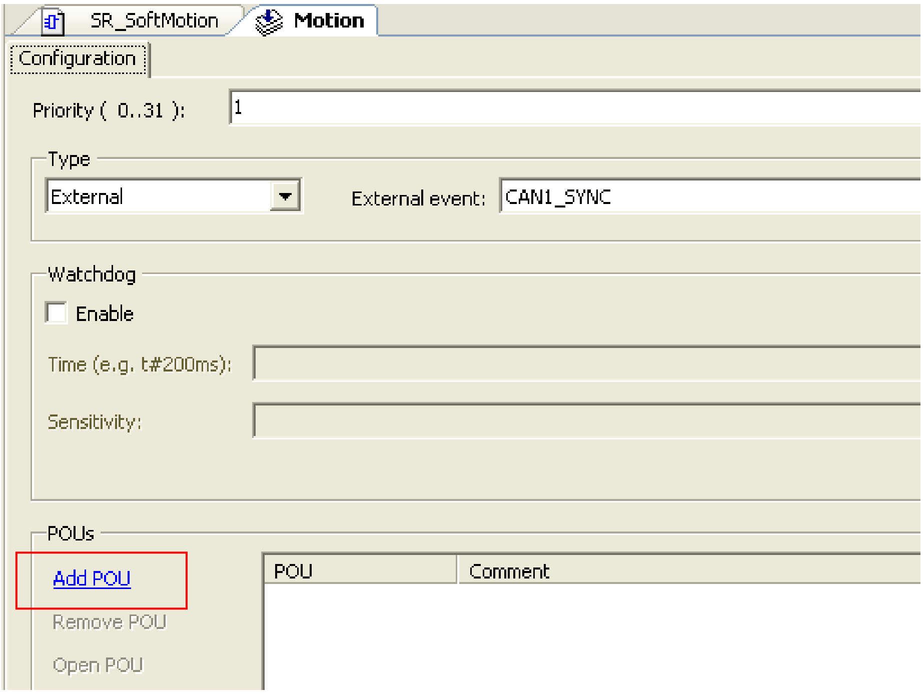
```
PROGRAM SR_SoftMotion
VAR
    IFB_Axis1_ReadStatus    :   MC_READSTATUS;
    IFB_Axis1_Power         :   MC_POWER;
    IFB_Axis1_Reset         :   MC_RESET;
    IFB_Axis1_Stop          :   MC_STOP;
    IFB_Axis1_SetPosition   :   MC_SETPOSITION;
    IFB_Axis1_Home          :   MC_HOME;
    IFB_Axis1_MoveVelocity  :   MC_MOVEVELOCITY;
    IFB_Axis1_MoveAbsolute  :   MC_MOVEABSOLUTE;
END_VAR
```

4. Call the POU from the Motion Task

i. Double-click the Motion task to open the task configuration editor.



ii. Select **Add POU** in the task editor…



Priority (0..31): 1

Type: External

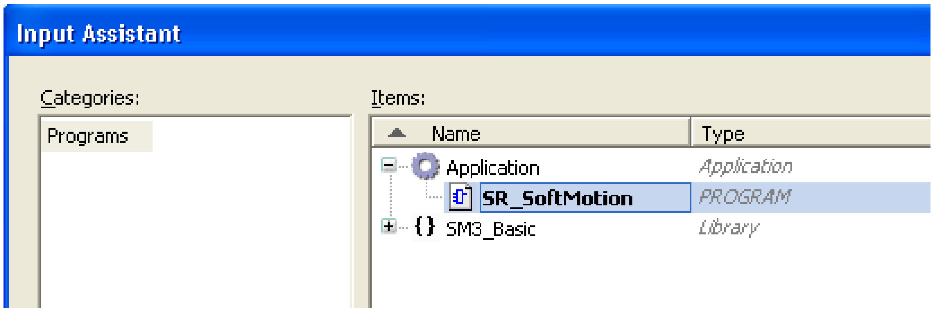
External event: CAN1_SYNC

☐ Enable

Time (e.g. t#200ms):

Sensitivity:

iii. … and select **SR_SoftMotion** from the available ***Application*** choices.



Name	Type
Application	*Application*
SR_SoftMotion	*PROGRAM*
SM3_Basic	*Library*

5. Build and Save

i. **Build** the project as before, and confirm that there are no build errors.

ii. **Save** the project

This completes the exercise.

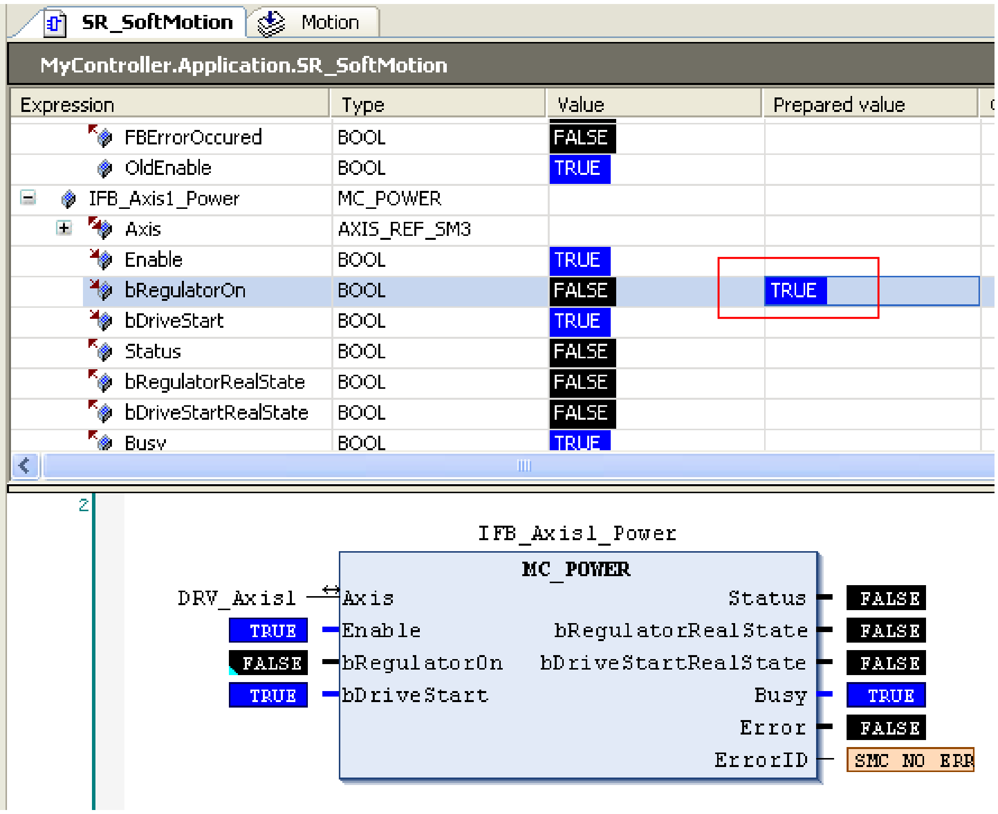
Controlling an Axis Using System Variables

At this point, all of the necessary functional mapping is in place to control Axis1. When the application is downloaded to the controller, and the controller is started, the function blocks within SR_SoftMotion will be scanned on every cycle of the motion task.

To control the axis, we simply have to confirm the axis state, and manage the state of the FB parameters and execution inputs. SoMachine provides a convenient method for manipulating these system variables online.

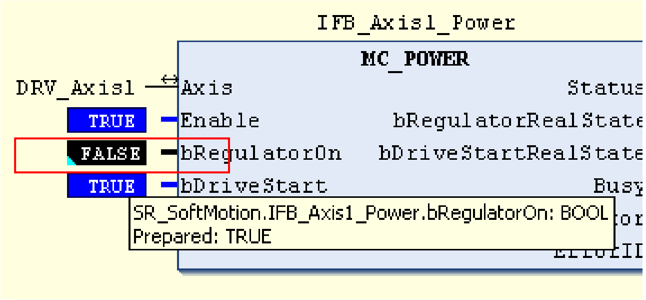
Online Declaration Variables

While online, the SoMachine declaration editor provides a **Prepared Value** column. Here, the user can modify the selected variable state or value, and impose the change with the <CTRL><F7> keys. Click in the **Prepared Value** field as shown to edit the value.



Expression	Type	Value	Prepared value
FBErrorOccured	BOOL	FALSE	
OldEnable	BOOL	TRUE	
IFB_Axis1_Power	MC_POWER		
Axis	AXIS_REF_SM3		
Enable	BOOL	TRUE	
bRegulatorOn	BOOL	FALSE	TRUE
bDriveStart	BOOL	TRUE	
Status	BOOL	FALSE	
bRegulatorRealState	BOOL	FALSE	
bDriveStartRealState	BOOL	FALSE	
Busy	BOOL	TRUE	

It is also possible to click directly on the FB input to "prepare" a new value or state. However, a progression of clicks is required to move from TRUE to FALSE to CANCEL, and the pointer text is the only way to verify the selected state.

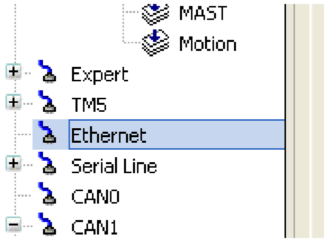


SR_SoftMotion.IFB_Axis1_Power.bRegulatorOn: BOOL
Prepared: TRUE

Exercise – Control an Axis

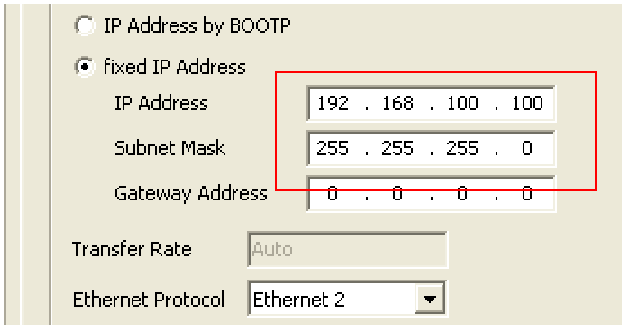
1. Configure the Ethernet communication settings.

i. From the browser, double-click the **Ethernet** port object



MAST
Motion
Expert
TM5
Ethernet
Serial Line
CAN0
CAN1

ii. Enter the Controller IP address and subnet in the Ethernet editor as indicated below.



○ IP Address by BOOTP

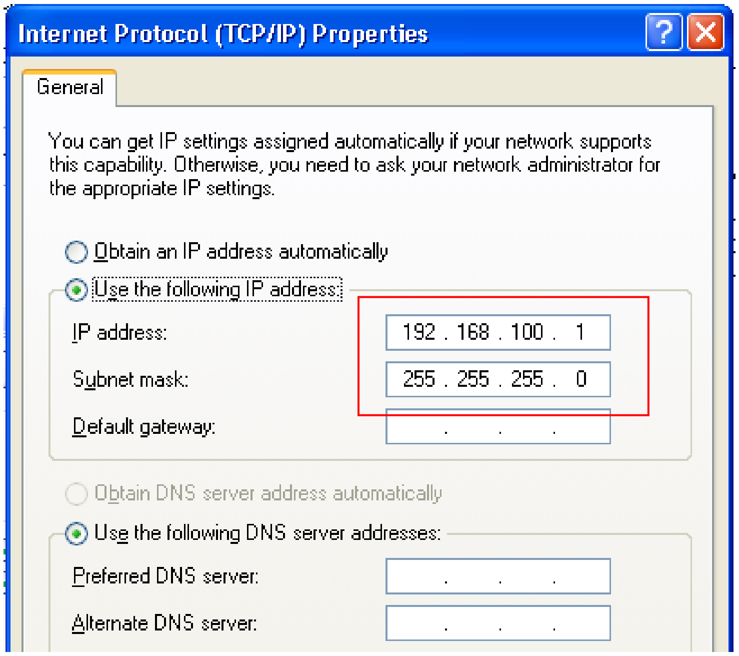
● fixed IP Address

IP Address 192 . 168 . 100 . 100
Subnet Mask 255 . 255 . 255 . 0
Gateway Address 0 . 0 . 0 . 0

Transfer Rate Auto

Ethernet Protocol Ethernet 2

iii. Make sure that the laptop is configured with a fixed IP address on the same subnet as the controller.



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

○ Obtain an IP address automatically

● Use the following IP address:

IP address: 192 . 168 . 100 . 1
Subnet mask: 255 . 255 . 255 . 0
Default gateway: . . .

○ Obtain DNS server address automatically

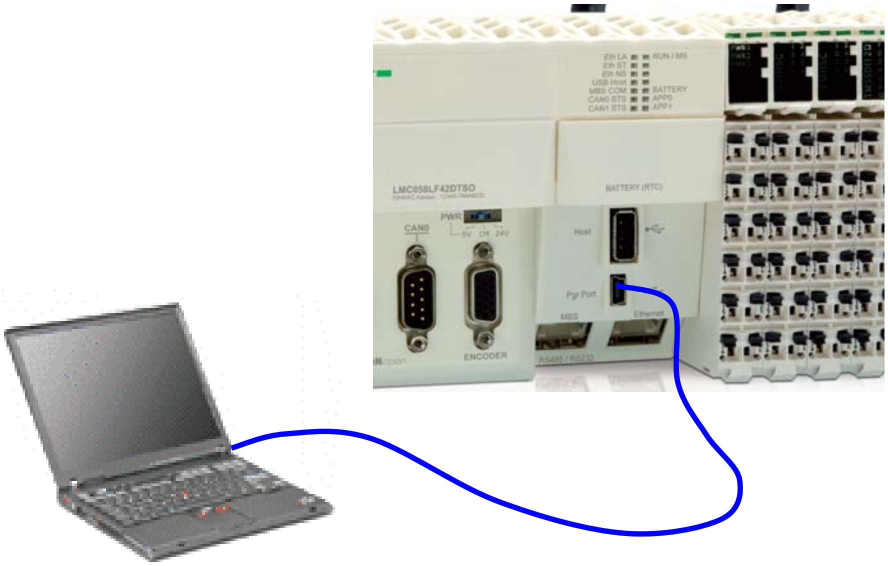
● Use the following DNS server addresses:

Preferred DNS server: . . .
Alternate DNS server: . . .

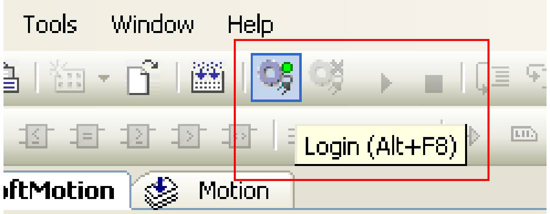
iii. **Save** the project.

2. Connect to the Controller

i. Connect a USB serial cable between the laptop and mini USB port on the controller.



ii. From SoMachine, connect to the controller using the **Login** icon at the top level menu.



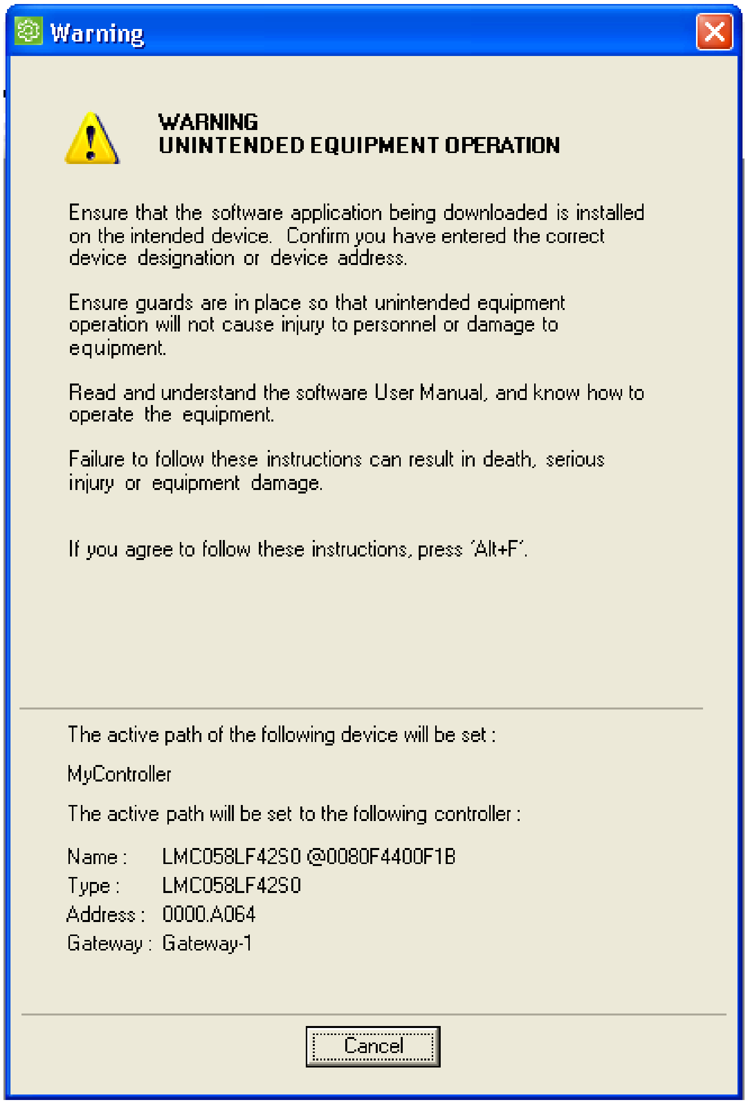
If this is the first connection, SoMachine will attempt to establish a Gateway connection to the controller.

Active path not set for login.
Scanning for matching controllers...

Using gateway:

Gateway-1

iii. Press the **<ALT> <F>** keys to complete the connection.



WARNING
UNINTENDED EQUIPMENT OPERATION

Ensure that the software application being downloaded is installed on the intended device. Confirm you have entered the correct device designation or device address.

Ensure guards are in place so that unintended equipment operation will not cause injury to personnel or damage to equipment.

Read and understand the software User Manual, and know how to operate the equipment.

Failure to follow these instructions can result in death, serious injury or equipment damage.

If you agree to follow these instructions, press 'Alt+F'.

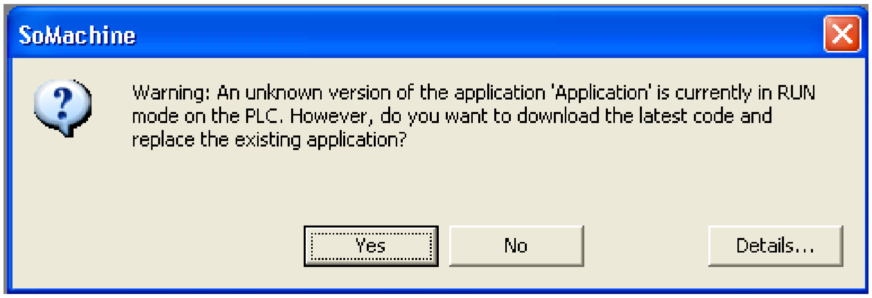
The active path of the following device will be set :

MyController

The active path will be set to the following controller :

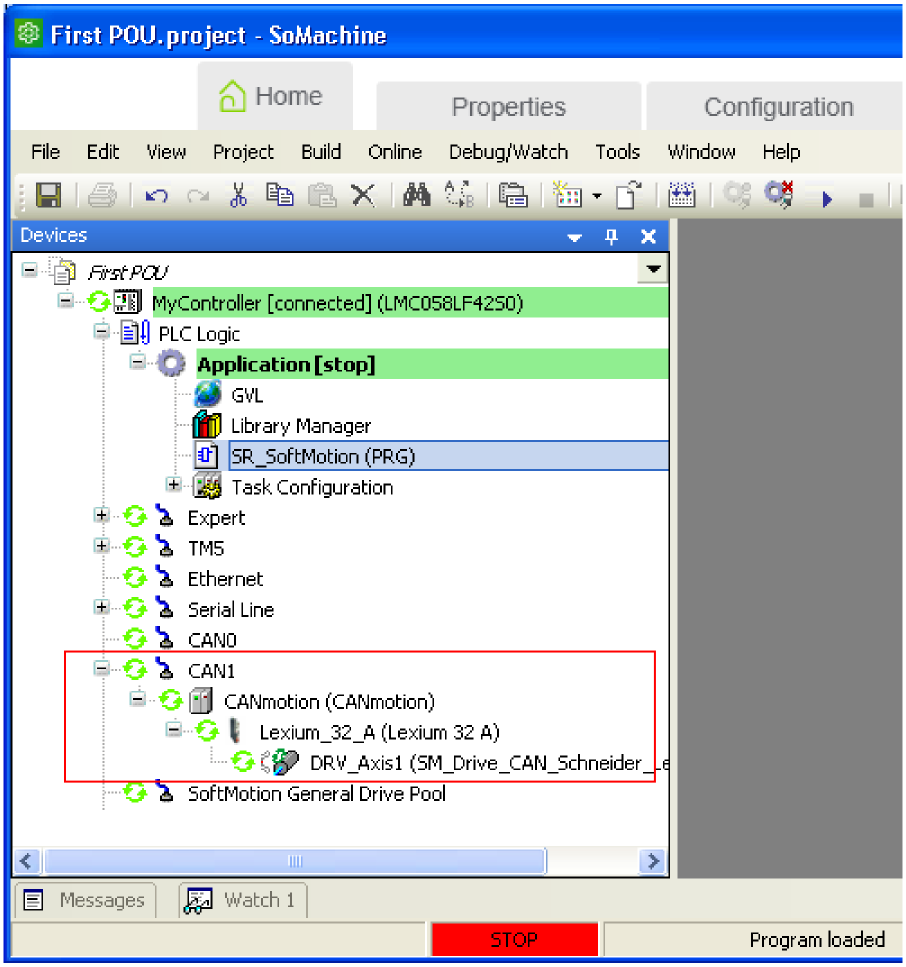
Name : LMC058LF42S0 @0080F4400F1B
Type : LMC058LF42S0
Address : 0000.A064
Gateway : Gateway-1

iv. Select **Yes** if prompted to download the application.



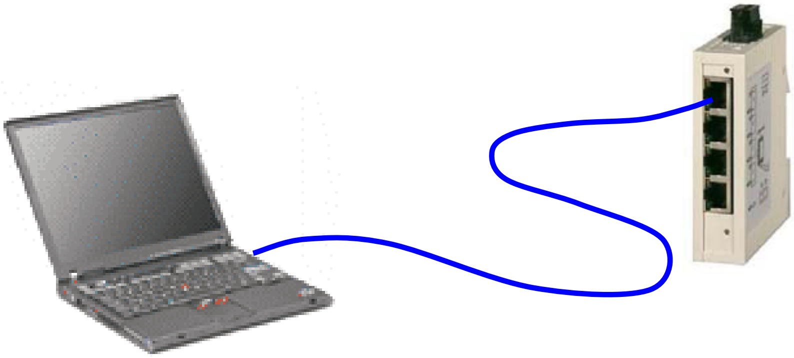
Warning: An unknown version of the application 'Application' is currently in RUN mode on the PLC. However, do you want to download the latest code and replace the existing application?

When the download has completed, the browser will indicate correct configuration status for each the hardware objects by the green symbol.



As indicated above, the mapped CANmotion bus and axis objects match the hardware configuration of the physical drive, and communication has been established.

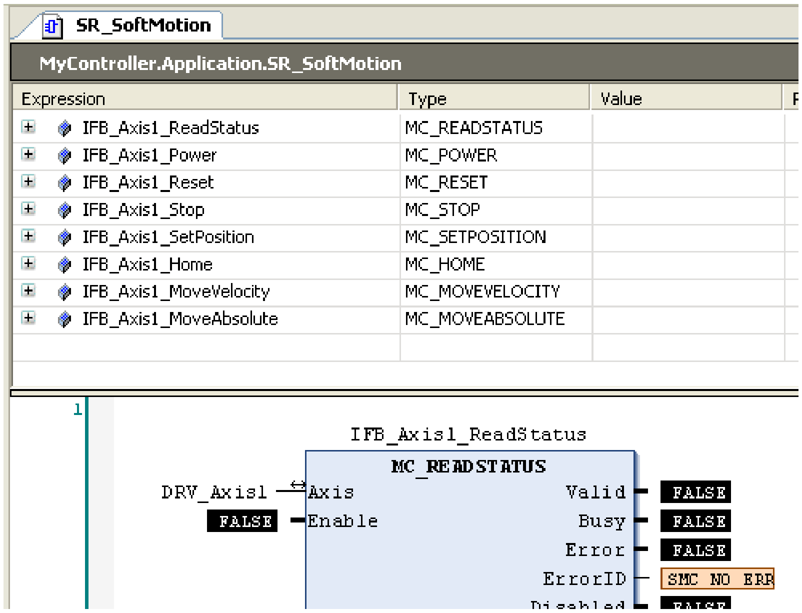
At this time, the Ethernet settings have been applied to the controller. Once the controller settings are known, it is possible to use Ethernet instead of the serial connection for all subsequent downloads.



3. Activate the Axis using System Variables

i. Place the Controller into Run mode using the Start icon at the top level menu.

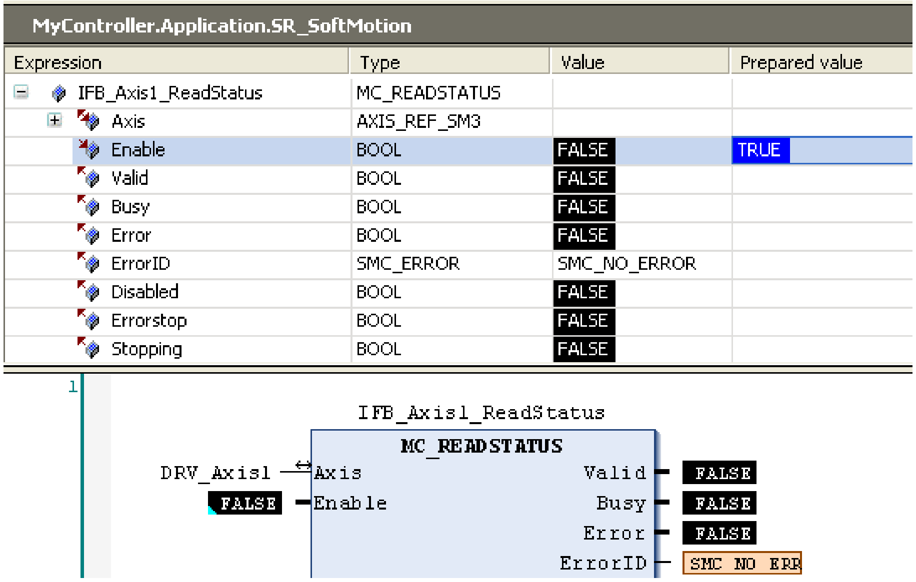
ii. Double-click on the **SR_SoftMotion** POU to open the online declaration control panel.



MyController.Application.SR_SoftMotion

Expression	Type	Value
⊞ IFB_Axis1_ReadStatus	MC_READSTATUS	
⊞ IFB_Axis1_Power	MC_POWER	
⊞ IFB_Axis1_Reset	MC_RESET	
⊞ IFB_Axis1_Stop	MC_STOP	
⊞ IFB_Axis1_SetPosition	MC_SETPOSITION	
⊞ IFB_Axis1_Home	MC_HOME	
⊞ IFB_Axis1_MoveVelocity	MC_MOVEVELOCITY	
⊞ IFB_Axis1_MoveAbsolute	MC_MOVEABSOLUTE	

iii. Access the **MC_ReadStatus** structure variables by clicking on the + sign, and prepare the value TRUE for the Enable input.

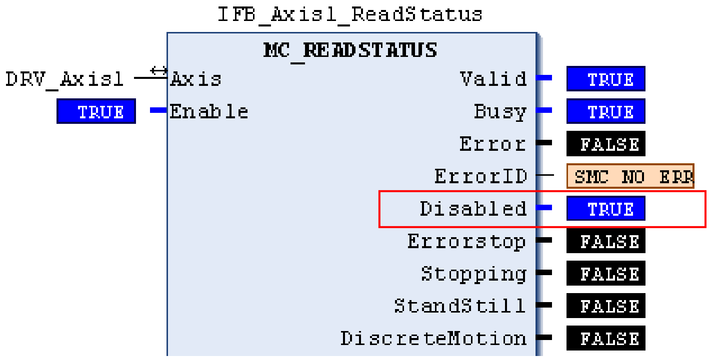


MyController.Application.SR_SoftMotion

Expression	Type	Value	Prepared value
⊟ IFB_Axis1_ReadStatus	MC_READSTATUS		
⊞ Axis	AXIS_REF_SM3		
Enable	BOOL	FALSE	TRUE
Valid	BOOL	FALSE	
Busy	BOOL	FALSE	
Error	BOOL	FALSE	
ErrorID	SMC_ERROR	SMC_NO_ERROR	
Disabled	BOOL	FALSE	
Errorstop	BOOL	FALSE	
Stopping	BOOL	FALSE	

iv. Apply the Enable >> TRUE by pressing <CRTL> <F7>.

Note that the initial PLCopen state of the axis is “Disabled”

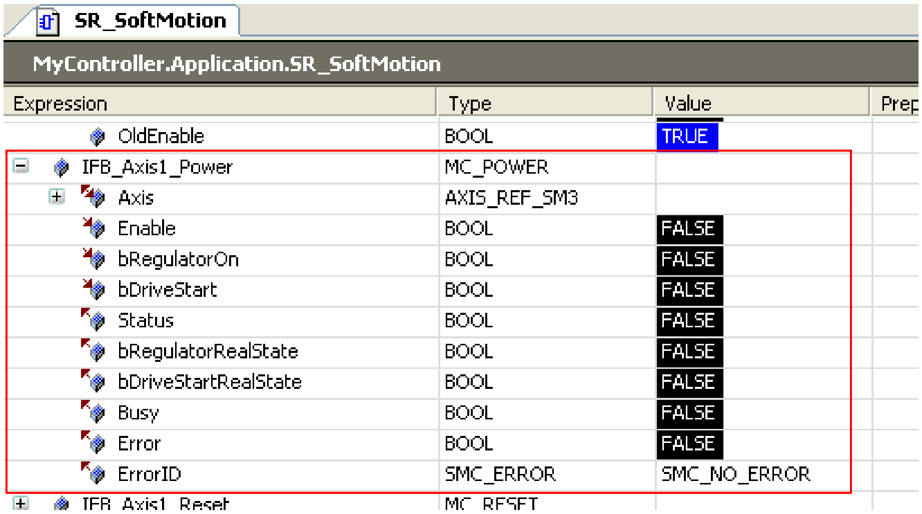


IFB_Axis1_ReadStatus

MC_READSTATUS

Input	Value	Output	Value
Axis	DRV_Axis1	Valid	TRUE
Enable	TRUE	Busy	TRUE
		Error	FALSE
		ErrorID	SMC_NO_ERR
		Disabled	TRUE
		Errorstop	FALSE
		Stopping	FALSE
		StandStill	FALSE
		DiscreteMotion	FALSE

v. Scroll down to the **MC_Power** FB, and open the structure as before.



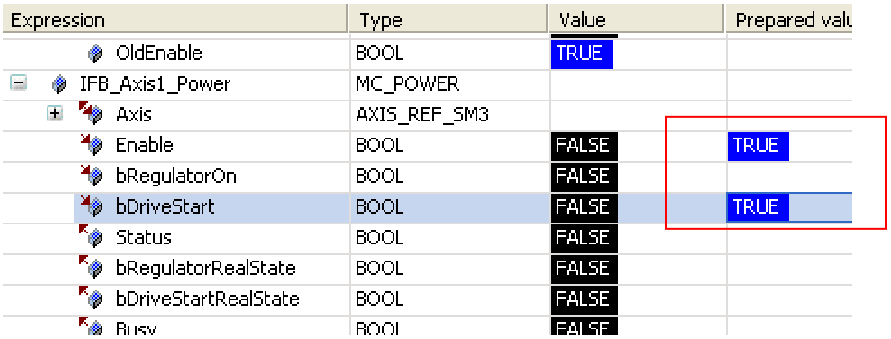
SR_SoftMotion

MyController.Application.SR_SoftMotion

Expression	Type	Value	Prep
OldEnable	BOOL	TRUE	
IFB_Axis1_Power	MC_POWER		
Axis	AXIS_REF_SM3		
Enable	BOOL	FALSE	
bRegulatorOn	BOOL	FALSE	
bDriveStart	BOOL	FALSE	
Status	BOOL	FALSE	
bRegulatorRealState	BOOL	FALSE	
bDriveStartRealState	BOOL	FALSE	
Busy	BOOL	FALSE	
Error	BOOL	FALSE	
ErrorID	SMC_ERROR	SMC_NO_ERROR	
IFB_Axis1_Reset	MC_RESET		

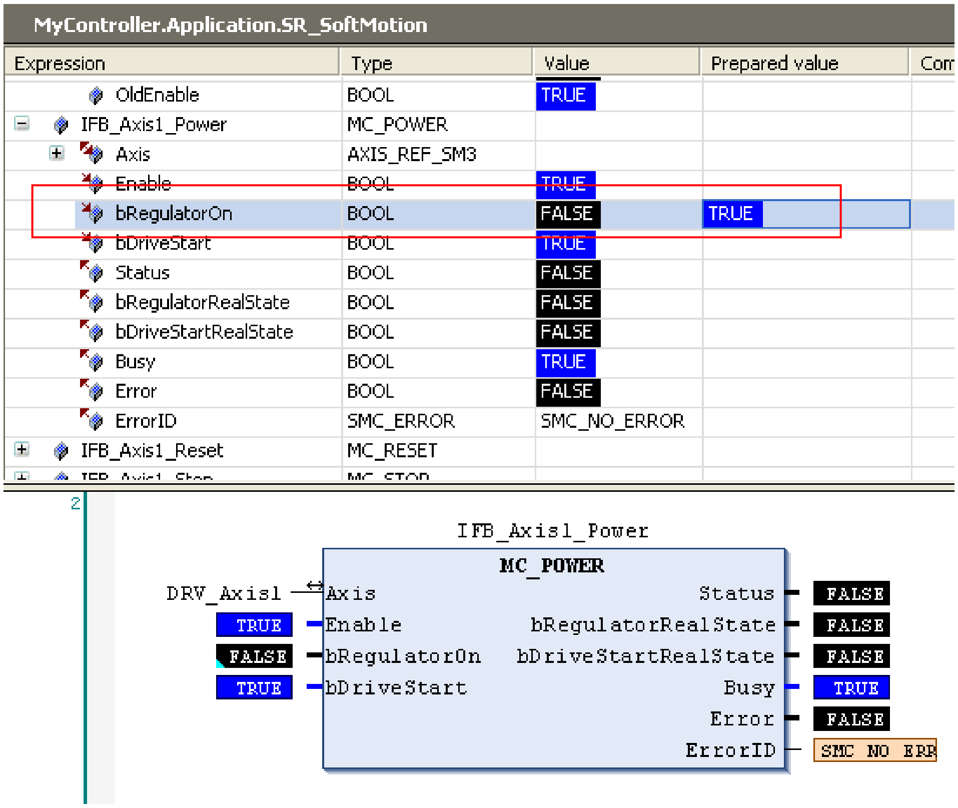
__MC_Power__ accommodates functionality that is not supported by the Lexium32 servo drives. Therefore, the __Enable__ and __bDriveStart__ inputs are generally hard-coded as __TRUE__, and the power stage of the drive is activated using the __bRegulatorOn__ input.

vi. Prepare the Enable and bDriveStart inputs TRUE, and apply using <CTRL><F7> as before.



Expression	Type	Value	Prepared valu
OldEnable	BOOL	TRUE	
IFB_Axis1_Power	MC_POWER		
Axis	AXIS_REF_SM3		
Enable	BOOL	FALSE	TRUE
bRegulatorOn	BOOL	FALSE	
bDriveStart	BOOL	FALSE	TRUE
Status	BOOL	FALSE	
bRegulatorRealState	BOOL	FALSE	
bDriveStartRealState	BOOL	FALSE	
Busy	BOOL	FALSE	

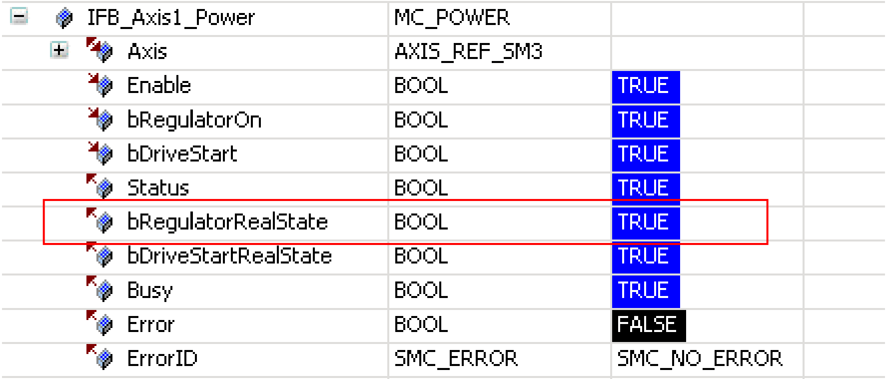
vii. Activate the power stage by setting the **bRegulatorON** input TRUE.



MyController.Application.SR_SoftMotion

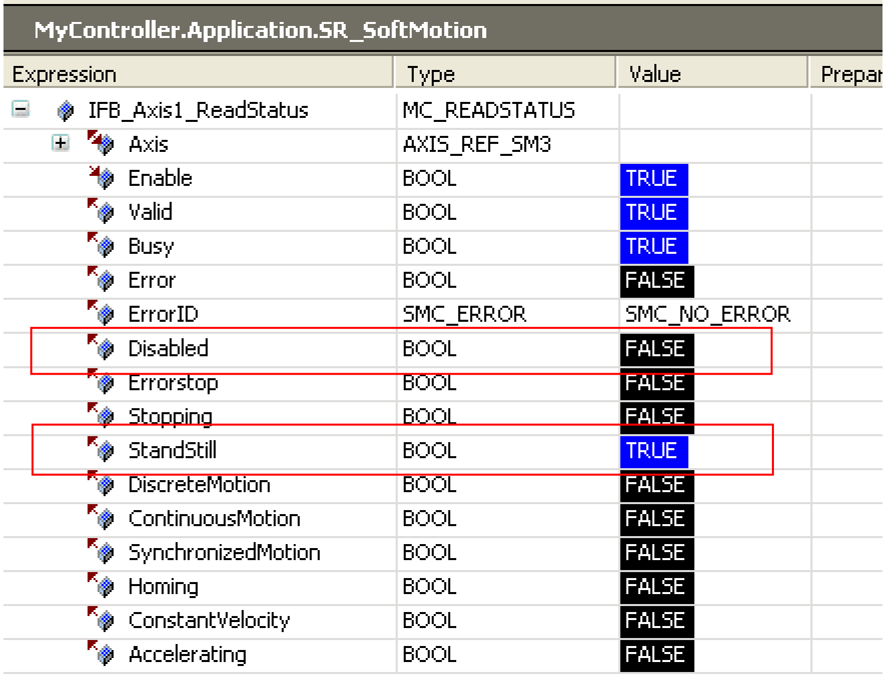
Expression	Type	Value	Prepared value	Com
OldEnable	BOOL	TRUE		
IFB_Axis1_Power	MC_POWER			
Axis	AXIS_REF_SM3			
Enable	BOOL	TRUE		
bRegulatorOn	BOOL	FALSE	TRUE	
bDriveStart	BOOL	TRUE		
Status	BOOL	FALSE		
bRegulatorRealState	BOOL	FALSE		
bDriveStartRealState	BOOL	FALSE		
Busy	BOOL	TRUE		
Error	BOOL	FALSE		
ErrorID	SMC_ERROR	SMC_NO_ERROR		
IFB_Axis1_Reset	MC_RESET			

viii. Confirm that the **bRegulatorRealState** output turns ON, and the motor shaft locks into position.



IFB_Axis1_Power	MC_POWER	
Axis	AXIS_REF_SM3	
Enable	BOOL	TRUE
bRegulatorOn	BOOL	TRUE
bDriveStart	BOOL	TRUE
Status	BOOL	TRUE
bRegulatorRealState	BOOL	TRUE
bDriveStartRealState	BOOL	TRUE
Busy	BOOL	TRUE
Error	BOOL	FALSE
ErrorID	SMC_ERROR	SMC_NO_ERROR

ix. Finally, confirm from the **ReadStatus** FB, that the PLCopen state has changed from **Disabled** to **Standstill**.



MyController.Application.SR_SoftMotion

Expression	Type	Value	Prepar
IFB_Axis1_ReadStatus	MC_READSTATUS		
Axis	AXIS_REF_SM3		
Enable	BOOL	TRUE	
Valid	BOOL	TRUE	
Busy	BOOL	TRUE	
Error	BOOL	FALSE	
ErrorID	SMC_ERROR	SMC_NO_ERROR	
Disabled	BOOL	FALSE	
Errorstop	BOOL	FALSE	
Stopping	BOOL	FALSE	
StandStill	BOOL	TRUE	
DiscreteMotion	BOOL	FALSE	
ContinuousMotion	BOOL	FALSE	
SynchronizedMotion	BOOL	FALSE	
Homing	BOOL	FALSE	
ConstantVelocity	BOOL	FALSE	
Accelerating	BOOL	FALSE	

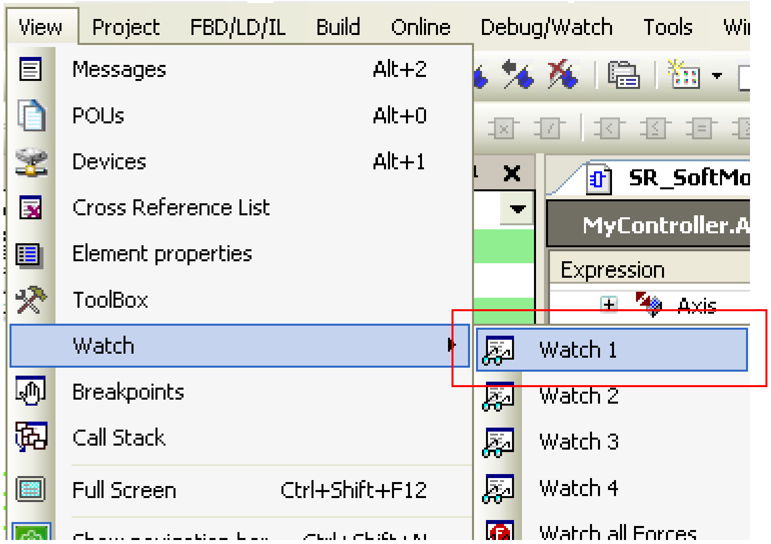
The axis is now active and ready for movement commands.

---

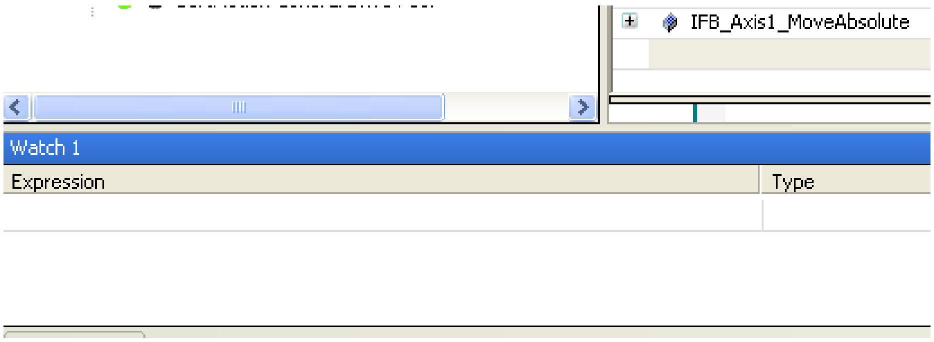
4. Create a Watch List

A watch list can be used to monitor or manually control the application using system and user variables.

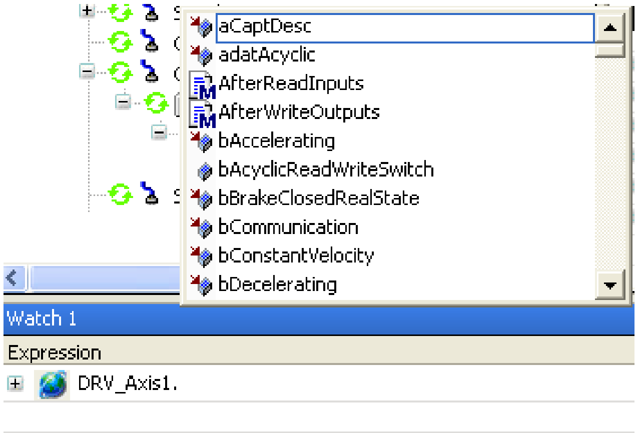
i. Create a Watch list for axis control from the SoMachine top level menu by selecting **View >>Watch >> Watch 1.**



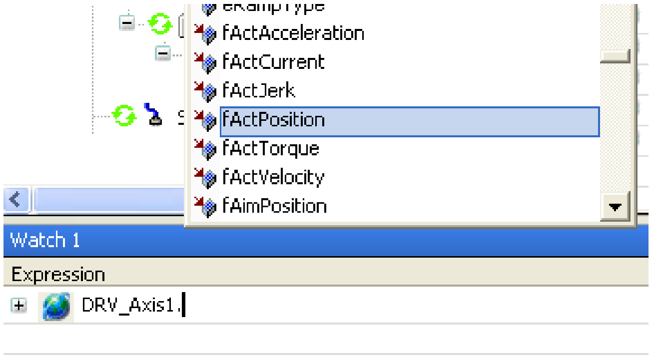
The watch list is created at the bottom of the SoMachine workspace.



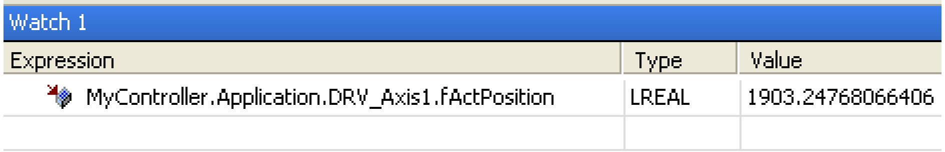
ii. Double-click in the first entry row, and type **DRV_Axis1.** as shown. **DRV_Axis1** is a global structure, and its components are accessed using the dot "." convention.



ii. Scroll down the list to the **fActPosition** component, and <Enter> the selection.



iii. The actual position of **DRV_Axis1** is displayed in the watch list.



Watch 1

Expression	Type	Value
MyController.Application.DRV_Axis1.fActPosition	LREAL	1903.24768066406

4. On your own…

i. Using **the Watch list** to monitor the axis position, and **Online Declaration** panel for control, manipulate the parameters and execution inputs for the remaining FBs to:

- ➢ Set the axis position
- ➢ Perform an Absolute and Relative Move
- ➢ Perform a Velocity Move
- ➢ Stop the Axis

5. Save the Project

i. Save the Project as “First Control”

This completes the Exercise

Chapter 5: Interface Structures

In the previous lesson, a single axis was configured and operated using system variables. In a machine, it is important to have a well-defined structure to manage the flow of commands from the top level operator panel down to the individual axes and synchronized axis sets. Status variables provide annunciation for operator intervention, as well as permissives for control. Control variables dictate the start of processes. These command variables must ultimately be "tied" to the SoftMotion function block inputs (for control) and outputs (for status).

Consider a motion application with 4 physical axes, and a virtual master. If each of these axes make use of 10 SoftMotion function blocks, and each of these function blocks make use of an average of 10 inputs and output variables, then the application requires the instantiation of :

- at least 50 SoftMotion function blocks, and
- over 500 variable and function block declarations

just to accommodate all of the possible control interface requirements!

This is a significant "book-keeping" effort that diverts a substantial amount of time and energy before a single piece of "functional" code is written!

Function block and variable declarations are among the most common sources of program errors !

In this chapter we will configure and implement a user-defined, structured variable interface. This interface will allow us to create re-useable variables, and apply those variables over and over again to as many instances as necessary…. most importantly, without error.

A Note about Standardization

Standardization can be critical to the success of a project, or series of projects. Five different programmers will have five different methods of tackling the same task, and they can all accomplish the same goal. What makes the particular method useful is consistency, and adaptability. When the machine design changes, or is made more flexible, a consistent standard can significantly speed up the modification process. Multiple programmers or technical support staff can view code produced by any other member and understand what's been done, and move forward quickly and efficiently.

International standardization has already been adopted in the form of PLCopen function blocks, and object-oriented programming methods. We have seen examples of this standardization in the instantiation of the SoftMotion function blocks from the previous chapter.

From this point on, we will make use of a PLCopen variable naming convention that applies to all user variables created within the program environment.

Variable Naming Convention

A “PLCopen-compliant” variable naming convention is applied throughout the SoftStruXure Motion template and advanced libraries. The convention carries the benefits of standardization and recognition, both of which can assist in development, readability, and troubleshooting.

For consistency, variables are created using capitalized, concatenated words (no underscore except as required for a prefix). Function block instances, POU, and SFC Step names make use of an underscore to separate parts of the name.

Example:

➢ Local variable:	rActPosition
➢ POU name:	Calc_Position_Deviation

The naming convention describes the variable format, as well as prefixes which indicate the variable and data type. Examples are illustrated in the following table.

Datatype		Prefix 1	Example
BOOL		x	xStart
REAL		r	rProductLength
LREAL		lr	lrTargetPosition
INT		i	iProductCount
DINT		di	diState
UINT		ui	uiCounter
TIME		ti	tiPresetTime
WORD		w	wAlarm
DWORD		dw	dwAxisAlarm
String		s	sExtStateMsg
Enumeration		n	nAxisState
Structure		st	stJogParameters
Pointer		p	pAxisRef
Array		a	arRecipePosition[3] (Array of REAL)
			apAxis_ID[0] (Array of PONTER)
			astAxisControl[1] (Array of structure)
			aIFB_CANMotion_AxisModule[1] (Array of AxisModule instances)

Variable type	Prefix 0		Example
Local	none		diMachineState
Global	g_		g_xActiveAlarm
Physical Input	i_		i_xInputStart
Physical output	q_		q_xFBAlarm
Constant	c_		c_rValueOfPi
Global constant	gc_		gc_iNumberOfAxes
Global input	g_i_		g_i_xEmergencyStopSw
Global_output	g_q_		g_q_xWarningLampOn

Axis Interface

As we determined from the PLCopen state diagram, A motion axis requires a basic set of instructions as well as parameters for control and status. Instructions can be Administrative or Movement including:

Administrative Instructions:

- Power ON OFF (Enable/Disable)
- Reset
- Synchronize

Movement Instructions:

- Homing
- Positioning
- Jogging

Additional parameters are required for the basic instructions, and to monitor the status of the axis. Typical movement parameters include…

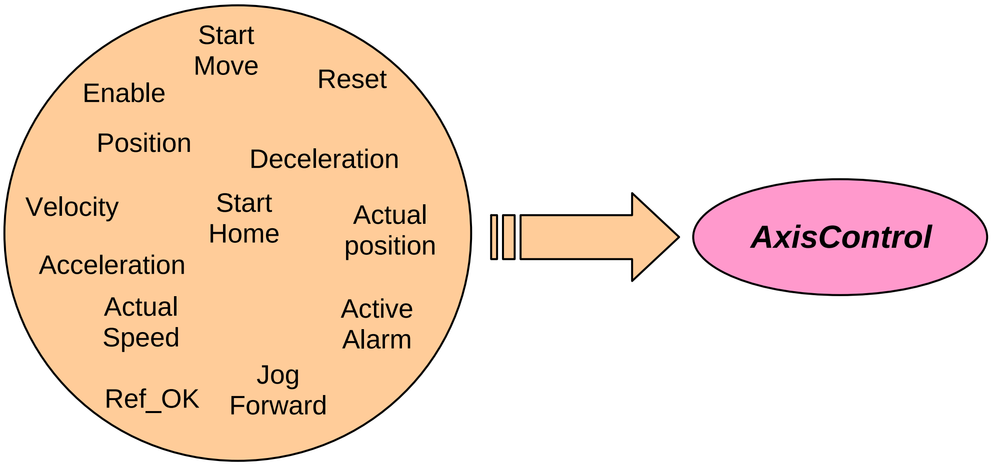
- Target position / velocity
- Acceleration / Deceleration rates
- Homing Speed

… with status parameters such as:

- Power state
- Actual position /velocity
- Alarm state
- Synchronization state
- Reference_OK

We can exploit this common requirement to create a re-usable, custom data type that can be declared once, and then easily applied to every axis in the machine regardless of the number of axes used.

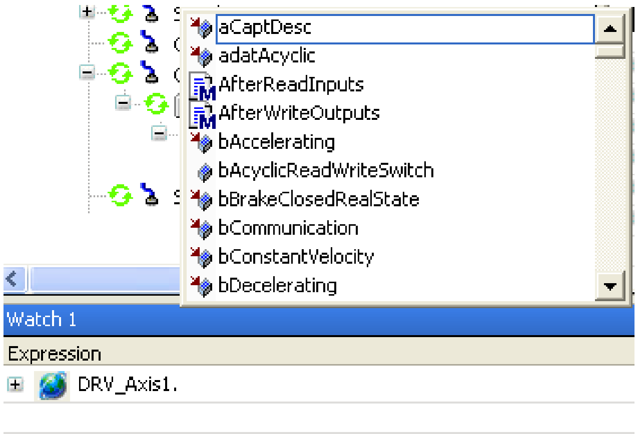
This new data type will be used to create a single variable "structure" that will serve as the control interface to each axis in the machine.



Structures – Compound Data types

Every industrial programmer has used some of the most fundamental data types, such as BOOL, REAL, WORD, STRING, etc. IEC 61131-compliant languages and object-oriented techniques can improve programming efficiency by using "layered" variables, or structures, that contains a collection of related data types as components.

Access to individual components is provided in the form of a "dot" notation. You may recall this notation from the previous section when we controlled the axes using the system variables.



Axis Interface Structure

As an example, consider a new structured data type called **AxisControl_Interface** with the following components.

AxisControl_Interface

- i_xReset
- i_xStartMoveAbs
- i_xStartMoveVel
- i_xSetPosition
- i_rPosition
- i_rVelocity
- i_rAcceleration
- i_rDeceleration
- q_xInRun
- q_xAlarm
- q_nAxisState
- q_ActPosition

We can declare a new program variable in SoMachine…

AxisControl : AxisControl_Interface;

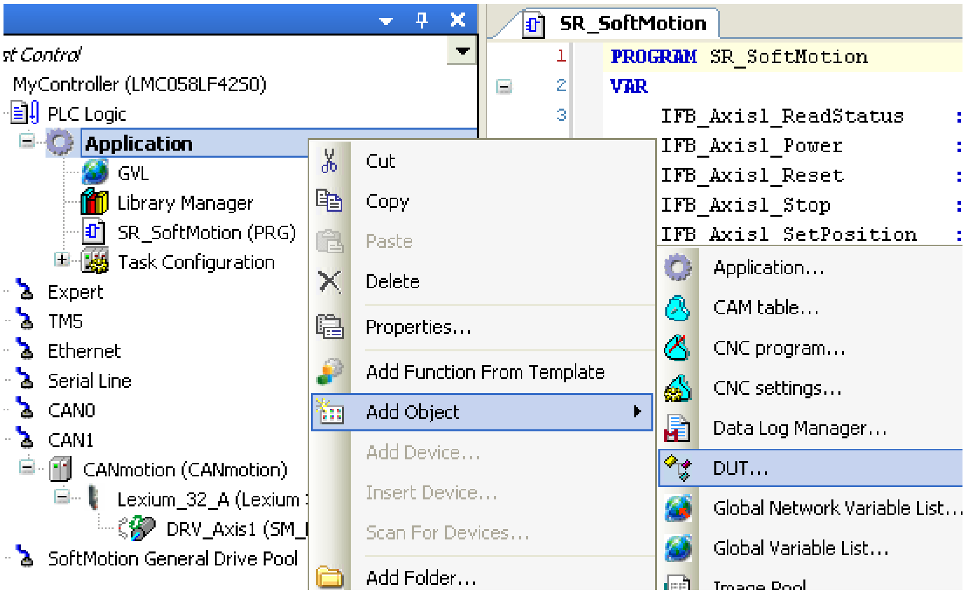
… and apply this new variable to the SoftMotion FBs.

By declaring this variable structure as an Array, a single declaration will accommodate the new command interface for an "umlimited" number of servo axes.

Exercise – Create an Axis Interface Structure

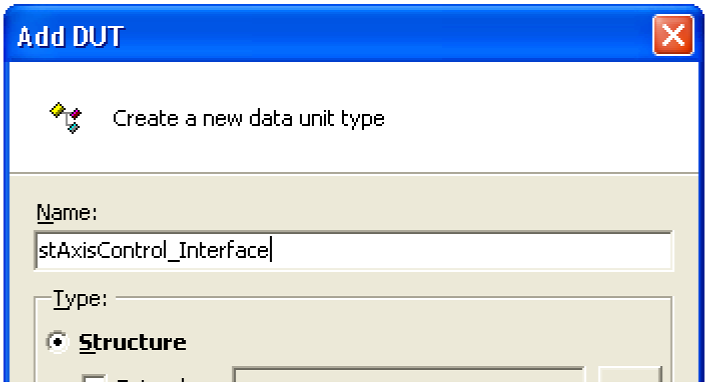
1. Add an Object - DUT

i. From the browser, right-click on the Application object, and select Add Object >> DUT.



```
PROGRAM SR_SoftMotion
VAR
    IFB_Axis1_ReadStatus    :
    IFB_Axis1_Power         :
    IFB_Axis1_Reset         :
    IFB_Axis1_Stop          :
    IFB_Axis1_SetPosition   :
```

ii. Name the DUT "**stAxisControl_Interface**"…

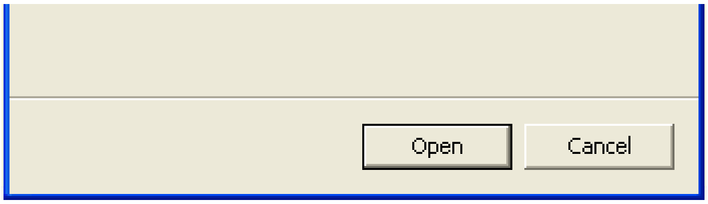


Name: stAxisControl_Interface

Type:

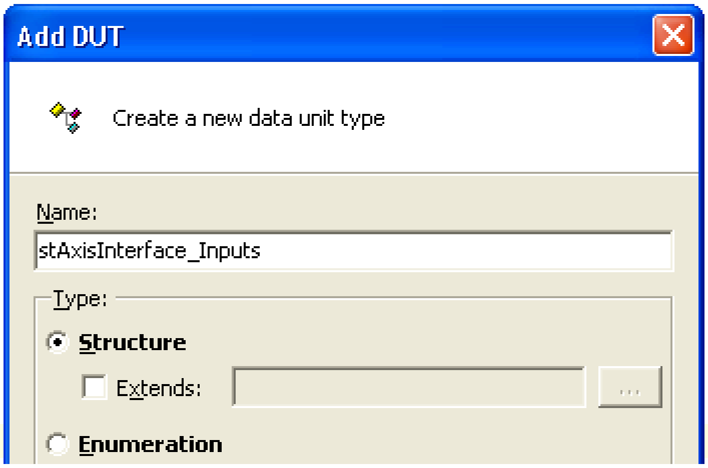
● Structure

iii. … and click **Open** to continue



2. Configure an input (stI) and output (stQ) structure

i. Repeat the process above again to create a DUT Input structure "**stAxisInterface_Inputs**" as shown.



Add DUT

Create a new data unit type

Name: stAxisInterface_Inputs

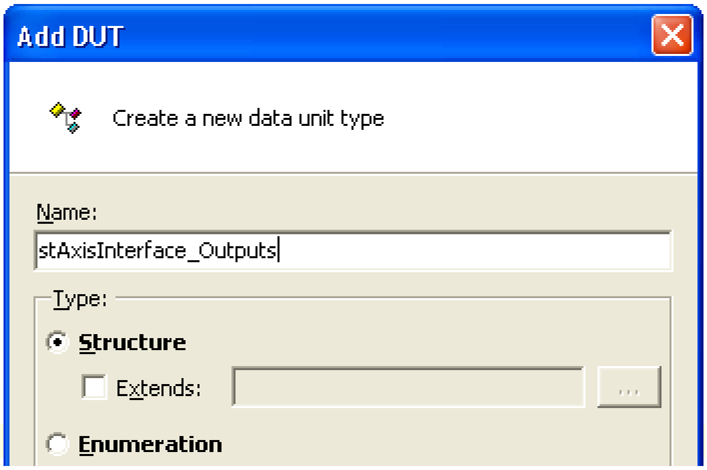
Type:

● Structure

☐ Extends:

○ Enumeration

ii. Create the output structure "**AxisInterface_Outputs**".



Add DUT

Create a new data unit type

Name: stAxisInterface_Outputs

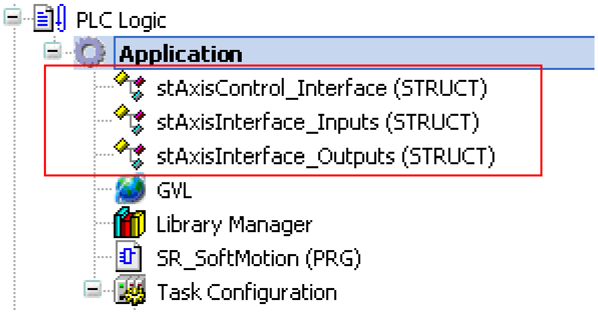
Type:

● Structure

☐ Extends:

○ Enumeration

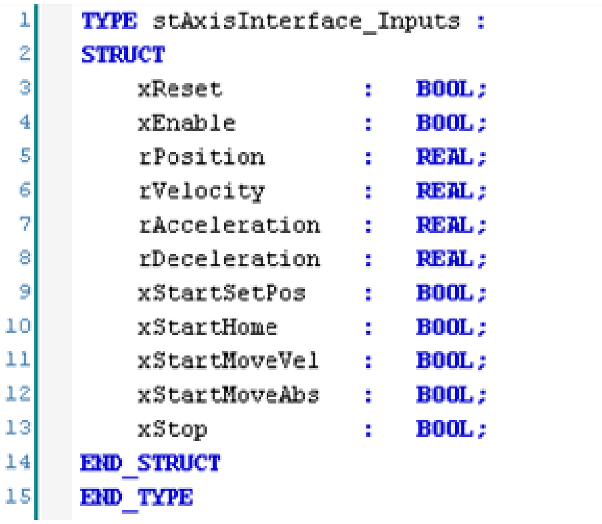
The browser should now contain all 3 new data-types as shown



PLC Logic
  Application
    stAxisControl_Interface (STRUCT)
    stAxisInterface_Inputs (STRUCT)
    stAxisInterface_Outputs (STRUCT)
    GVL
    Library Manager
    SR_SoftMotion (PRG)
    Task Configuration

3. Configure the input structure components

i. Double-click the **stAxisInterface_Inputs** DUT to open the editor, and create the following declaration list.

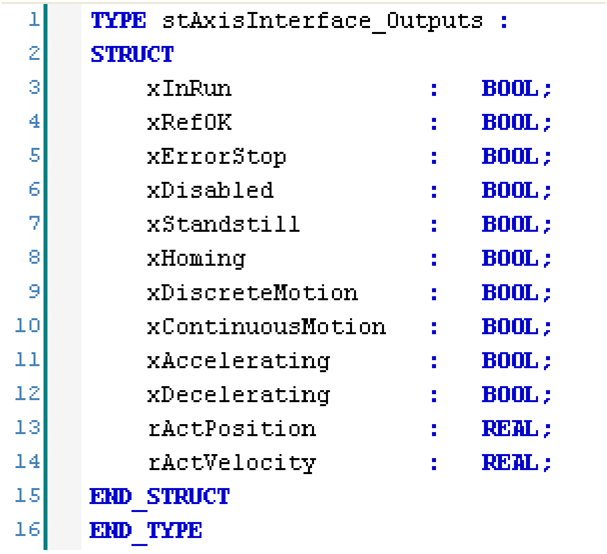


```
TYPE stAxisInterface_Inputs :
STRUCT
    xReset          :   BOOL;
    xEnable         :   BOOL;
    rPosition       :   REAL;
    rVelocity       :   REAL;
    rAcceleration   :   REAL;
    rDeceleration   :   REAL;
    xStartSetPos    :   BOOL;
    xStartHome      :   BOOL;
    xStartMoveVel   :   BOOL;
    xStartMoveAbs   :   BOOL;
    xStop           :   BOOL;
END_STRUCT
END_TYPE
```

---

4. Configure the output structure components

i. Double-click the **stAxisInterface_Outputs** DUT to open the editor, and create the following declaration list.

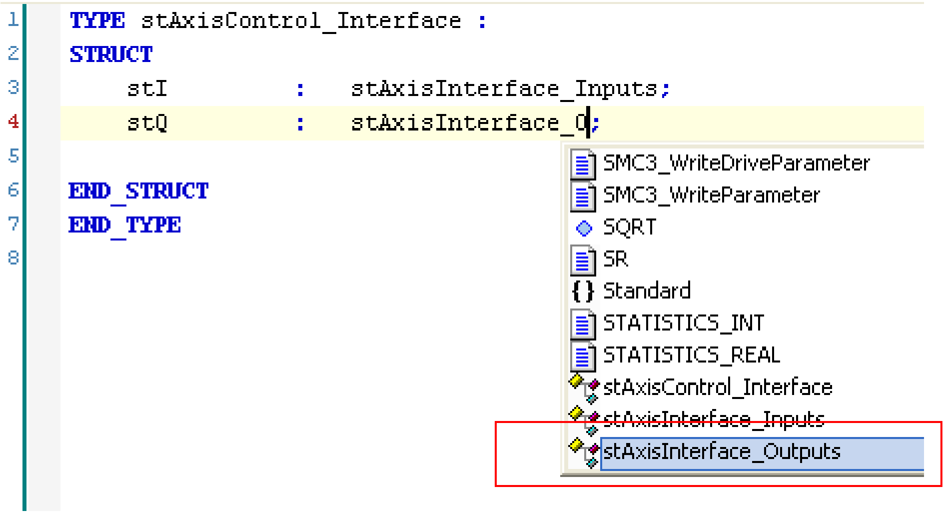


```
TYPE stAxisInterface_Outputs :
STRUCT
    xInRun              :   BOOL;
    xRefOK              :   BOOL;
    xErrorStop          :   BOOL;
    xDisabled           :   BOOL;
    xStandstill         :   BOOL;
    xHoming             :   BOOL;
    xDiscreteMotion     :   BOOL;
    xContinuousMotion   :   BOOL;
    xAccelerating       :   BOOL;
    xDecelerating       :   BOOL;
    rActPosition        :   REAL;
    rActVelocity        :   REAL;
END_STRUCT
END_TYPE
```

5. Configure the Axis Interface structure

i. Open the **stAxisControl_Interface** DUT editor to create the input (stI) and output (stQ) structure component declarations as shown.

Note that the "Smart Code" list will automatically present the appropriate structure variables created previously.



```
TYPE stAxisControl_Interface :
STRUCT
    stI         :   stAxisInterface_Inputs;
    stQ         :   stAxisInterface_O;

END_STRUCT
END_TYPE
```

---

6. Build and save

i. Build the project to confirm that there are no errors.

ii. Save the project as "Interface Structures"

This completes the Exercise

Interface Application

We now have a new structured variable data-type that can be used to control and monitor the behavior of an axis of motion. The new data-type is subdivided with an Input component (.stI), and an Output component (.stQ). Each of these two components contains individual variable data-types that can be used to control or monitor the axis.

There are some distinct advantages to this application of a structure:

- A single “variable” contains all of the elements required for the interface. This improves portability.
- Significant reduction and simplification of variable declarations.
- There is no chance for typing errors, as each of the individual control variables is pre-defined.
- New variable components can easily be added to existing structure, and are immediately available for use throughout the program.

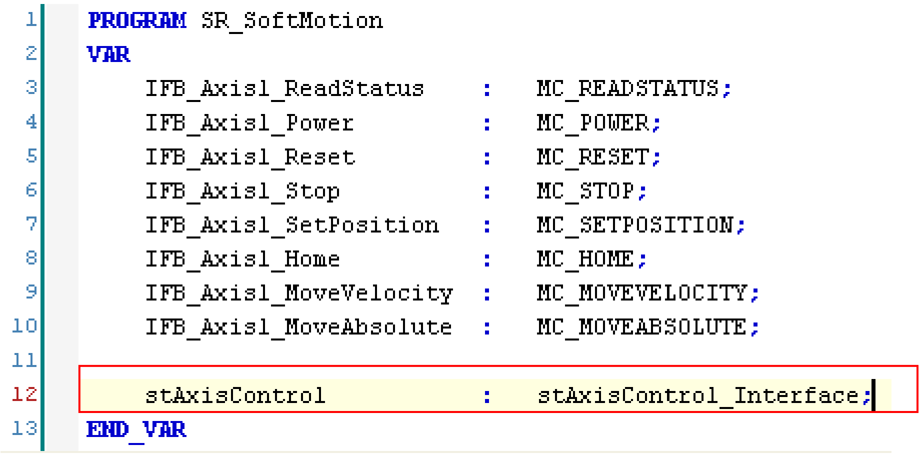
In the next Exercise, we will:

- Create a new **AxisControl** structured variable using the new data-type
- Apply the new structure to the existing SoftMotion FBs in the POU.
- Create a single-variable watch list to completely control and monitor the axis.

Exercise – Apply the Interface Structure to your program

1. Create a new AxisControl variable

i. In the **SR_SoftMotion** variable declaration pane, create a new variable "**stAxisControl**" of type **AxisControl_Interface**.

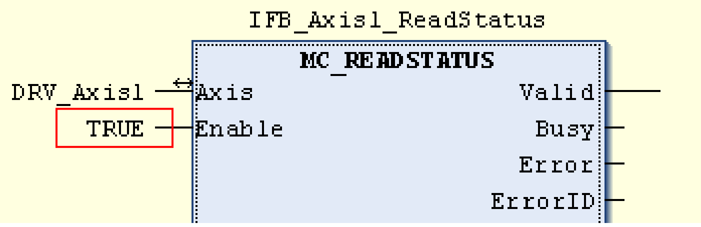


```
PROGRAM SR_SoftMotion
VAR
    IFB_Axis1_ReadStatus     :   MC_READSTATUS;
    IFB_Axis1_Power          :   MC_POWER;
    IFB_Axis1_Reset          :   MC_RESET;
    IFB_Axis1_Stop           :   MC_STOP;
    IFB_Axis1_SetPosition    :   MC_SETPOSITION;
    IFB_Axis1_Home           :   MC_HOME;
    IFB_Axis1_MoveVelocity   :   MC_MOVEVELOCITY;
    IFB_Axis1_MoveAbsolute   :   MC_MOVEABSOLUTE;

    stAxisControl            :   stAxisControl_Interface;
END_VAR
```

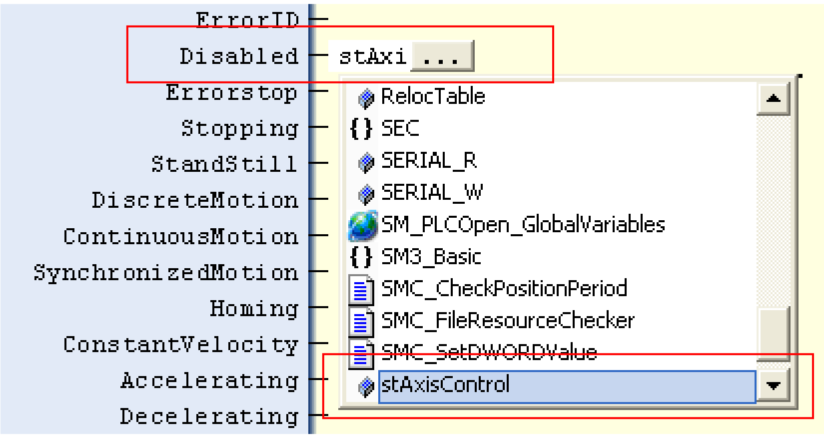
---

2. Apply the Interface variable components to MC_ReadStatus

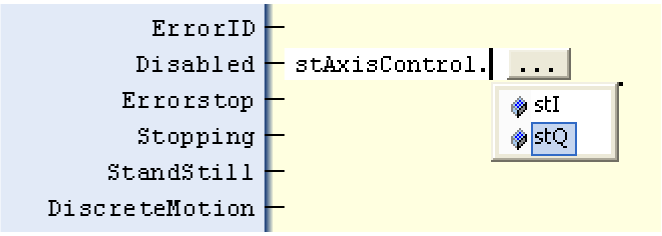
i. Begin by editing the ReadStatus FB. Hardcode the MC_ReadStatus FB **Enable** input **TRUE**.



ii. Apply the new Structure variable to the Disabled output. Click on the Disabled output pin and begin typing the name "stAxisControl"**.** As you type, look for the **stAxisControl** structure in the smart code dropdown, and select it.

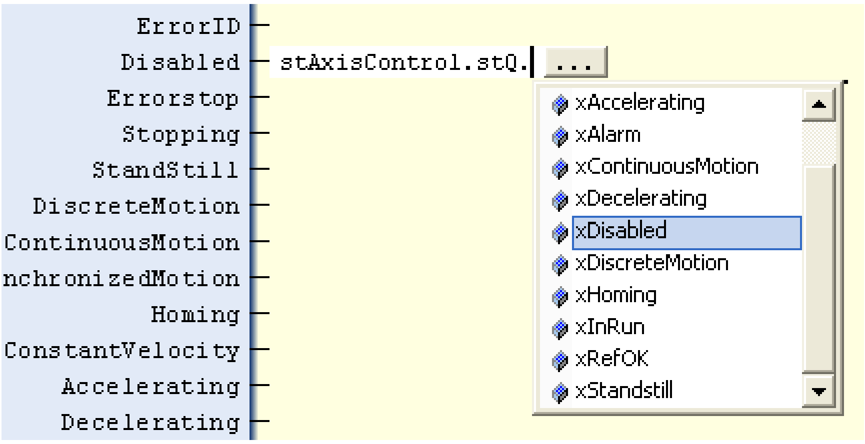


iii. Type a “.” (dot) immediately after the stAxisControl variable. This will bring up the structure components. Scroll to the **stQ** (outputs) component, and select it.



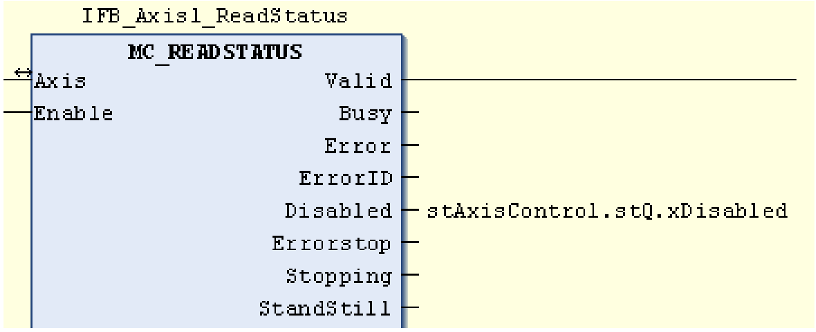
stAxisControl.

iv. Once again type a “dot” to open the **stQ** components, and scroll to **Disabled**.



stAxisControl.stQ.

The completed assignment is illustrated below.



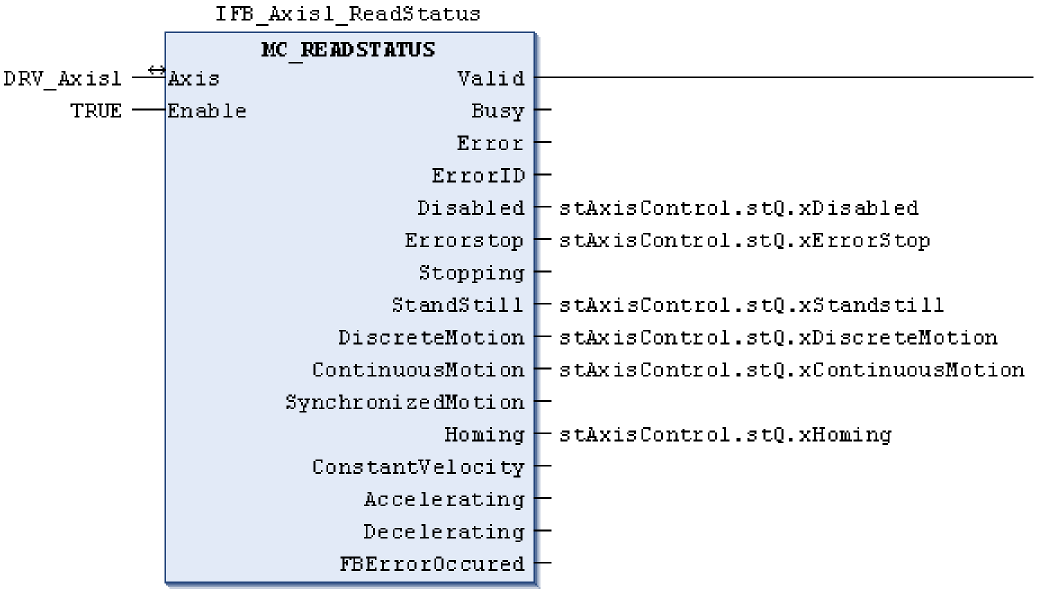
Disabled — stAxisControl.stQ.xDisabled

v. Repeat the is process for each of the PLCopen output state variables

- ErrorStop
- Standstill
- DiscreteMotion
- ContinuousMotion
- Homing

*Hint: copy and paste “**AxisControl.stQ**” from the previous output, and paste it into each of the PLCopen outputs on MC_ReadStatus. Then simply type a “dot” to select the component!*

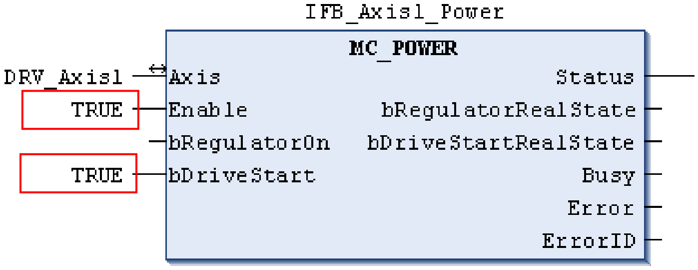
The completed MC_Readstatus output structure assignment is shown below.



---

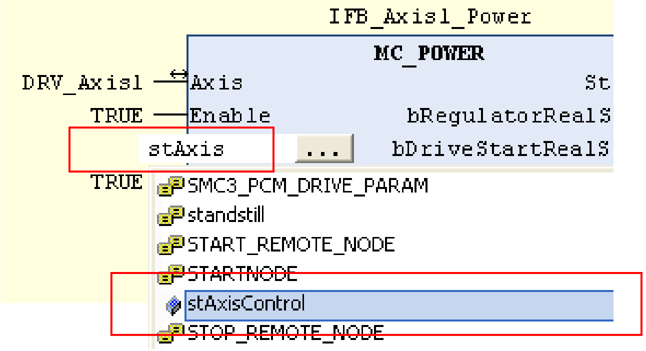
3. Apply the Input control Interface to the FBs

i. Scroll to the MC_Power FB. Hardcode the **Enable** and **bDriveStart** inputs TRUE.

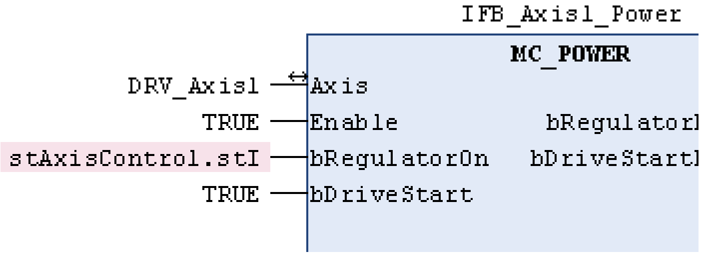


The **bRegulatorOn** input is used to enable the power stage of the drive. We will apply the control variable to this input.

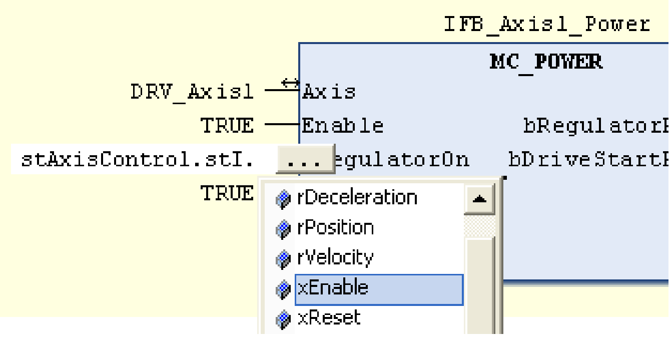
ii. Once again, begin typing “stAxisControl” at the input pin bRegulatorOn. As you type, look for the **stAxisControl** variable lookup, and select it.



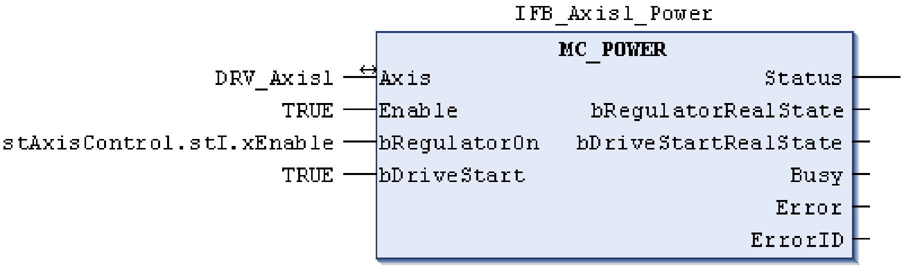
iii. Immediately after the stAxisControl variable, type a “dot” as before, to open and select the **stI** (input) components.



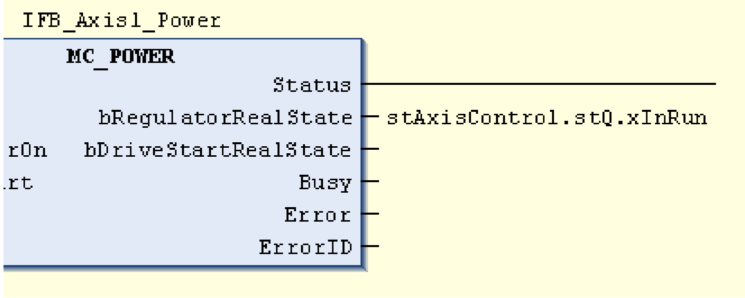
iv. Finally, type the “dot” once again the select the **xEnable** input component.



The completed **MC_Power** input should like the following.



v. Move to the bRegulatorRealState output, and assign the **stQ.** component **xInRun** as shown.

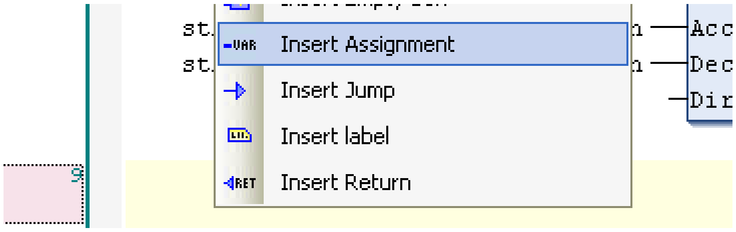


vi. Repeat this process for the remaining FBs making the following assignments

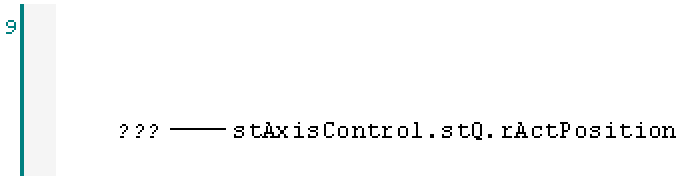
Input (.stl)	**FB assignment**
xReset	MC_Reset.Execute
xStop	MC_Stop.Execute
rDeceleration	MC_Stop.Deceleration
xStartSetPos	MC_SetPosition.Execute
xStartHome	MC_Home.Execute
xStartMoveVel	MC_MoveVelocity.Execute
rVelocity	MC_MoveVelocity.Velocity
rAcceleration	MC_MoveVelocity.Acceleration
rDeceleration	MC_MoveVelocity.Deceleration
xStartMoveAbsolute	MC_MoveAbsolute.Execute
rPosition	MC_MoveAbsolute.Position
rVelocity	MC_MoveAbsolute.Velocity
rAcceleration	MC_MoveAbsolute.Acceleration
rDeceleration	MC_MoveAbsolute.Deceleration

vii. Next we will make use of the Axis_Ref object **DRV_Axis1** to assign the actual position and actual velocity outputs.

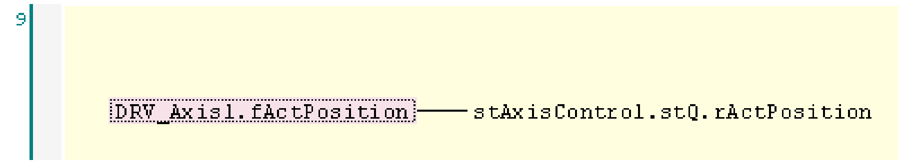
In the **SR_SoftMotion** FBD rick-click on the last network and select **Insert Assignment.**



viii. In the “assign to” field (right hand side) enter the variable component **.rActPosition**.

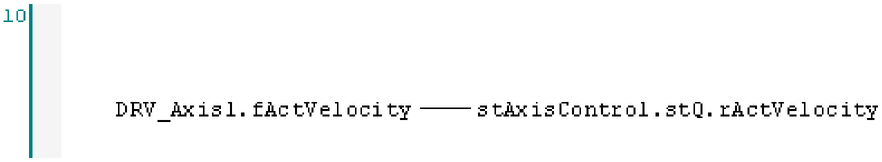


ix. In the “assign from” field, enter the DRV_Axis1 component **.fActPosition**.



```
DRV_Axis1.fActPosition ── stAxisControl.stQ.rActPosition
```

x. Copy and paste the previous network to create the actual Velocity assignment.



```
DRV_Axis1.fActVelocity ── stAxisControl.stQ.rActVelocity
```

---

4. Build and Download

i. Build the project to confirm there are no errors

ii. Save the project

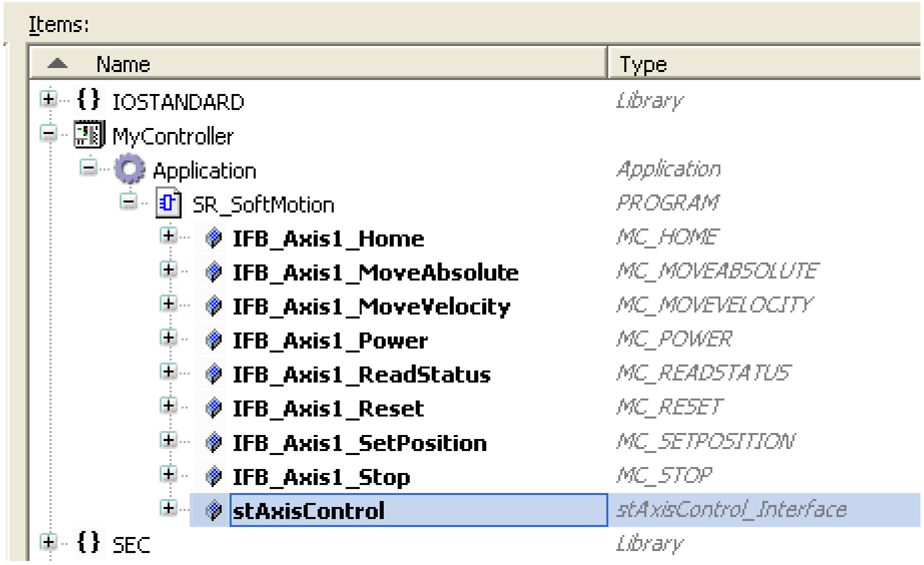
iii. Login as before, and download according to the prompt.

iv. Start the controller at the completion of the download.

---

5. Operate DRV_Axis1 using the interface structure variable

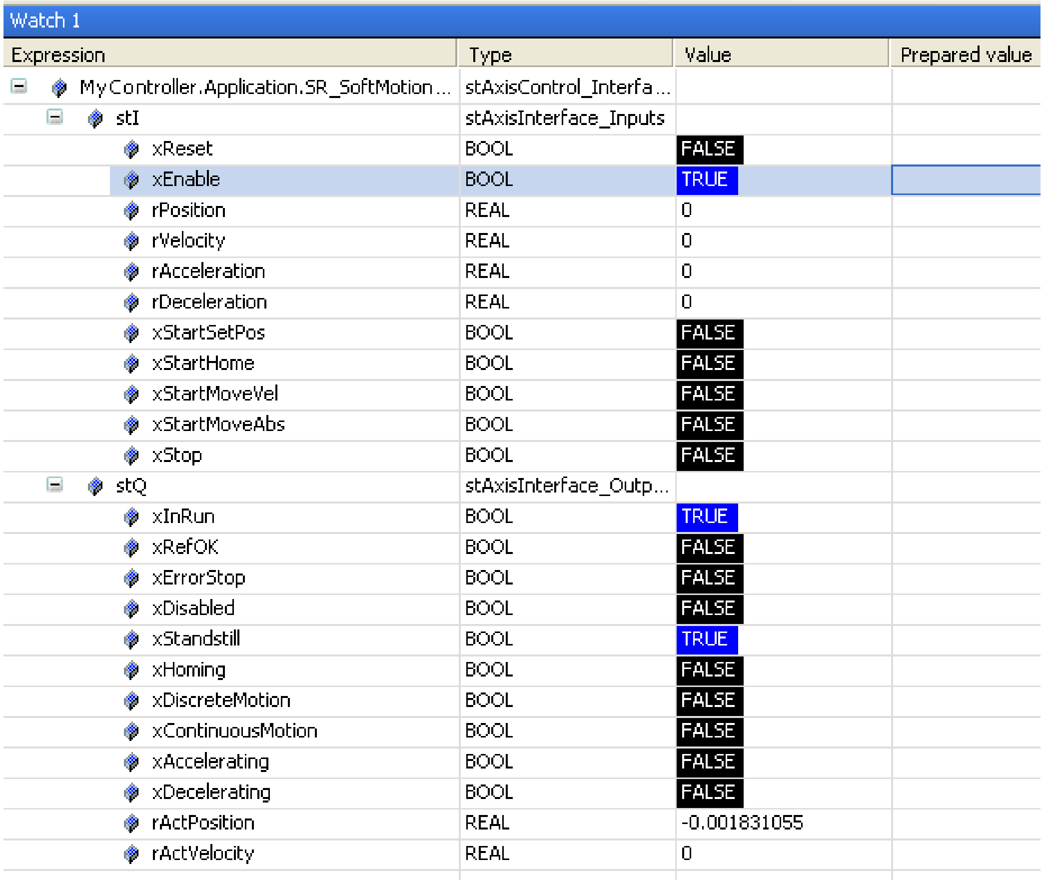
i. Create new watch list, and add the variable “**stAxisControl**” from the Input Assistant screen.



Items:

Name	Type
IOSTANDARD	Library
MyController	
Application	Application
SR_SoftMotion	PROGRAM
IFB_Axis1_Home	MC_HOME
IFB_Axis1_MoveAbsolute	MC_MOVEABSOLUTE
IFB_Axis1_MoveVelocity	MC_MOVEVELOCITY
IFB_Axis1_Power	MC_POWER
IFB_Axis1_ReadStatus	MC_READSTATUS
IFB_Axis1_Reset	MC_RESET
IFB_Axis1_SetPosition	MC_SETPOSITION
IFB_Axis1_Stop	MC_STOP
stAxisControl	stAxisControl_Interface
SEC	Library

ii. Open the + signs to reveal the structure components.



Watch 1

Expression	Type	Value	Prepared value
MyController.Application.SR_SoftMotion...	stAxisControl_Interfa...		
stI	stAxisInterface_Inputs		
xReset	BOOL	FALSE	
xEnable	BOOL	TRUE	
rPosition	REAL	0	
rVelocity	REAL	0	
rAcceleration	REAL	0	
rDeceleration	REAL	0	
xStartSetPos	BOOL	FALSE	
xStartHome	BOOL	FALSE	
xStartMoveVel	BOOL	FALSE	
xStartMoveAbs	BOOL	FALSE	
xStop	BOOL	FALSE	
stQ	stAxisInterface_Outp...		
xInRun	BOOL	TRUE	
xRefOK	BOOL	FALSE	
xErrorStop	BOOL	FALSE	
xDisabled	BOOL	FALSE	
xStandstill	BOOL	TRUE	
xHoming	BOOL	FALSE	
xDiscreteMotion	BOOL	FALSE	
xContinuousMotion	BOOL	FALSE	
xAccelerating	BOOL	FALSE	
xDecelerating	BOOL	FALSE	
rActPosition	REAL	-0.001831055	
rActVelocity	REAL	0	

iii. Operate and Monitor the axis exclusively from the AxisControl structure in the Watch list using Prepared values.

This completes the Exercise.

End of Day1

Chapter 6: Machine Control with SoftStruXure

In this Chapter, we take the idea of re-usable code and object-oriented methods a step further. Now, the entire machine is a preconfigured object, with built-in hardware and functional mapping. Furthermore, the SoftMotion FBs are replaced by a single AxisModule, and the interface structures become a bit larger, more complete, and more informative!

Every machine has unique characteristics and requirements. However, every machine has many similarities in form and function that can be exploited in the development of machine operational code.

Functionality that is likely to be found in all machines includes:

- Initialization / Preparation / Homing
- Managing hardware Inputs / Operator commands
- Managing hardware Outputs
- Mode Control / Axis control
- Alarm Detection and Reaction Handling

Unique functionality of a machine includes:

- The specific sequence of machine processes
- Exception handling (product present / product absent)
- User-specific Alarms

SoftStruXure is a project template that has been designed to promote each of these characterisitics. An overview of the template and interactive exercises, are offered in the following sections.

For a complete description of the template functionality and use, please refer to the SoftStruXure Machine Template User Guide V1.0.0.1.

SoftStruXure Overview

The development of machine code can be summarized as the “mapping” of hardware and functionality. Hardware mapping includes predominately configuration-based activities such as hardware I/O assignments, motion bus and axis configuration, drives, encoders and communication network configuration.

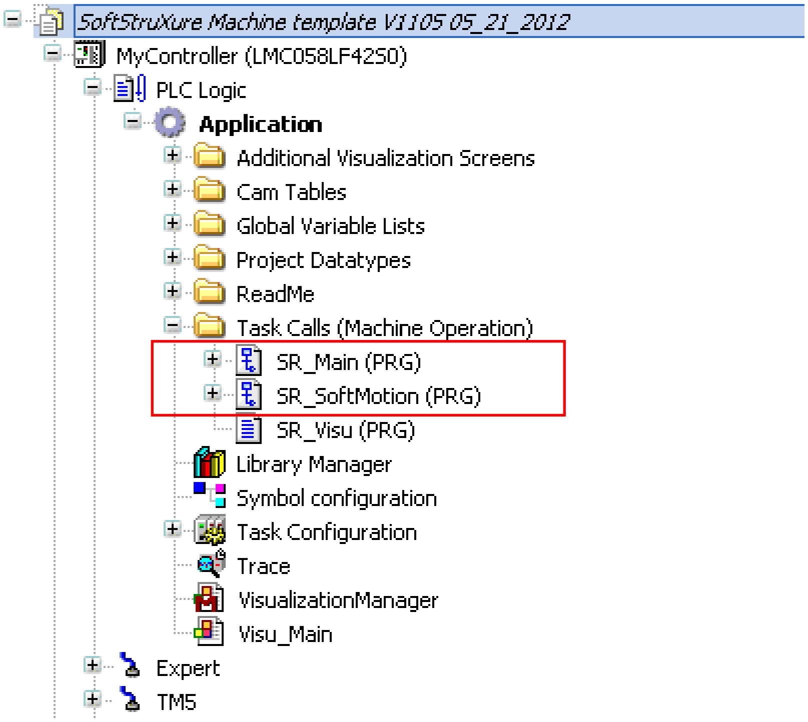
“Mapping the functionality” can be thought of as the development of the substantive code that performs the unique machine process and exception handling. This programming effort includes essential administrative functions required of nearly all machines independent of the specific task for which the machine was created.

The SoftStruXure Machine template provides a unique programming foundation for a machine solution using multi-axis motion control with the LMC058 controllers and Lexium32 (LXM32) servo drives. Based largely upon a similar architecture implemented by Elau for the EPAS software, machine I/O, a master encoder, CANopen and CANmotion bus are all pre-mapped. Four CANmotion servo axes and a variety of CANopen devices are configured with an extensive SDO startup list to support Fast Device Replacement (FDR). AxisModules and interface structures replace individual SoftMotion FB instances, and basic machine functionality is provided including mode selection, I/O management, and alarm handling at the axis and at the operator level.

A quick look at the SoftStruXure browser will help illustrate some of the features

Browser at a Glance

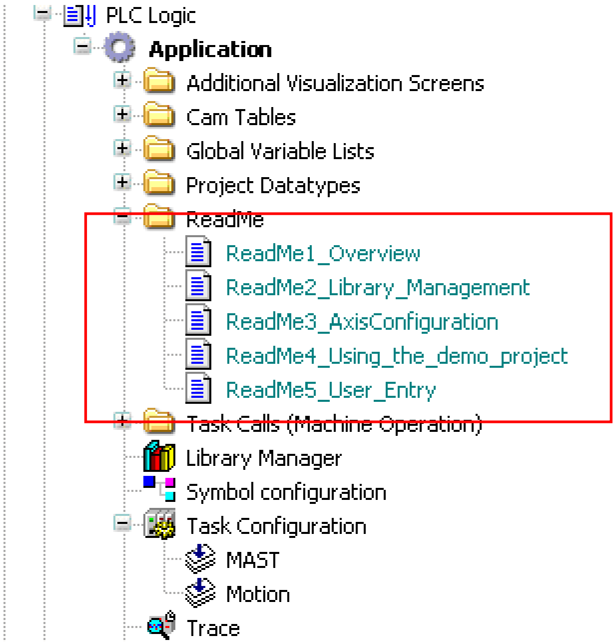
As indicated in the SoftStruXure browser, the machine process is driven by two task calls. **SR_SoftMotion** is associated with the motion task, and contains the instantiations for all CANmotion AxisModules for each of the virtual and physical axes. **SR_Main** is the main machine POU, and is called from the MAST task.



- SoftStruXure Machine template V1105 05_21_2012
  - MyController (LMC058LF42S0)
    - PLC Logic
      - **Application**
        - Additional Visualization Screens
        - Cam Tables
        - Global Variable Lists
        - Project Datatypes
        - ReadMe
        - Task Calls (Machine Operation)
          - SR_Main (PRG)
          - SR_SoftMotion (PRG)
          - SR_Visu (PRG)
        - Library Manager
        - Symbol configuration
        - Task Configuration
        - Trace
        - VisualizationManager
        - Visu_Main
    - Expert
    - TM5

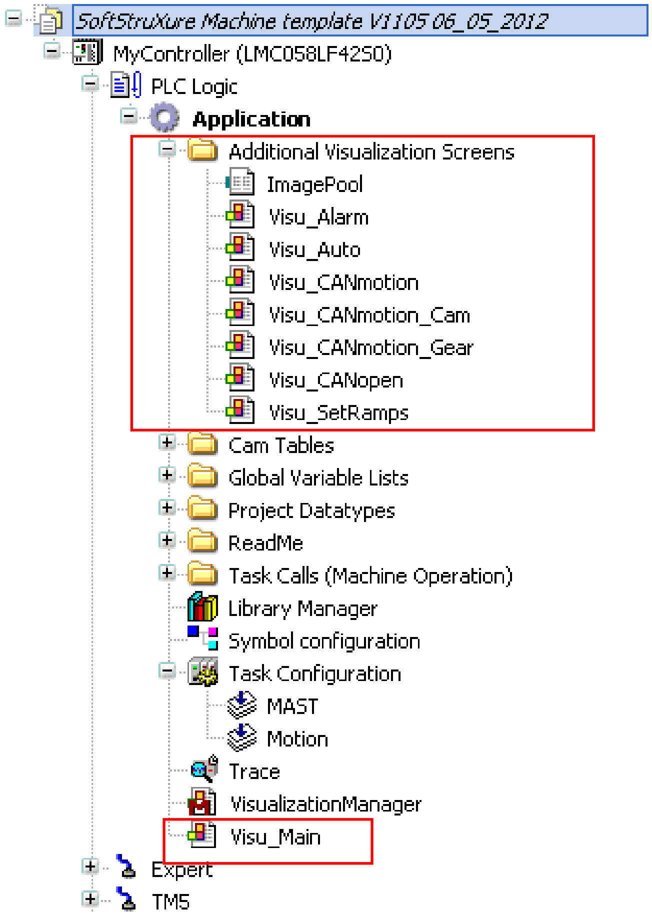
The **SR_Visu** task call is used primarily to manage display functionality for the included SoMachine Visualization screens.

A **Read-Me** folder contains text-based documentation on the basic features and use.



- PLC Logic
  - **Application**
    - Additional Visualization Screens
    - Cam Tables
    - Global Variable Lists
    - Project Datatypes
    - ReadMe
      - ReadMe1_Overview
      - ReadMe2_Library_Management
      - ReadMe3_AxisConfiguration
      - ReadMe4_Using_the_demo_project
      - ReadMe5_User_Entry
    - Task Calls (Machine Operation)
    - Library Manager
    - Symbol configuration
    - Task Configuration
      - MAST
      - Motion
    - Trace

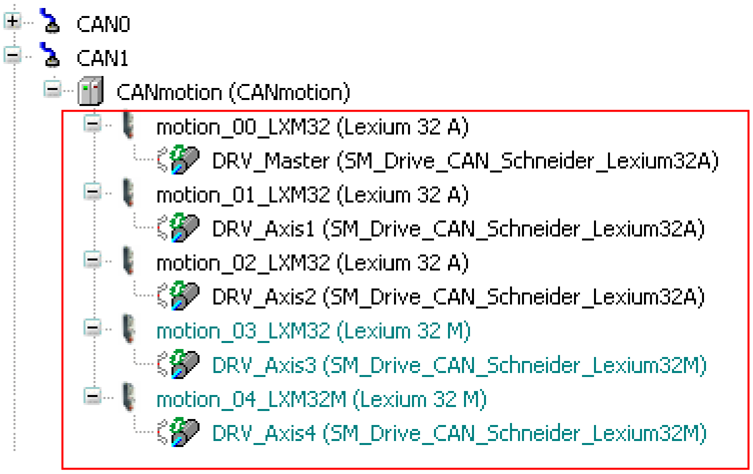
The template includes generic visualization screens that can be used to demonstrate the basic operator commands, independent and synchronized axis control, and a pre-configured auto mode pick-n-place application. The initial operator interface is “**Visu_Main**”. Supplemental screens, referenced by the main screen at run time, are located in the **Additional Visualization Screens** folder as shown.



- SoftStruXure Machine template V1105 06_05_2012
  - MyController (LMC058LF42S0)
    - PLC Logic
      - **Application**
        - Additional Visualization Screens
          - ImagePool
          - Visu_Alarm
          - Visu_Auto
          - Visu_CANmotion
          - Visu_CANmotion_Cam
          - Visu_CANmotion_Gear
          - Visu_CANopen
          - Visu_SetRamps
        - Cam Tables
        - Global Variable Lists
        - Project Datatypes
        - ReadMe
        - Task Calls (Machine Operation)
        - Library Manager
        - Symbol configuration
        - Task Configuration
          - MAST
          - Motion
        - Trace
        - VisualizationManager
        - Visu_Main
    - Expert
    - TM5

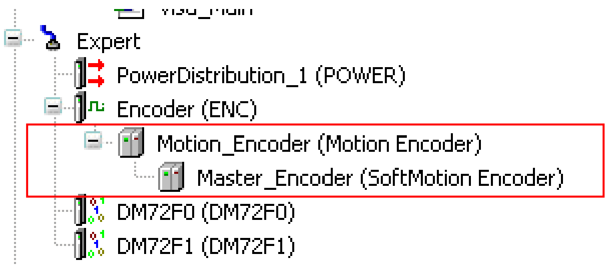
Hardware Map

The SoftStruXure template is pre-mapped with 5 CANmotion servo axes including a virtual master (DRV_Master). Two of the axes are excluded from build, but available if required.



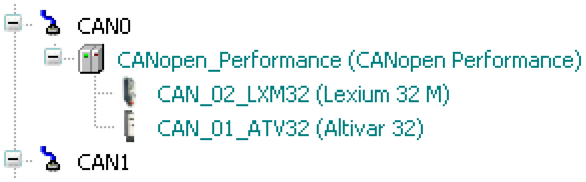
- CAN0
- CAN1
  - CANmotion (CANmotion)
    - motion_00_LXM32 (Lexium 32 A)
      - DRV_Master (SM_Drive_CAN_Schneider_Lexium32A)
    - motion_01_LXM32 (Lexium 32 A)
      - DRV_Axis1 (SM_Drive_CAN_Schneider_Lexium32A)
    - motion_02_LXM32 (Lexium 32 A)
      - DRV_Axis2 (SM_Drive_CAN_Schneider_Lexium32A)
    - motion_03_LXM32 (Lexium 32 M)
      - DRV_Axis3 (SM_Drive_CAN_Schneider_Lexium32M)
    - motion_04_LXM32M (Lexium 32 M)
      - DRV_Axis4 (SM_Drive_CAN_Schneider_Lexium32M)

The hardware map includes a SoftMotion encoder mapped to the high density D-sub connector. The Softmotion encoder is required as an optional master axis Input/ Output interface for the AxisModules.



- Expert
  - PowerDistribution_1 (POWER)
  - Encoder (ENC)
    - Motion_Encoder (Motion Encoder)
      - Master_Encoder (SoftMotion Encoder)
  - DM72F0 (DM72F0)
  - DM72F1 (DM72F1)

Two CANopen drives including 1 LXM32 axis, and 1 ATV axis, are also pre-mapped and excluded from build.

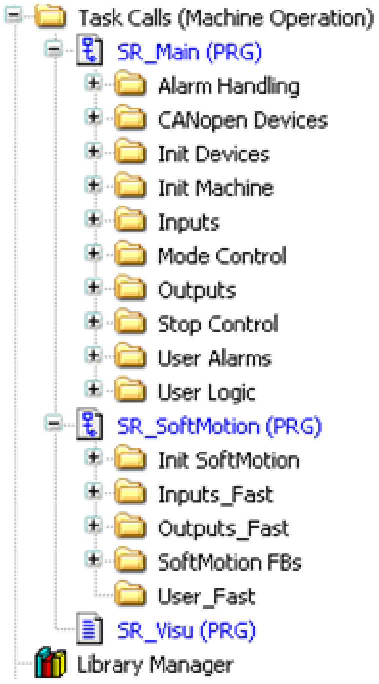


- CAN0
  - CANopen_Performance (CANopen Performance)
    - CAN_02_LXM32 (Lexium 32 M)
    - CAN_01_ATV32 (Altivar 32)
- CAN1

Functional Map

The task calls, SR_Main and SR_SoftMotion, are the principle elements of the Functional Map. SR_Main supports the basic machine functionality, and SR_SoftMotion manages the SoftMotion instantiations and fast User functions. A folder structure is used to organize the action calls beneath each of these Programs.

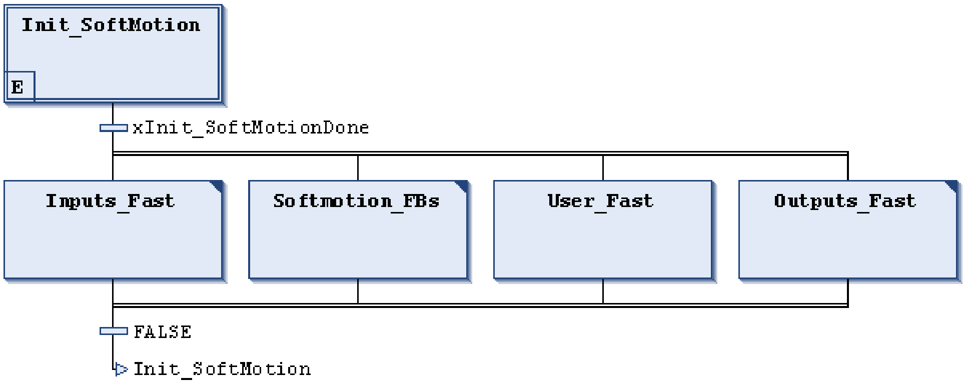
SR_Visu is a separate program that manages custom visualization screens for demo purposes, as well as machine commissioning.



- Task Calls (Machine Operation)
  - SR_Main (PRG)
    - Alarm Handling
    - CANopen Devices
    - Init Devices
    - Init Machine
    - Inputs
    - Mode Control
    - Outputs
    - Stop Control
    - User Alarms
    - User Logic
  - SR_SoftMotion (PRG)
    - Init SoftMotion
    - Inputs_Fast
    - Outputs_Fast
    - SoftMotion FBs
    - User_Fast
  - SR_Visu (PRG)
- Library Manager

SR_SoftMotion

SR_Softmotion is organized into SFC steps (“containers”) which hold the Softmotion FB instantiation, and manage any fast logic requirements of the machine. As seen by the final FALSE transition in the SFC chart, all SFC steps remain active for the duration of operation once the Initialization has completed.



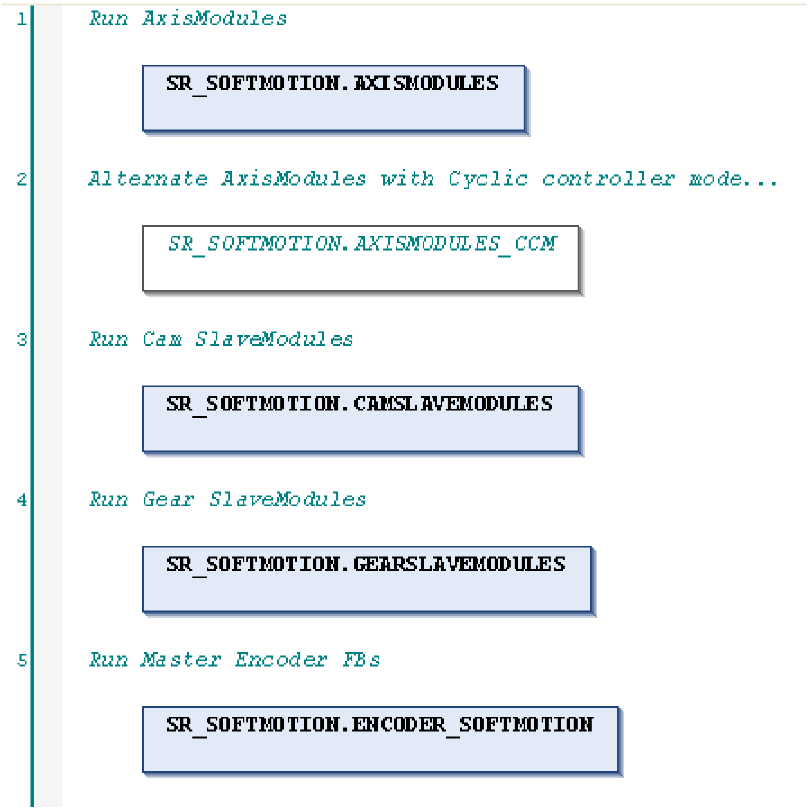
A description of the **SR_SoftMotion** SFC containers is summarized in the following table.

SFC “Container”	**Description**
Init_SoftMotion	Initialize SoftMotion axis pointers and assignments
Inputs_Fast	Manage and buffer the global and hardware mapped inputs (FDI_xx) to internal variables
SoftMotion_FBs	Instantiation of all CANmotion AxisModules
User_Fast	Fast user logic as required (initially empty)
Outputs	Buffer and apply internal variables to global and hardware outputs (FDQ_xx)

For a complete description of the functionality contained within SR_SoftMotion, please refer to the SoftStruXure Machine Template User Guide V1.0.0.1.

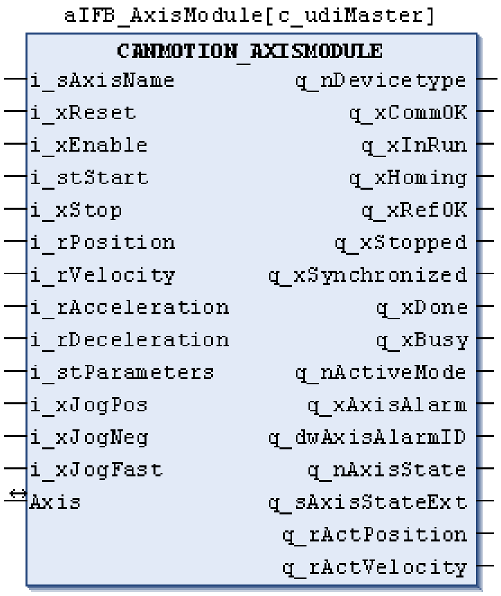
SoftMotion_FBs The SoftMotion_FBs step manages the instantiation of the required independent and synchronized (Master Slave) motion control functions for each of the configured axes on CANmotion.

The step contains action calls in FBD as shown, and these can be activated or deactivated as needed by “toggling the network comment state”.



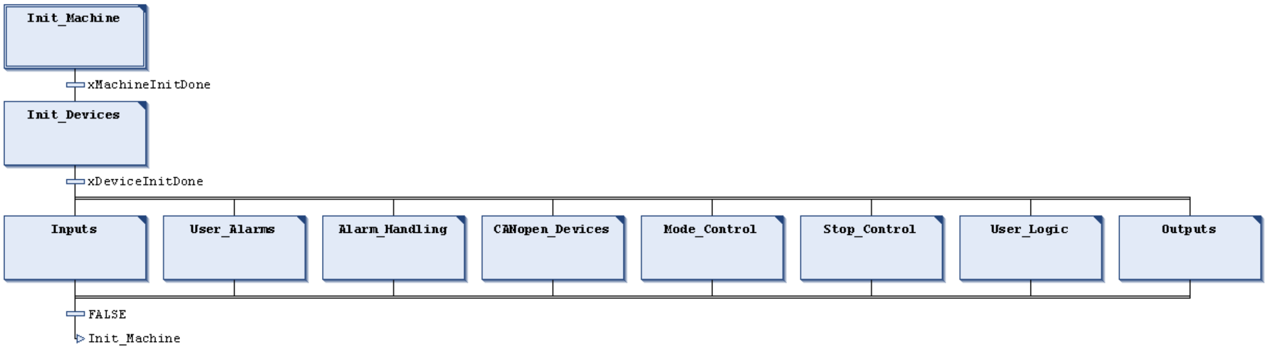
```
1   Run AxisModules
        SR_SOFTMOTION.AXISMODULES

2   Alternate AxisModules with Cyclic controller mode...
        SR_SOFTMOTION.AXISMODULES_CCM

3   Run Cam SlaveModules
        SR_SOFTMOTION.CAMSLAVEMODULES

4   Run Gear SlaveModules
        SR_SOFTMOTION.GEARSLAVEMODULES

5   Run Master Encoder FBs
        SR_SOFTMOTION.ENCODER_SOFTMOTION
```

AxisModules In the template, unique SoftMotion FB instances are replaced by AxisModules. These AxisModules internally manage 13 SoftMotion function blocks including those required for synchronous and asynchronous alarm handling.



SR_Main

Using the concept and programming methods of the EPAS template, **SR_Main** is organized into SFC steps (“containers”) each of which manages the corresponding property of the machine. As seen by the final FALSE transition in the SFC chart, all SFC steps remain active for the duration of operation once the Initialization has completed.



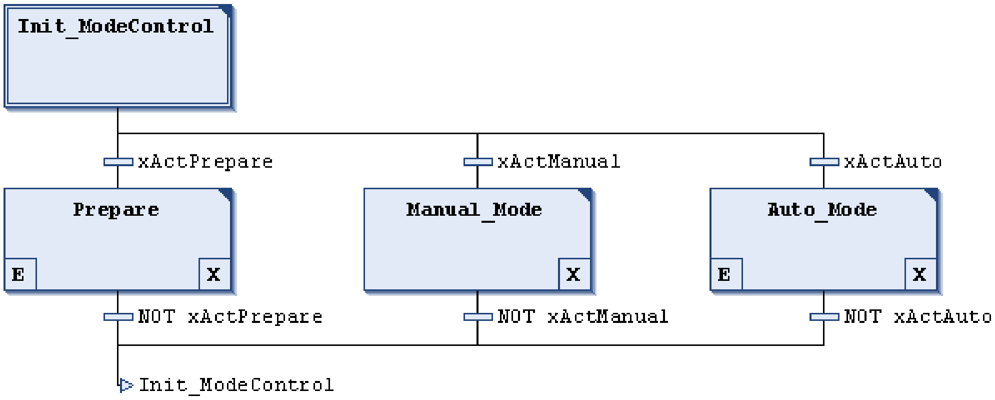
A brief description of the SR_Main “containers” and the corresponding function is summarized in the following table.

SFC “Container”	**Description**
Init_Machine	Initialize machine and axis profile parameters
Init_Devices	Initialize CANmotion and CANopen Axis parameters
Inputs	Manage and buffer the global and hardware mapped inputs (DI_xx) to internal variables
User_Alarms	Create user–specified alarms and reactions. The Alarm Handler FBs are included in the library SoftStruXure_AlarmHandler_SoMV3.
Alarm_Handling	Manage the display and acknowledgement of active alarms
CANopen_Devices	CANopen Device instantiation and CAN device status (including CANmotion devices)
Mode_Control	Mode selector for Prepare, Auto, and Manual operating modes
Stop_Control	Manages the Stop reactions based on the User alarms
User_Logic	User logic as dependent upon the operating mode
Outputs	Buffer and apply internal variables to global and hardware outputs (DQ_xx)

For a complete description of the functionality contained within SR_Main, please refer to the SoftStruXure Machine Template User Guide V1.0.0.1.

Mode Control

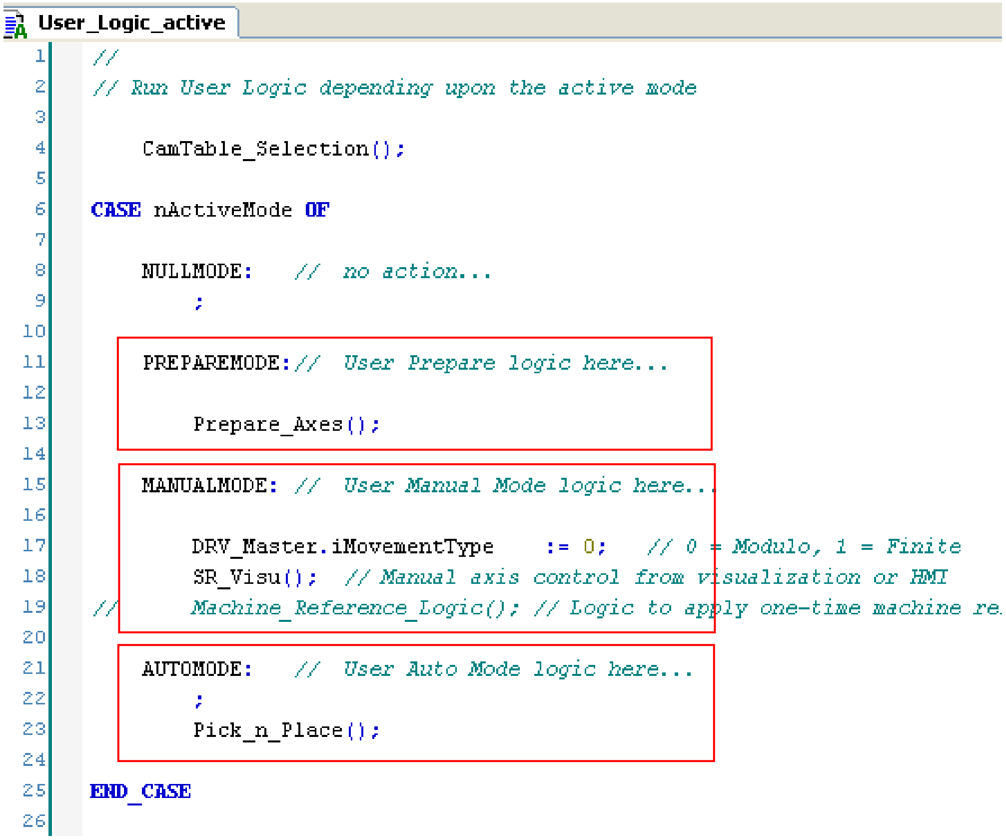
Often machines require the implement of multiple operating modes. The template provides Mode Control functionality in the form of a basic Mode handler. Logic within the init step evaluates machine conditions, including inputs, to determine whether or not to move to one of several possible machine operating modes. Within the operating mode, logic within that mode determines the appropriate conditions (including an active alarm) to exit the mode and return to the Init step.



These modes can be edited as needed by the user.

User Logic

Within the **SR_Main >> User_Logic** step, a simple CASE statement is used to manage the specific logic that solved within each operating mode.

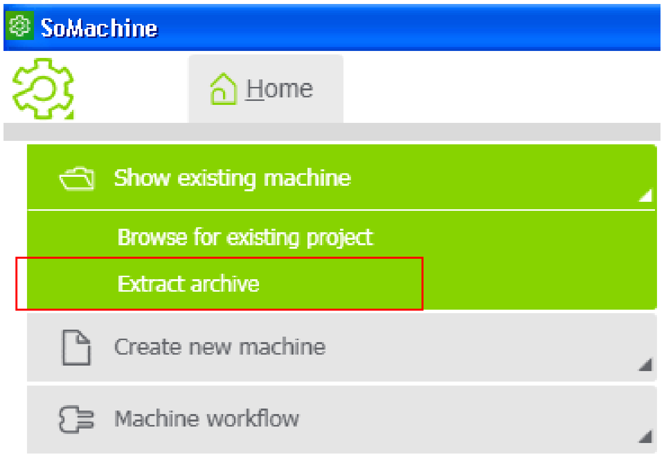


```
//
// Run User Logic depending upon the active mode

    CamTable_Selection();

CASE nActiveMode OF

    NULLMODE:   //  no action...
        ;

    PREPAREMODE://  User Prepare logic here...

        Prepare_Axes();

    MANUALMODE: //  User Manual Mode logic here...

        DRV_Master.iMovementType    := 0;   // 0 = Modulo, 1 = Finite
        SR_Visu();  // Manual axis control from visualization or HMI
//      Machine_Reference_Logic(); // Logic to apply one-time machine re

    AUTOMODE:   //  User Auto Mode logic here...
        ;
        Pick_n_Place();

END_CASE

```

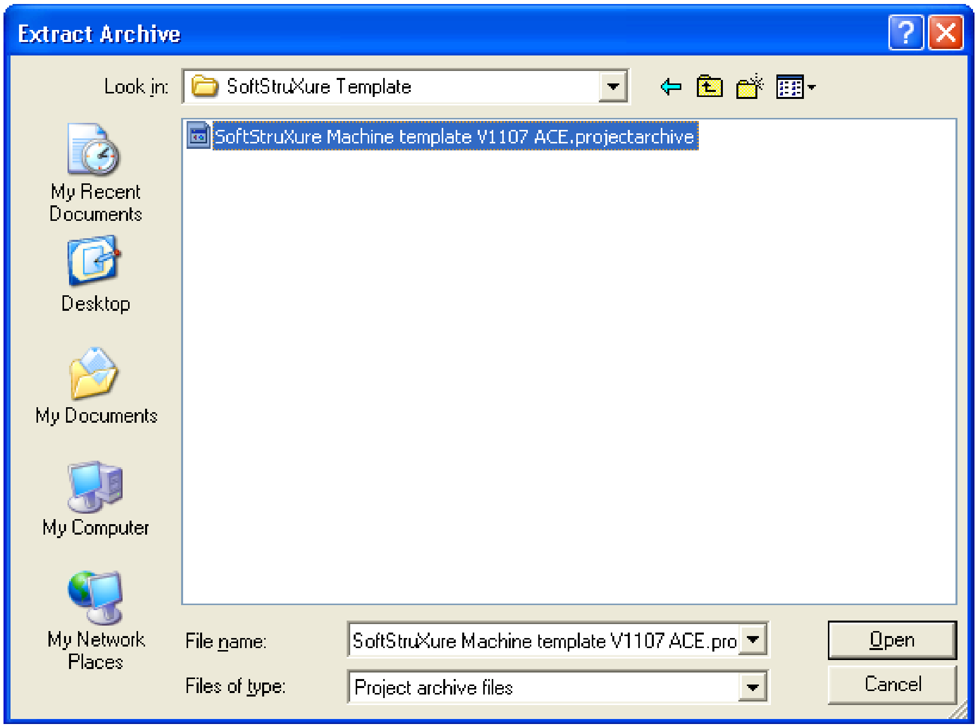
Exercise – Operate the SoftStruXure template

1. Open the project archive

i. From the SoMachine Home screen, select Extract archive.

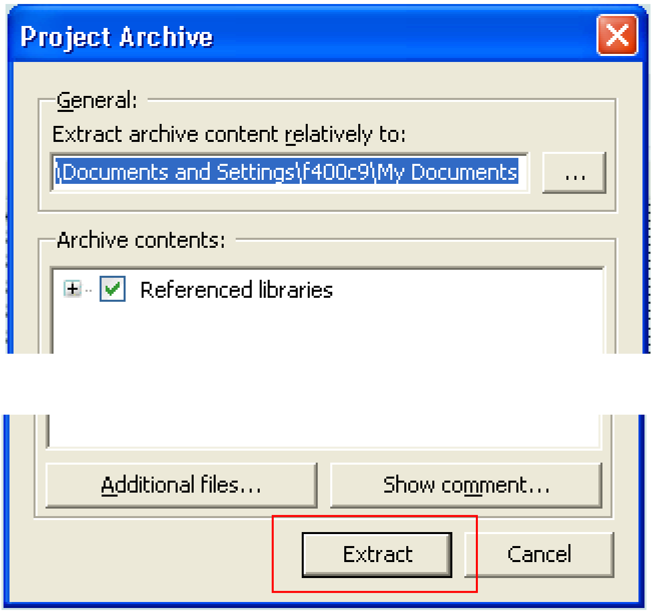


ii. Browse to the desktop training folder **ACE University Motion with SoMachine >> SoftStruXUre Template**, and select the project archive “SoftStruXure Machine Template V1107 ACE.projectarchive”



iii. Click **OK** to extract.

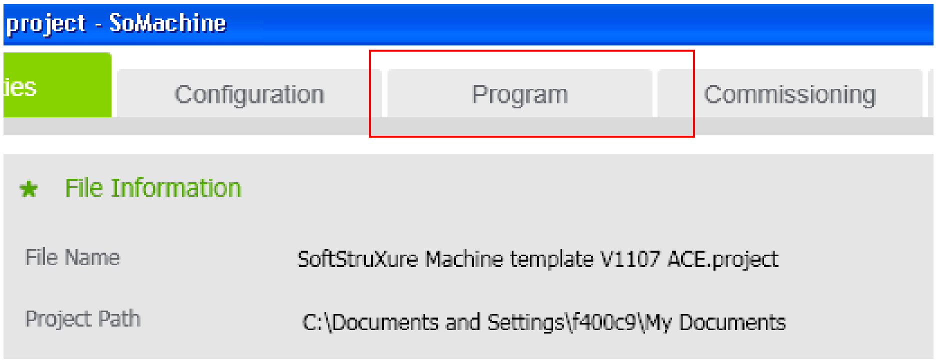
iii. Select **Extract** at the Prompt.



Extract archive content relatively to: \Documents and Settings\f400c9\My Documents

☑ Referenced libraries

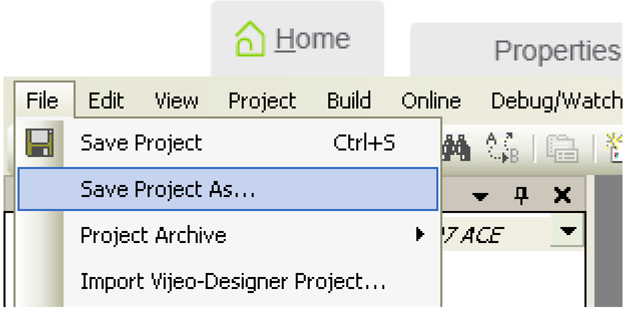
iv. When the extraction has completed, select the **Program** tab at the main screen.



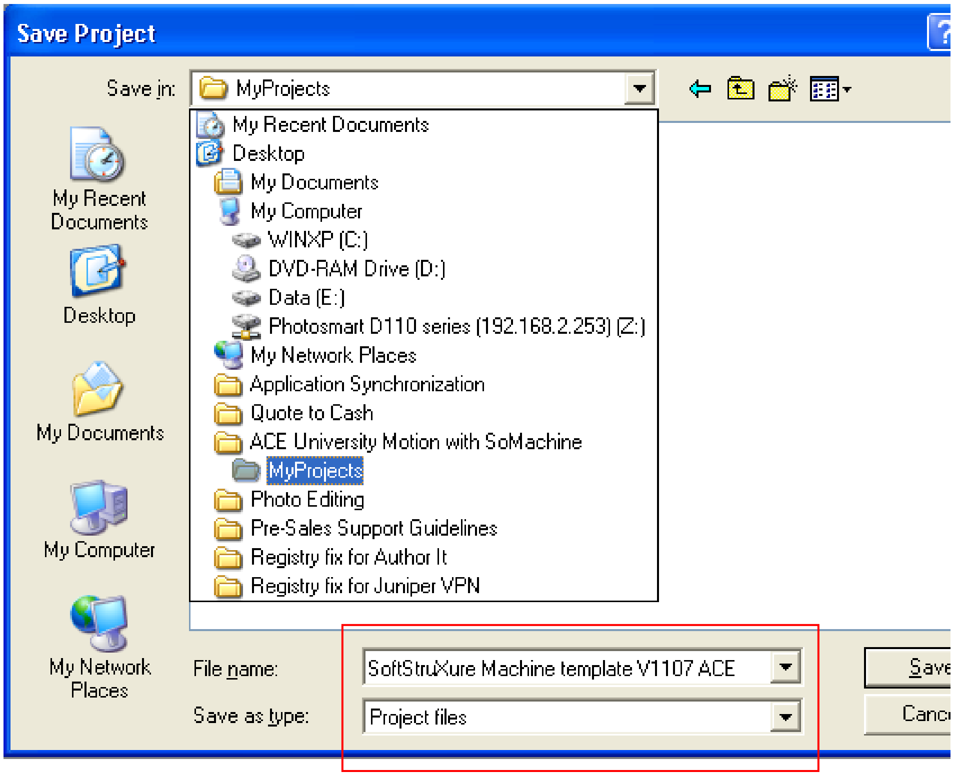
File Name SoftStruXure Machine template V1107 ACE.project

Project Path C:\Documents and Settings\f400c9\My Documents

v. Select “**Save Project As…**“ from the top level menu.



vi. Save the project to the folder, **ACE University Motion with SoMachine >> MyProjects**



Save in: MyProjects

File name: SoftStruXure Machine template V1107 ACE

Save as type: Project files

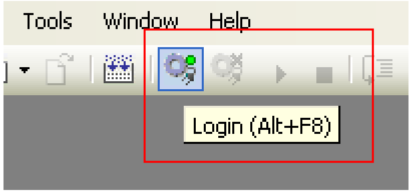
*Make sure that the file type "**Project files**" is chosen.*

---

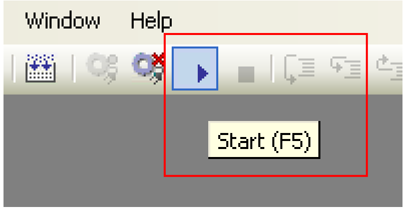
2. Connect to the Controller and Download

i. Connect to the controller using Ethernet. If necessary, delete and replace the existing Gateway.

ii. **Login** to the controller, and download the project at the prompt.

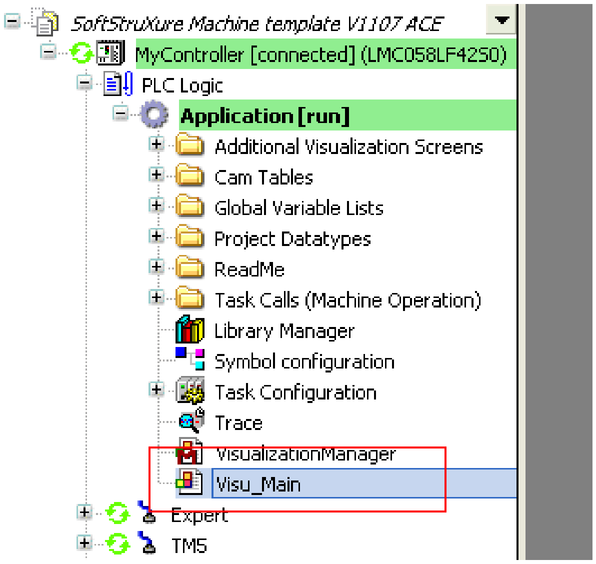


iii. Select Start at the completion of the download.

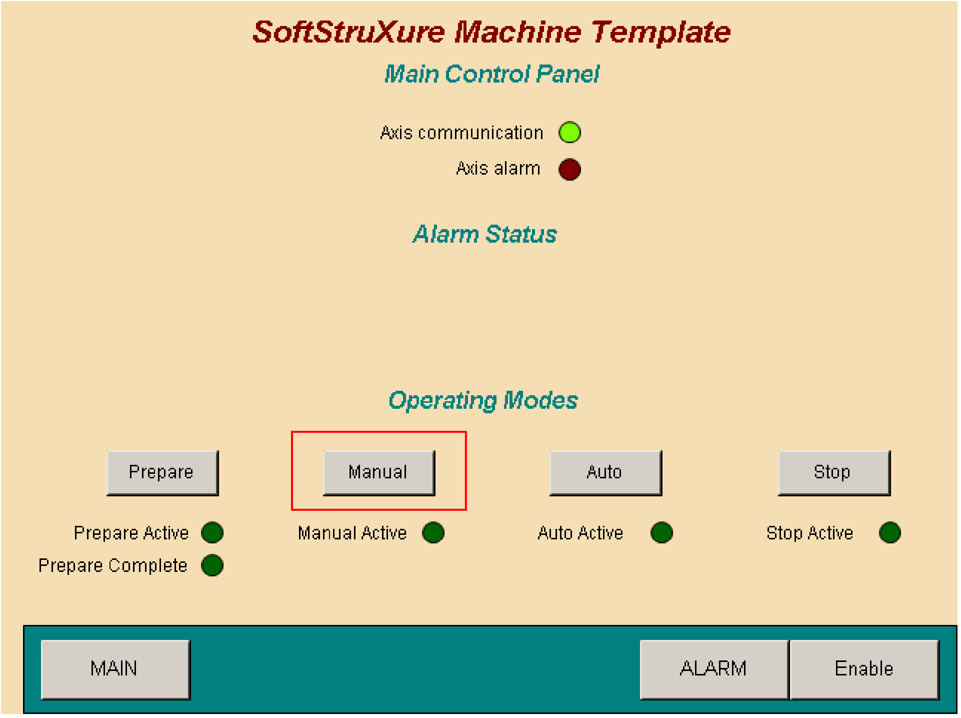


3. Access Manual Mode Control

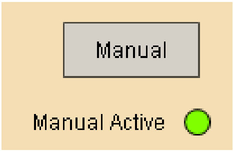
i. Expand the browser, and double-click **Visu_Main** to open the main visualization control screen.



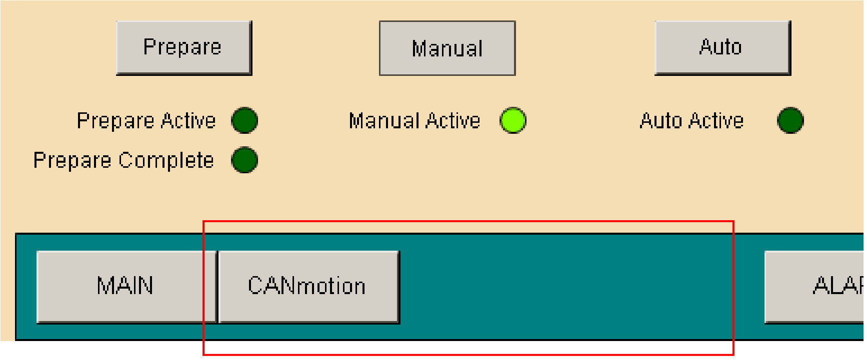
ii. From the main control screen, press the **Manual** button to select Manual Mode.



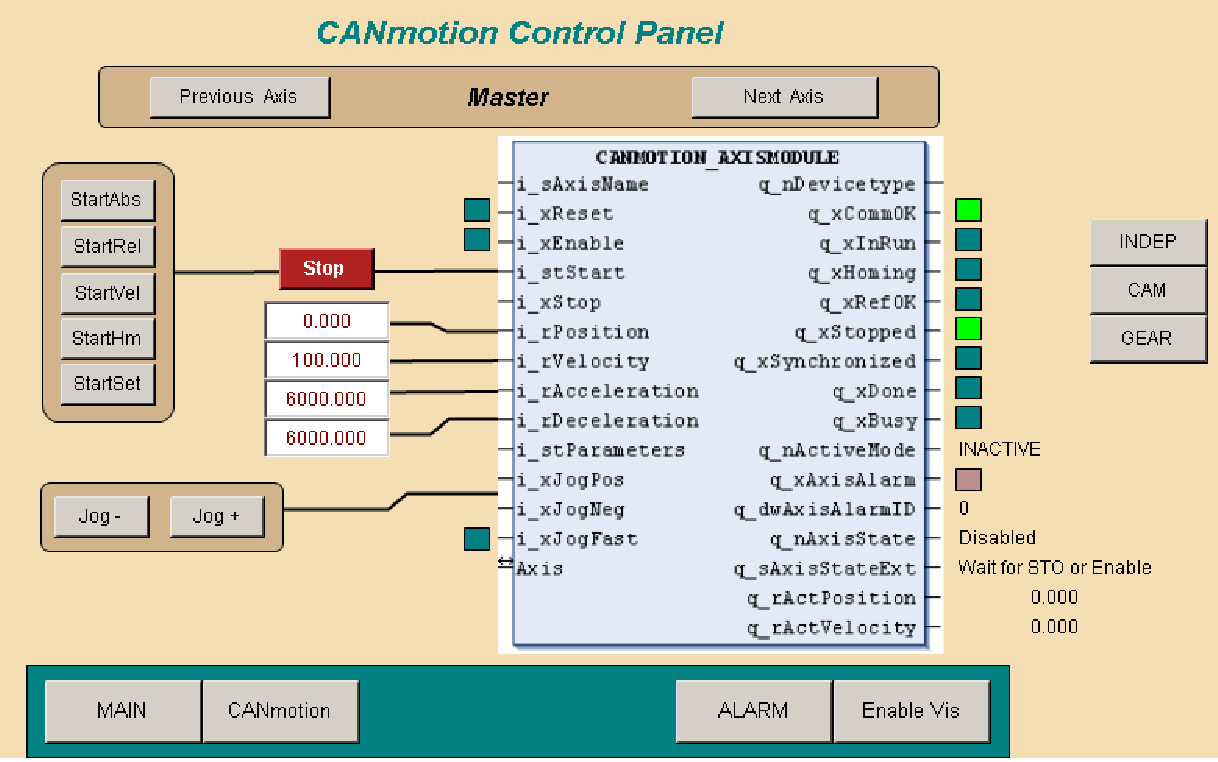
The **Manual Active** light will turn ON in Manual mode.



In addition, the Manual control screen selection button (CANmotion/CANopen) will appear at the bottom menu bar.



iv. Select **CANmotion** to open the CANmotion device control screen.

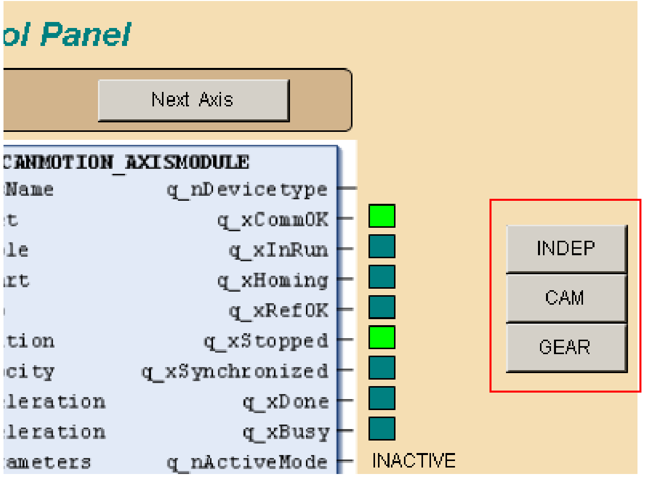


The CANmotion control screen displays the AxisModule for the Virtual Master axis, as well as representative inputs to control basic functionality.

From this screen the axis can be operated manually with PLCopen output status updated along with the axis position and velocity in real-time.

v. Use the Axis navigation buttons to select additional axes.

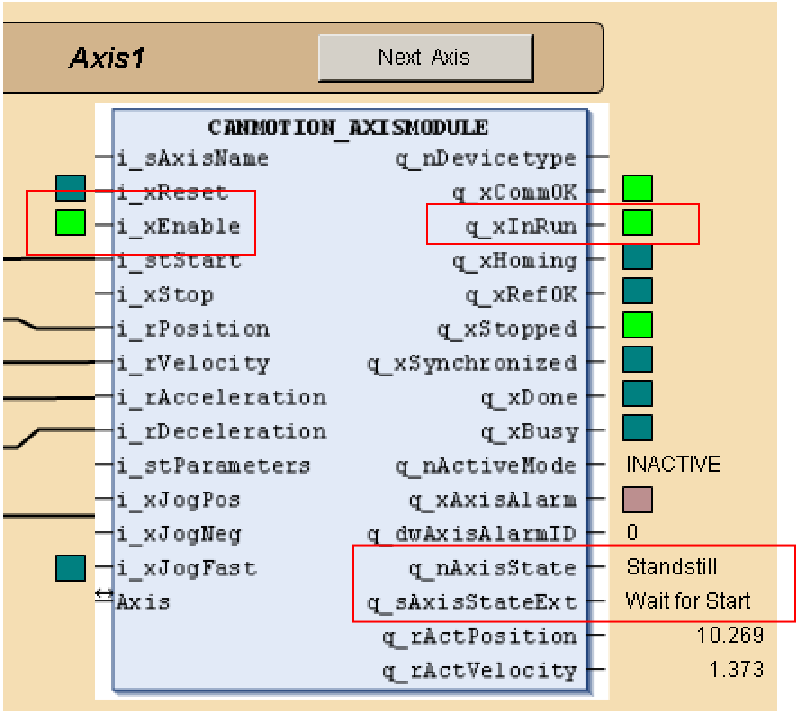
The 3 buttons on the right-hand side of the screen access the **CAM**ming and **GEAR**ing synchronization controls. The current screen is **INDEP**endent control.



---

3. Operate the Axes as a Synchronous Set

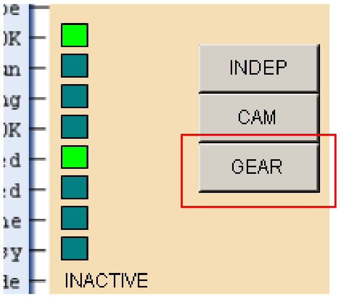
i. From the Independent control panel, **Enable** the Master, Axis1, and Axis2 drives by clicking on the **i_xEnable** input box as shown.



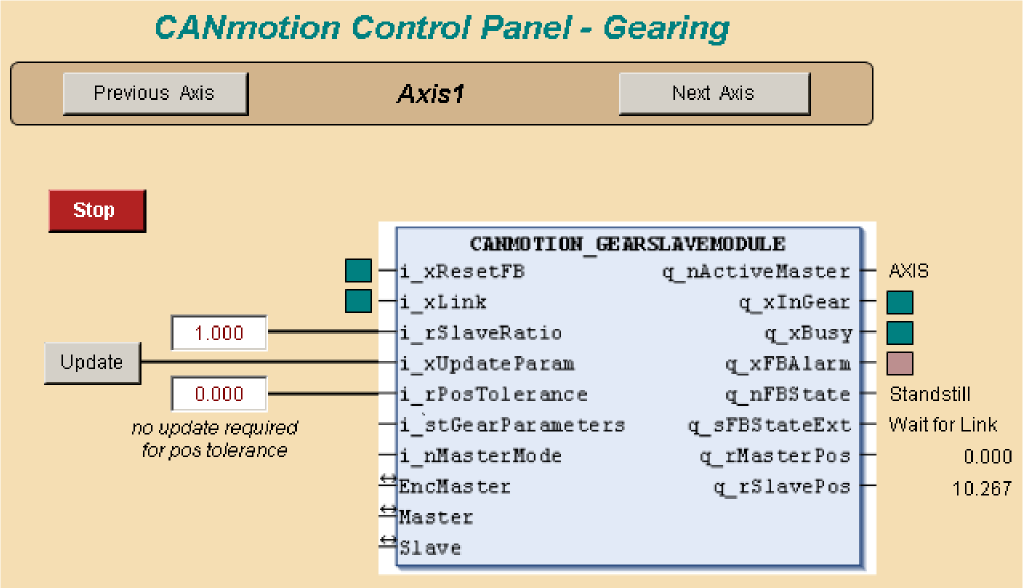
Note the change in the output status **q_xInRun**. In addition, the PLCopen Status (nAxisState) changes from **Disabled** to **Standstill**.

Also note the extended message indicating that the axis is “waiting for a start input”.

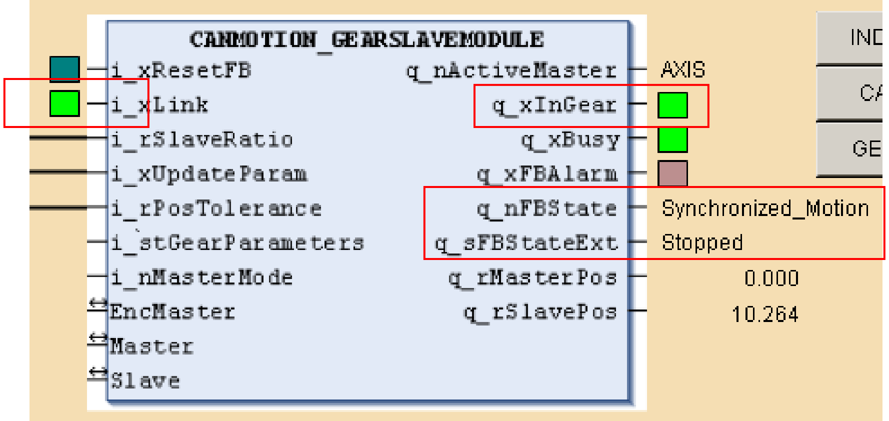
ii. Select the GEAR button to open Gearing control.



iii. Navigate to **Axis1** using the **Previous Axis / Next Axis** buttons.

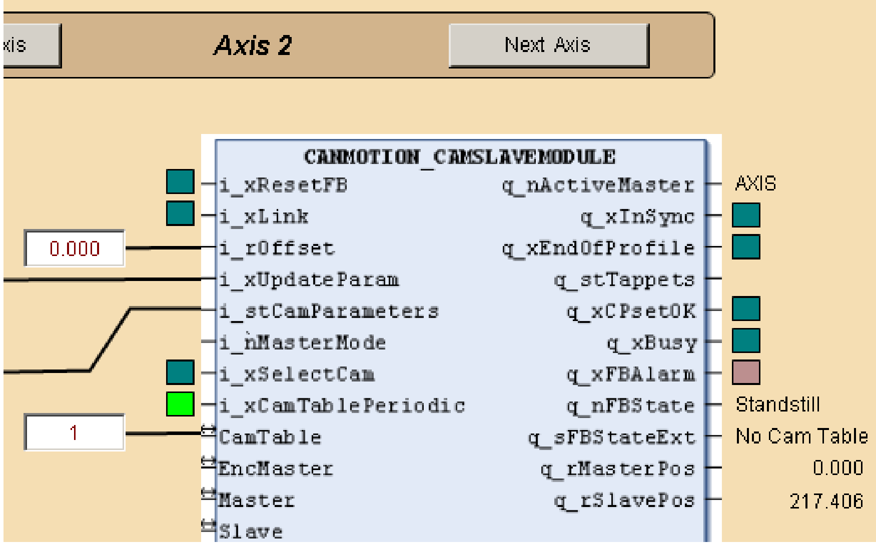


iv. Synchronize the axis to the Virtual Master by selecting the **i_xLink** input.

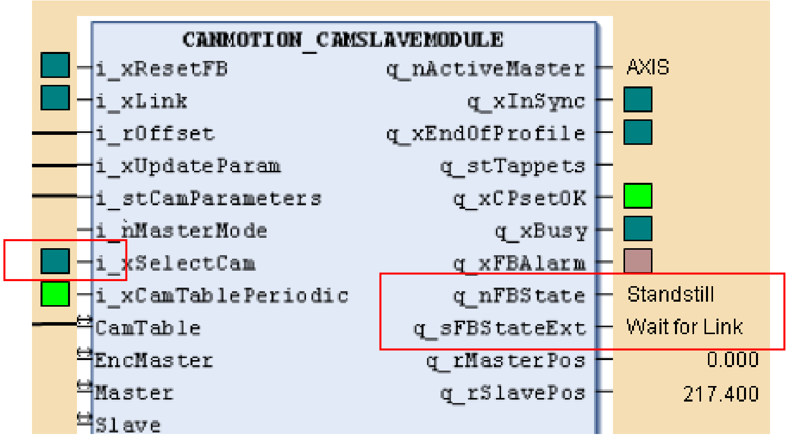


Note the response in the **q_xInGear** output, as well as the change in PLCopen state from **Standstill** to **Synchronized_motion**.

v. Navigate to Axis 2 and change the operating screen to CAM.

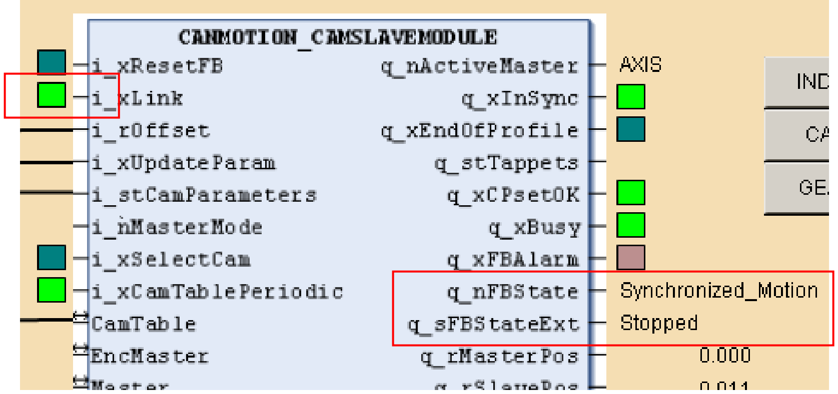


vi. Select CAM Profile “1” (entry field) by clicking on the **i_xSelectCam** input. The input is momentary.

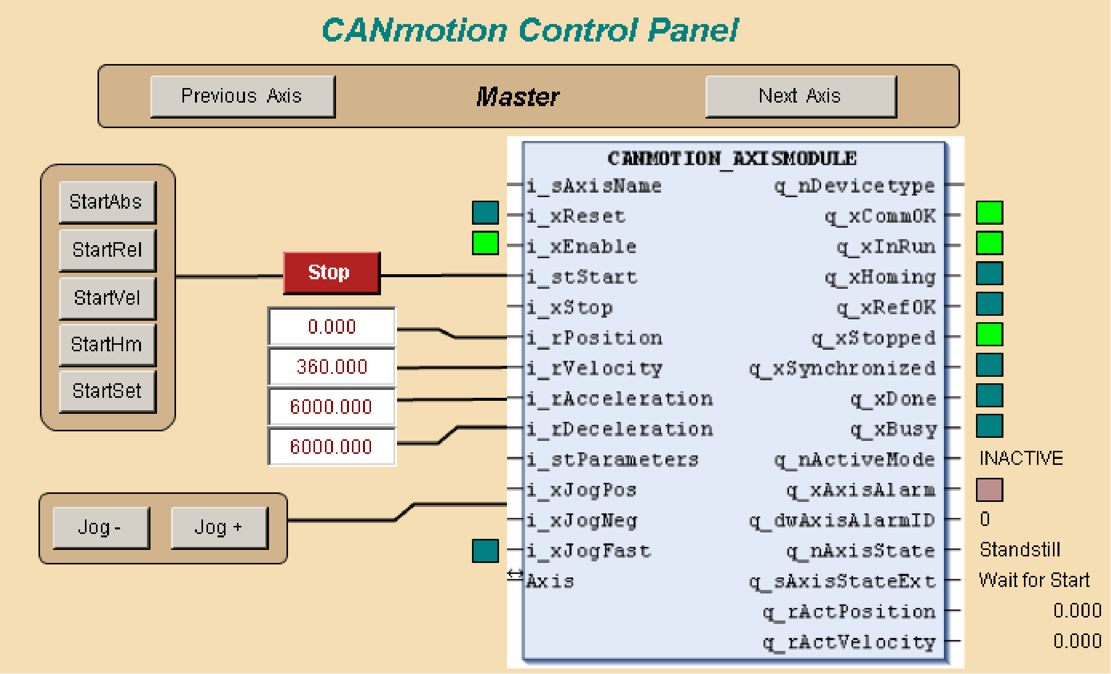


Note that the axis now “Waiting for Link”.

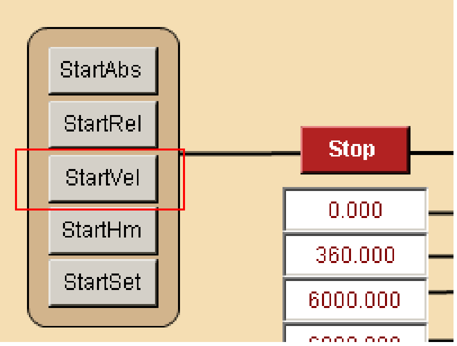
vii. Link the Axis to the Virtual Master as before, and make sure that the PLCopen state changes to **Synchronized_motion**.



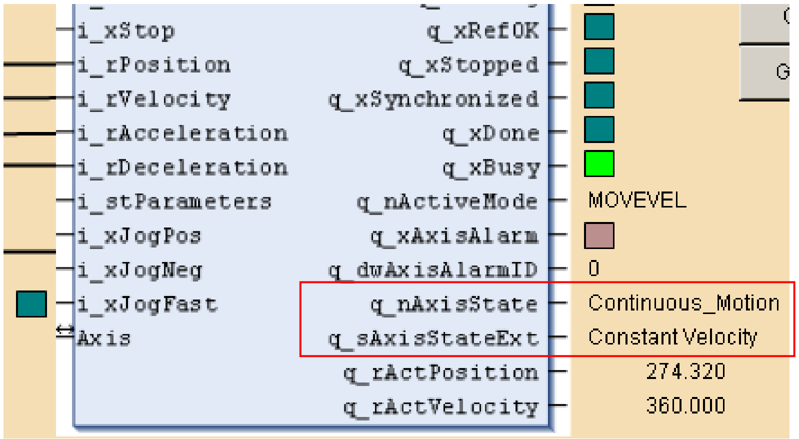
viii. Now select the **INDEP** screen button, and navigate to the **Master** axis as shown.



ix. Select **StartVel** to initiate a continuous velocity movement of the Master axis. The slave axes will respond according to the Gearing parameters and Cam profile



Note the transition of the PLCopen state to **Continuous_Motion**.



4. On your own

i. Experiment with a variety of independent and synchronous movement types using the Manual interface screens

ii. Be creative… you can’t hurt anything!

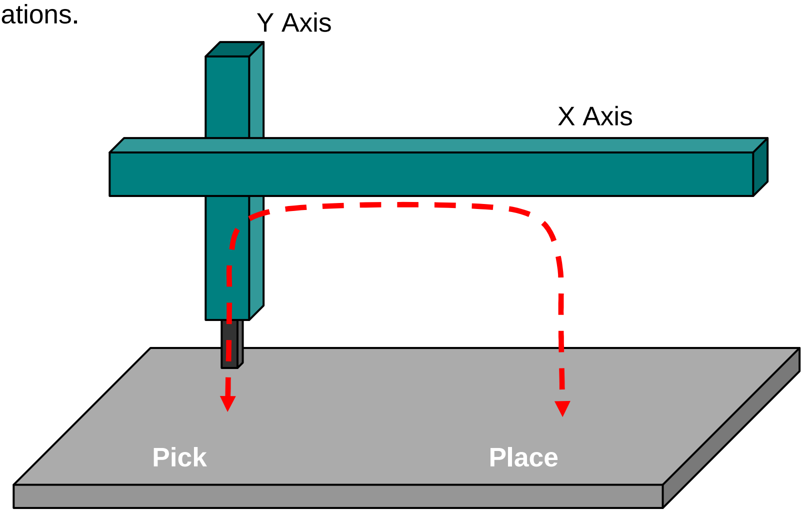
This completes the Exercise

Chapter 7: Applying SoftStruXure – Robot

In this chapter, we will apply the SoftStruXure template to a machine application. A pick-n-place robot will be used to illustrate the techniques of a state machine in governing a defined sequence of events. The application will also highlight the management of exceptions.

Machine Overview

A Cartesian robot is required to pick a product from one location, and place it in another. The Y axis moves the gripper vertically into position to perform a product pick or place. The X axis performs a horizontal transit between the pick and place locations.



Each axis is coupled to timing belt actuator with a 5:1 gearbox. The pitch diameter of the pulleys is 2.5 inches. The nominal transit speed of the robot is 10 inches/sec.

Basic Requirements

The machine requirements include:

- Automatic Axis Enable on Power up
- Prepare Mode for Homing the axes
- Manual Mode for Jogging the axes
- Automatic Mode for normal operation
- Start and Stop buttons for Automatic mode

The machine will incorporate 2 sensors that indicate an available product for pick and an available location for placing the product.

Inputs

Hardware Inputs include:

- Input 1: Momentary Start button
- Input 2: Momentary Stop button
- Input 3: Pick position OK
- Input 4: Place position OK

User entry into the SoftStruXure template is often limited to the following:

- Establish the correct number and type of axes
- Initialize machine and axis parameters
- Assign hardware inputs and outputs
- Create the Operational state machine
- Manage User Alarms and Exceptions

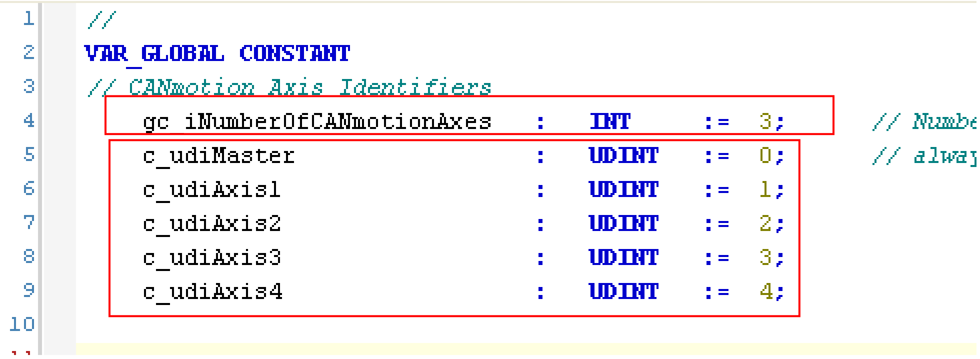
In the next sections, we will navigate the template browser to determine how to perform these steps as required for the Robot application.

Axis Configuration

For the robot example, 2 servo axes are required. These are already pre-mapped in the template, so the only requirement is to scale the axes according to the mechanical system.

The number of axes instantiated in the template is determined by a global constant located in the Global Variable List, **GVL_Constants**.

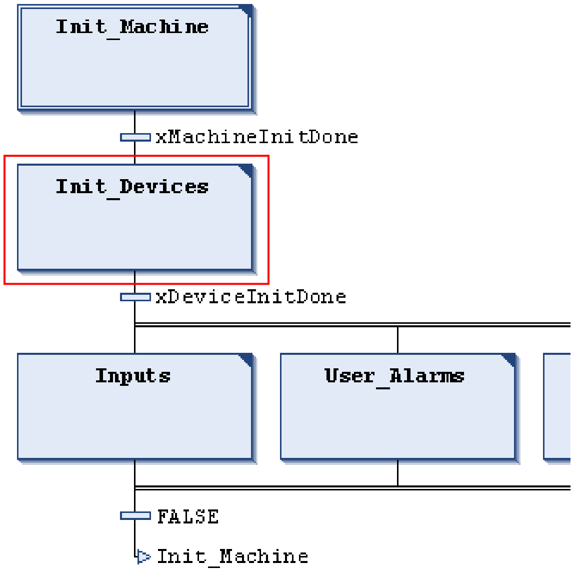
The Declaration sets the number of axis objects (including a virtual master), and an enumeration constant for each axis.



```
//
VAR_GLOBAL CONSTANT
// CANmotion Axis Identifiers
    gc_iNumberOfCANmotionAxes   :   INT     :=  3;      // Numbe
    c_udiMaster                 :   UDINT   :=  0;      // alway
    c_udiAxis1                  :   UDINT   :=  1;
    c_udiAxis2                  :   UDINT   :=  2;
    c_udiAxis3                  :   UDINT   :=  3;
    c_udiAxis4                  :   UDINT   :=  4;

```

The enumeration constant is used to identify the appropriate axis as an array element.

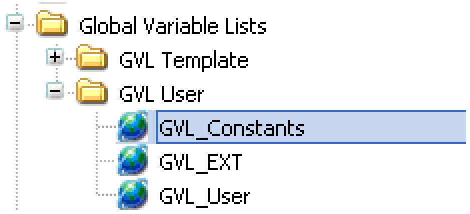
Initialization parameters for these axes are found in the **Init_Devices** step of SR_Main.



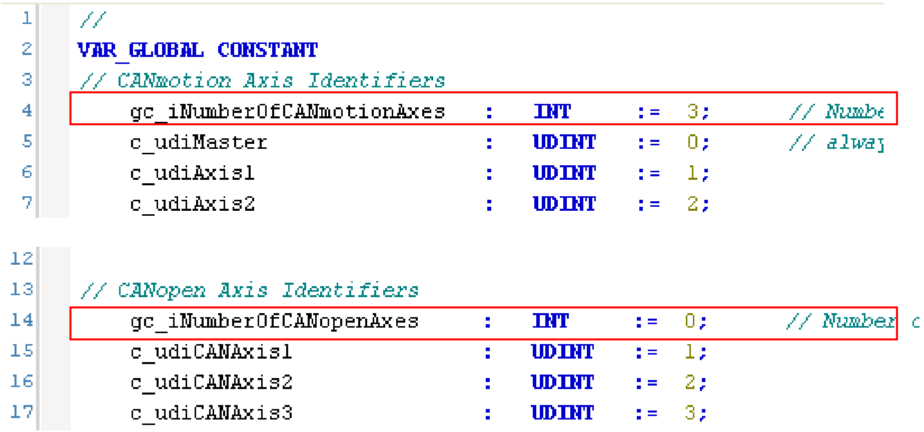
Exercise – Configure the Robot Axes

1. Set the number of axes

i. From the browser, open the **GVL_Constants** declaration list.


Global Variable Lists
- GVL Template
- GVL User
  - GVL_Constants
  - GVL_EXT
  - GVL_User

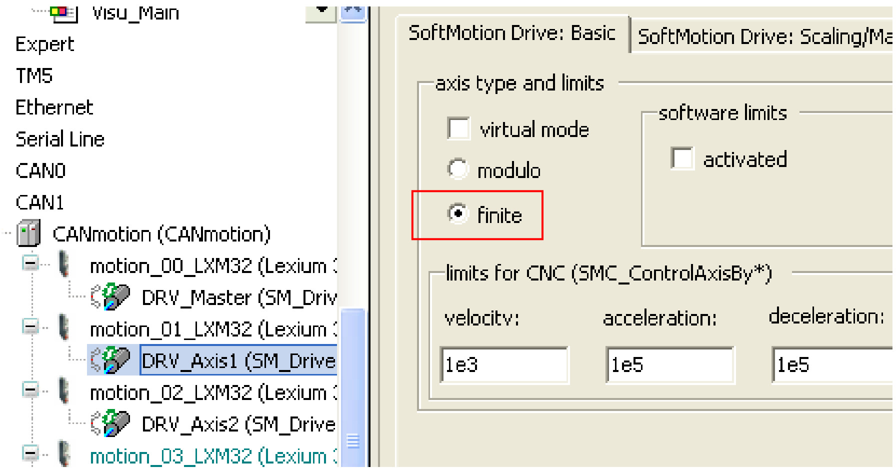
ii. Make sure that the number of CANmotion axes equals 3, and the number of CANopen axes equals 0.


```
//
VAR_GLOBAL CONSTANT
// CANmotion Axis Identifiers
    gc_iNumberOfCANmotionAxes     :  INT     :=  3;      // Numbe
    c_udiMaster                   :  UDINT   :=  0;      // alwa
    c_udiAxis1                    :  UDINT   :=  1;
    c_udiAxis2                    :  UDINT   :=  2;

// CANopen Axis Identifiers
    gc_iNumberOfCANopenAxes       :  INT     :=  0;      // Number c
    c_udiCANAxis1                 :  UDINT   :=  1;
    c_udiCANAxis2                 :  UDINT   :=  2;
    c_udiCANAxis3                 :  UDINT   :=  3;
```

---

2. Map the Axis hardware

i. From the CAN1 port, double-click the axis **DRV_Axis1** SoftMotion object. Select the Axis type "**finite**".



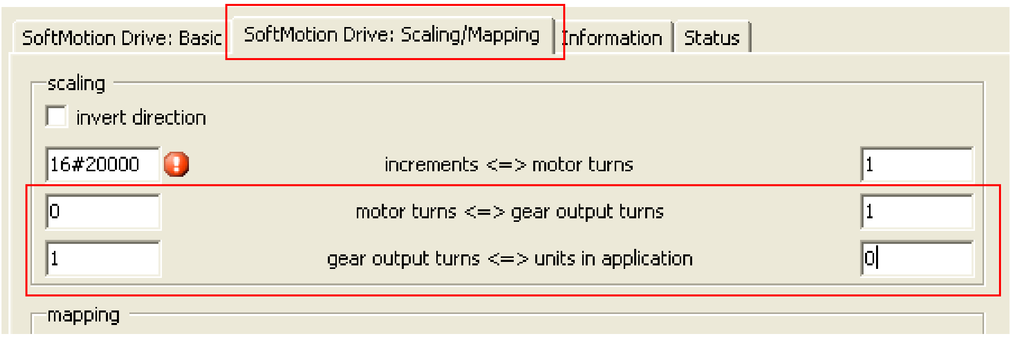
axis type and limits
- ☐ virtual mode
- ○ modulo
- ● finite

software limits
- ☐ activated

limits for CNC (SMC_ControlAxisBy*)
- velocity: 1e3
- acceleration: 1e5
- deceleration: 1e5

ii. Select the **Scaling/Mapping** tab, and set the axis scaling according to the machine specifications...

Hint: remember, each of the motors drives a 2.5 pitch diameter pulley through a 5:1 gearbox.



scaling

☐ invert direction

16#20000	increments <=> motor turns	1
0	motor turns <=> gear output turns	1
1	gear output turns <=> units in application	0

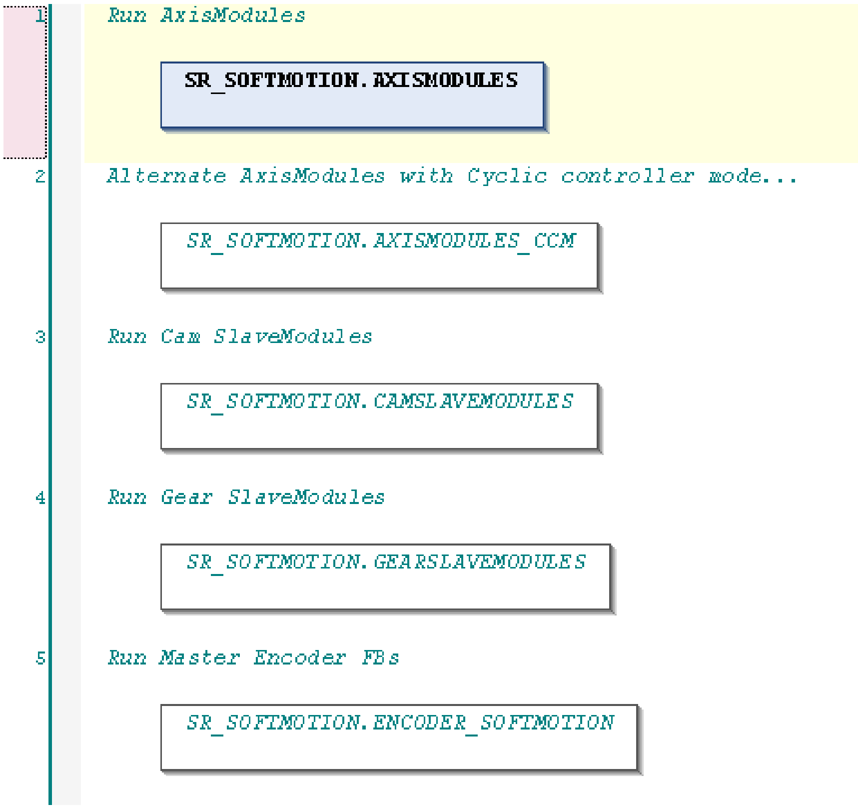
mapping

iii. Repeat steps i and ii for the second axis, **DRV_Axis2**.

---

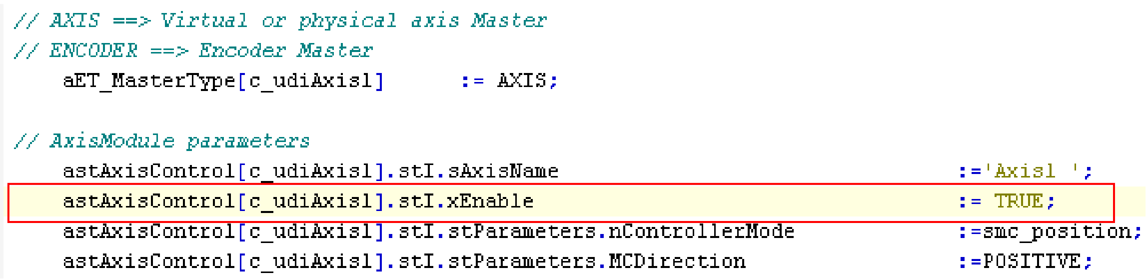
3. Map the Axis functionality

i. In **Task_Calls >> SR_SoftMotion**, Open the step **SoftMotion FBs** and comment all but the AxisModules call.



```
1  Run AxisModules
       SR_SOFTMOTION.AXISMODULES
2  Alternate AxisModules with Cyclic controller mode...
       SR_SOFTMOTION.AXISMODULES_CCM
3  Run Cam SlaveModules
       SR_SOFTMOTION.CAMSLAVEMODULES
4  Run Gear SlaveModules
       SR_SOFTMOTION.GEARSLAVEMODULES
5  Run Master Encoder FBs
       SR_SOFTMOTION.ENCODER_SOFTMOTION
```

ii. Browse to **SR_Main >> Init_Devices >> Init_Axis1**, and remove the Enable line comment to activate auto enable.



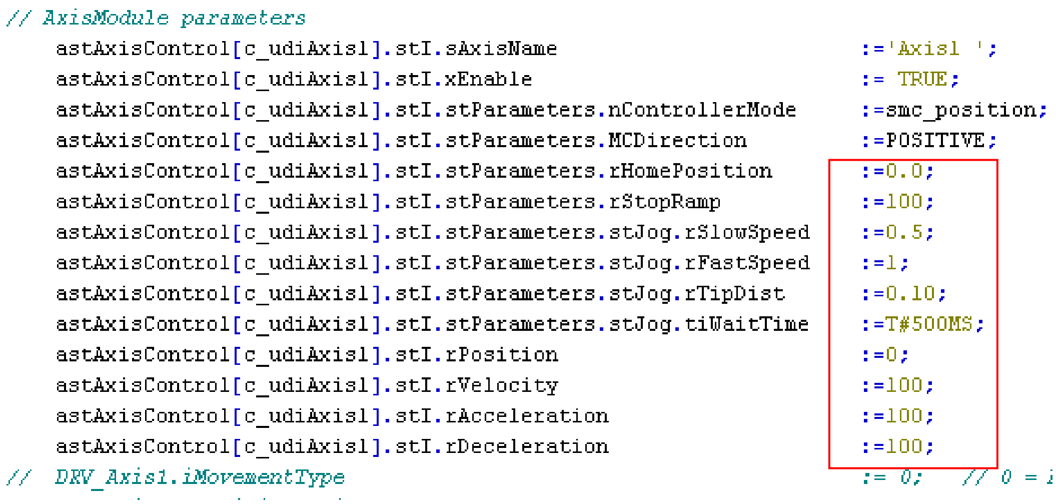
```
// AXIS ==> Virtual or physical axis Master
// ENCODER ==> Encoder Master
    aET_MasterType[c_udiAxis1]      := AXIS;

// AxisModule parameters
    astAxisControl[c_udiAxis1].stI.sAxisName                                :='Axis1 ';
    astAxisControl[c_udiAxis1].stI.xEnable                                  := TRUE;
    astAxisControl[c_udiAxis1].stI.stParameters.nControllerMode             :=smc_position;
    astAxisControl[c_udiAxis1].stI.stParameters.MCDirection                 :=POSITIVE;
```

Recall that the axis user units are set to:

- Position [inches]
- Velocity [in/sec]
- Acc/Dec[in/sec/sec]

Also, the nominal linear speed of the transit is 10 in/sec.

iii. Set the remaining Axis 1 movement parameters according to the user units.



```
// AxisModule parameters
    astAxisControl[c_udiAxis1].stI.sAxisName                            :='Axis1 ';
    astAxisControl[c_udiAxis1].stI.xEnable                              := TRUE;
    astAxisControl[c_udiAxis1].stI.stParameters.nControllerMode         :=smc_position;
    astAxisControl[c_udiAxis1].stI.stParameters.MCDirection             :=POSITIVE;
    astAxisControl[c_udiAxis1].stI.stParameters.rHomePosition           :=0.0;
    astAxisControl[c_udiAxis1].stI.stParameters.rStopRamp               :=100;
    astAxisControl[c_udiAxis1].stI.stParameters.stJog.rSlowSpeed        :=0.5;
    astAxisControl[c_udiAxis1].stI.stParameters.stJog.rFastSpeed        :=1;
    astAxisControl[c_udiAxis1].stI.stParameters.stJog.rTipDist          :=0.10;
    astAxisControl[c_udiAxis1].stI.stParameters.stJog.tiWaitTime        :=T#500MS;
    astAxisControl[c_udiAxis1].stI.rPosition                            :=0;
    astAxisControl[c_udiAxis1].stI.rVelocity                            :=100;
    astAxisControl[c_udiAxis1].stI.rAcceleration                        :=100;
    astAxisControl[c_udiAxis1].stI.rDeceleration                        :=100;
//  DRV_Axis1.iMovementType                                             := 0;   // 0 =
```

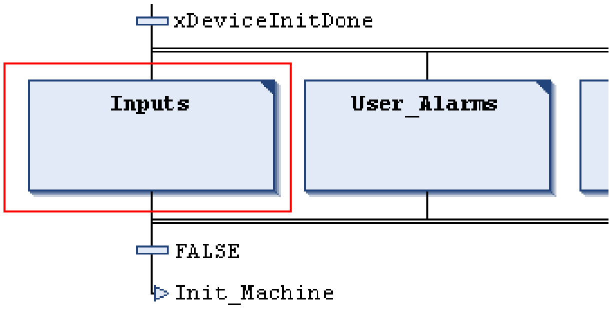
iv. Repeat the initialization for Axis 2.

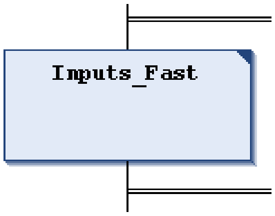
v. **Save** the project

This completes the Exercise

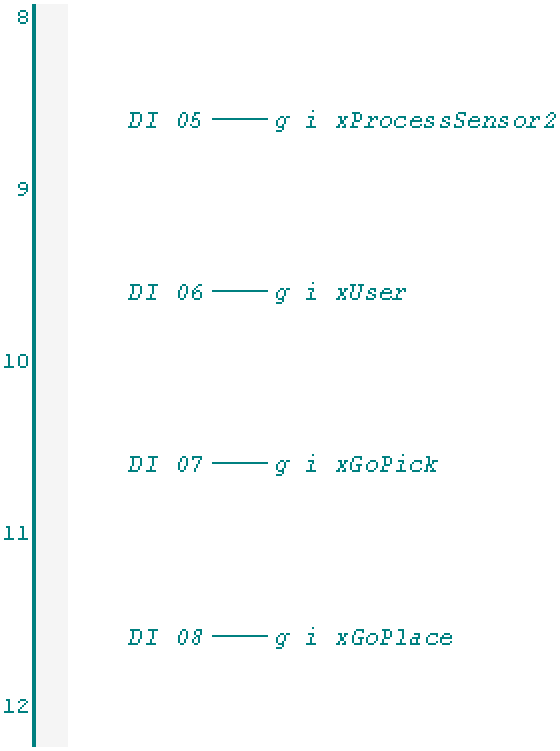
Hardware Inputs

TM5 standard and fast inputs are pre-mapped to symbolic names **DI_xx** and **FDI_xx**. Assignments to global user variables are made in the **Inputs** step and **Inputs_Fast** step of SR_Main and SR_SoftMotion respectively..





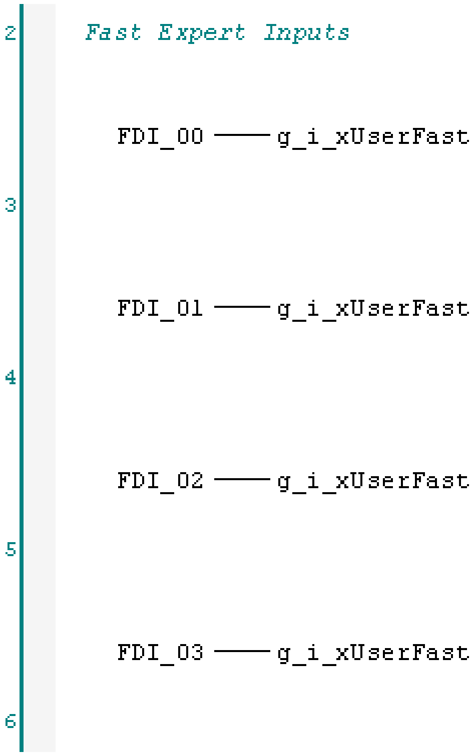
In **SR_Main >> Input_Hardware_Map,** TM5 hardware inputs are pre-mapped to generic global user input variables as shown.



```
DI 05 —— g i xProcessSensor2
DI 06 —— g i xUser
DI 07 —— g i xGoPick
DI 08 —— g i xGoPlace
```

It is up to the User to uncomment and edit these assignments, or create the global variables assignments as needed for the application.

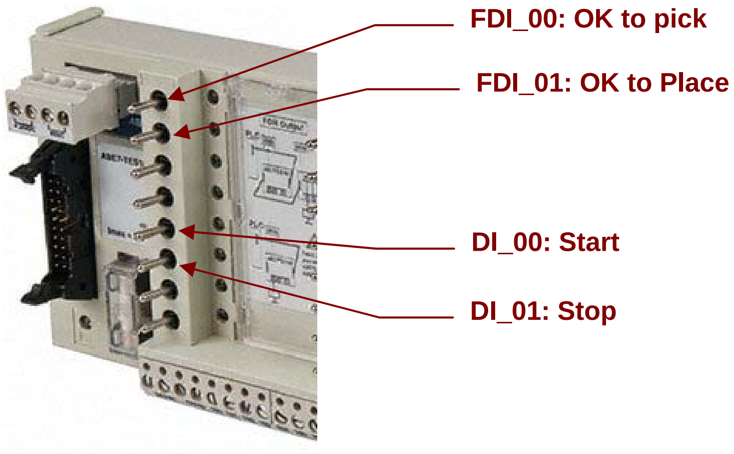
In **SR_SoftMotion >> Inputs_Fast**, a similar mapping is provided for the Fast Expert inputs.



```
Fast Expert Inputs

FDI_00 ——— g_i_xUserFast

FDI_01 ——— g_i_xUserFast

FDI_02 ——— g_i_xUserFast

FDI_03 ——— g_i_xUserFast
```

For this application, we will assign the following inputs:

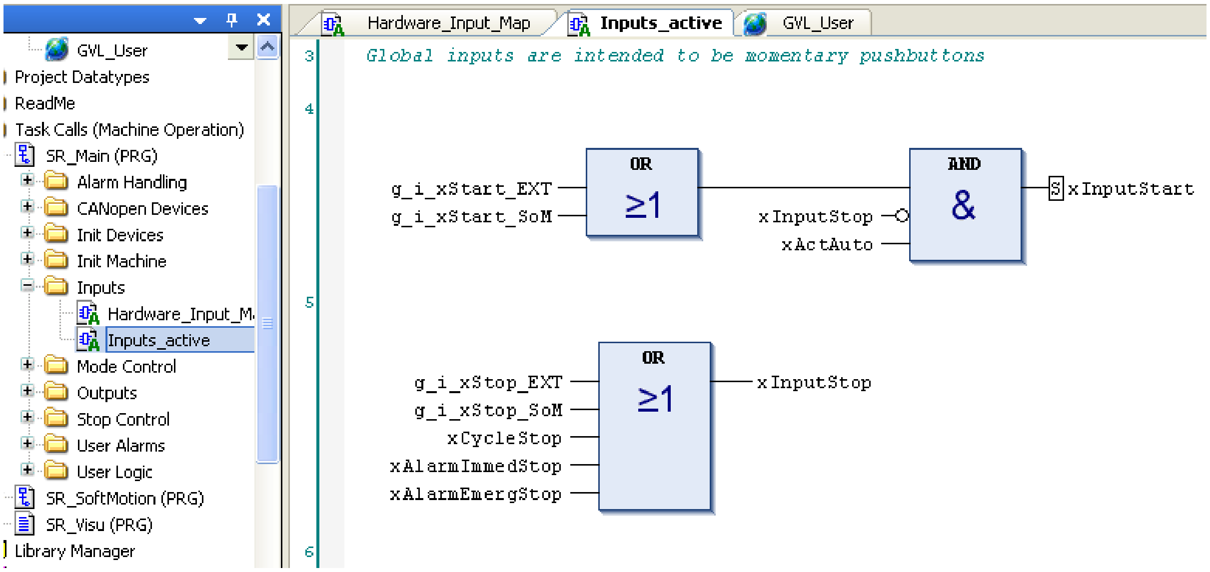
- DI_00: Momentary Start button
- DI_01: Momentary Stop button
- FDI_00: Pick position OK
- FDI_01: Place position OK



Exercise – Configure the Hardware Inputs

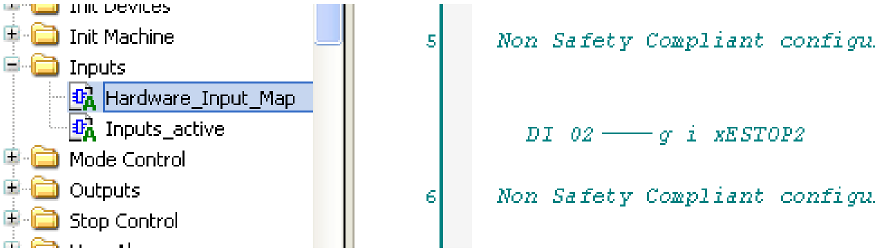
1. Assign the Start and Stop buttons

The template provides a pre-defined Start and Stop inputs identified within the input logic **SR_Main >> Inputs >> Inputs_active**. The variables **g_i_xStart_EXT** and **g_i_xStop_EXT** can be used as an entry point for Start and Stop commands from an HMI screen or hardwired push button.



Global inputs intended to be momentary pushbuttons

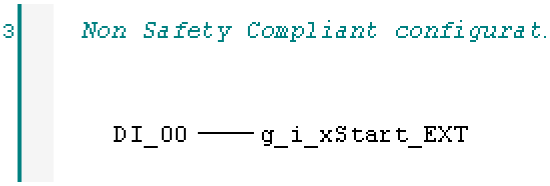
i. Select the **Hardware_Input_Map** logic to open the input assignment editor.



Non Safety Compliant configu

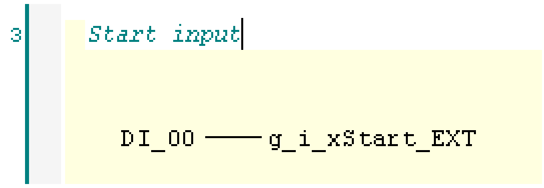
Non Safety Compliant configu

ii. Uncomment the rung, and assign the input **DI_00** to **g_i_xStart_EXT**.



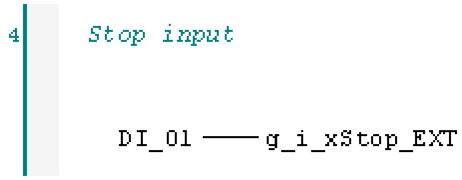
Non Safety Compliant configurat.

iii. Edit the comment field…



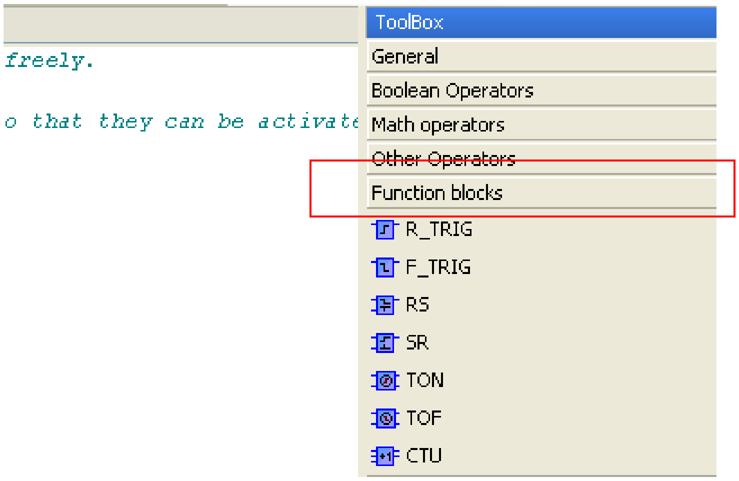
```
3   Start input

      DI_00 ——— g_i_xStart_EXT
```

iv. Repeat for the corresponding Stop input



```
4   Stop input

      DI_01 ——— g_i_xStop_EXT
```

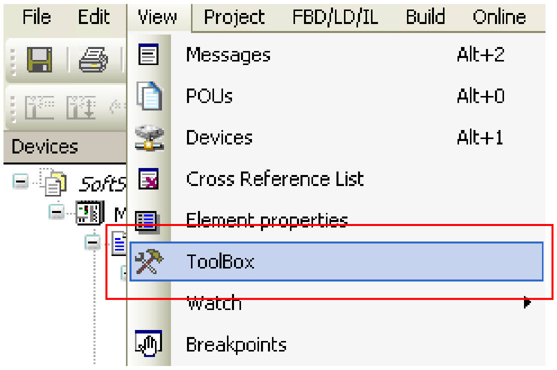
v. Make these input “momentary” (one-shot) by inserting a rising edge trigger. Open the **Toolbox** on the right-hand side of the editor screen, and select **Function blocks**.



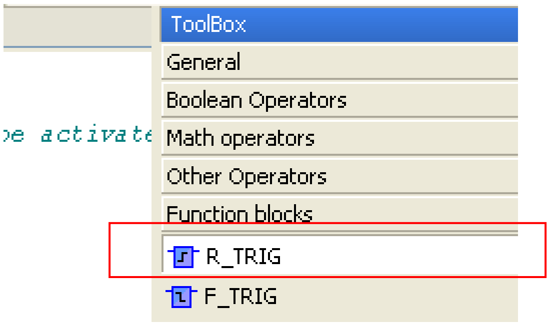
freely.

o that they can be activate

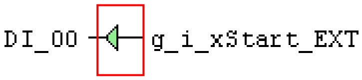
If the Toolbox is not visible, select it from the top level menu



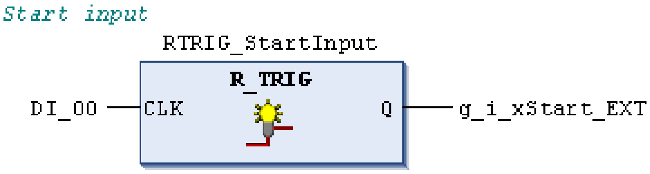
vi. Click on the **R_TRIG** function…



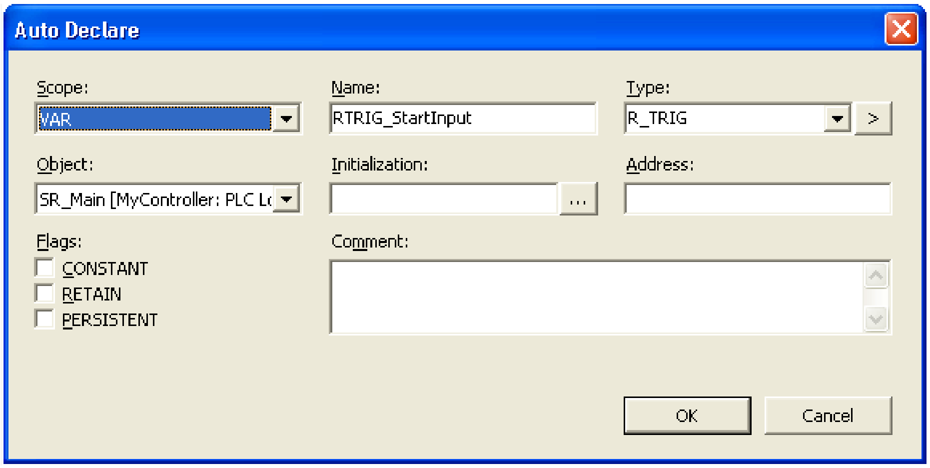
… and **drag** it to the insertion point as shown.



vii. Create the instance name "**RTRIG_StartInput**" as shown



viii. Click **OK** to accept the auto declaration.



Scope: VAR

Name: RTRIG_StartInput

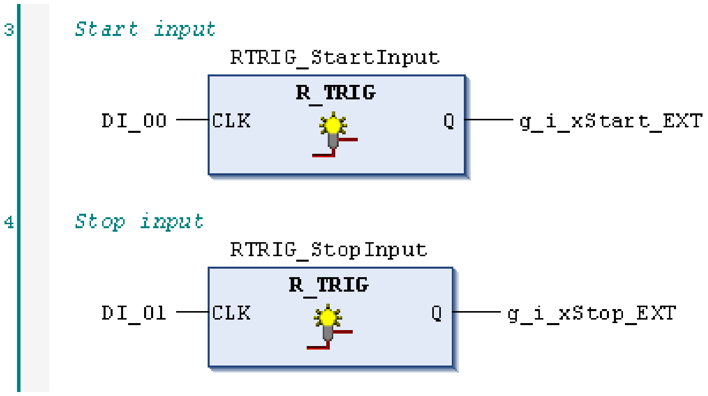
Type: R_TRIG

Object: SR_Main [MyController: PLC Lc

☐ CONSTANT
☐ RETAIN
☐ PERSISTENT

ix. Repeat this process for the Stop input.

The completed Start and Stop input assignments are shown below.

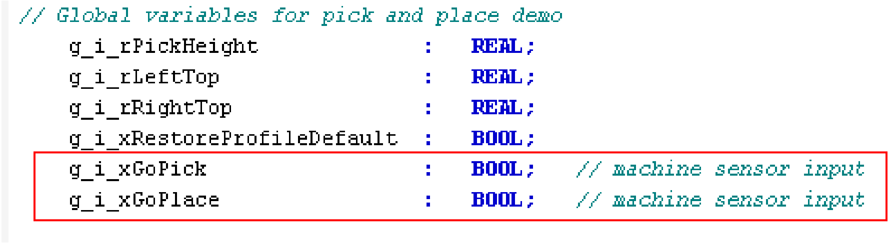


```
Start input
                 RTRIG_StartInput
             ┌──────────────────┐
             │      R_TRIG      │
DI_00 ───────┤CLK              Q├────── g_i_xStart_EXT
             └──────────────────┘

Stop input
                 RTRIG_StopInput
             ┌──────────────────┐
             │      R_TRIG      │
DI_01 ───────┤CLK              Q├────── g_i_xStop_EXT
             └──────────────────┘
```

---

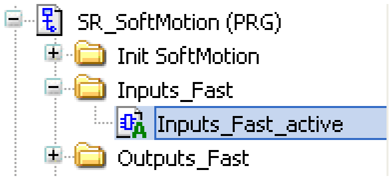
2. Assign the Pick and Place sensor inputs

The template includes predefined variables for the sensor inputs as shown. Alternatively, you could create your own.



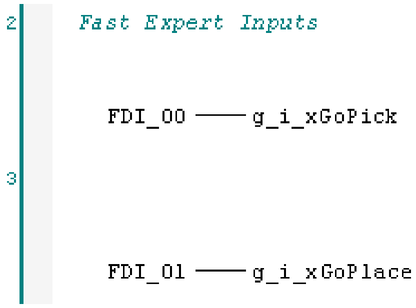
```
// Global variables for pick and place demo
    g_i_rPickHeight               :  REAL;
    g_i_rLeftTop                  :  REAL;
    g_i_rRightTop                 :  REAL;
    g_i_xRestoreProfileDefault    :  BOOL;
    g_i_xGoPick                   :  BOOL;   // machine sensor input
    g_i_xGoPlace                  :  BOOL;   // machine sensor input
```

i. Open the **Inputs_Fast_active** logic in **SR_SoftMotion**.



- SR_SoftMotion (PRG)
  - Init SoftMotion
  - Inputs_Fast
    - Inputs_Fast_active
  - Outputs_Fast

ii. Assign **g_i_xGoPick** to FDI_00, and **g_i_xGoPlace** to FDI_01.



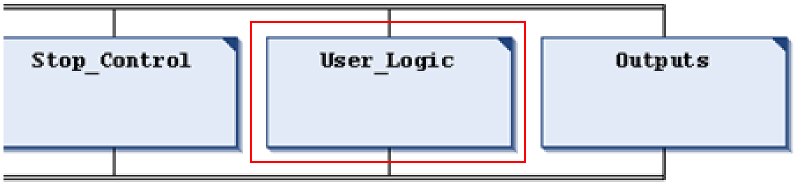
```
Fast Expert Inputs

FDI_00 ──── g_i_xGoPick

FDI_01 ──── g_i_xGoPlace
```

iii. Save the project.

This competes the Exercise

User_Logic

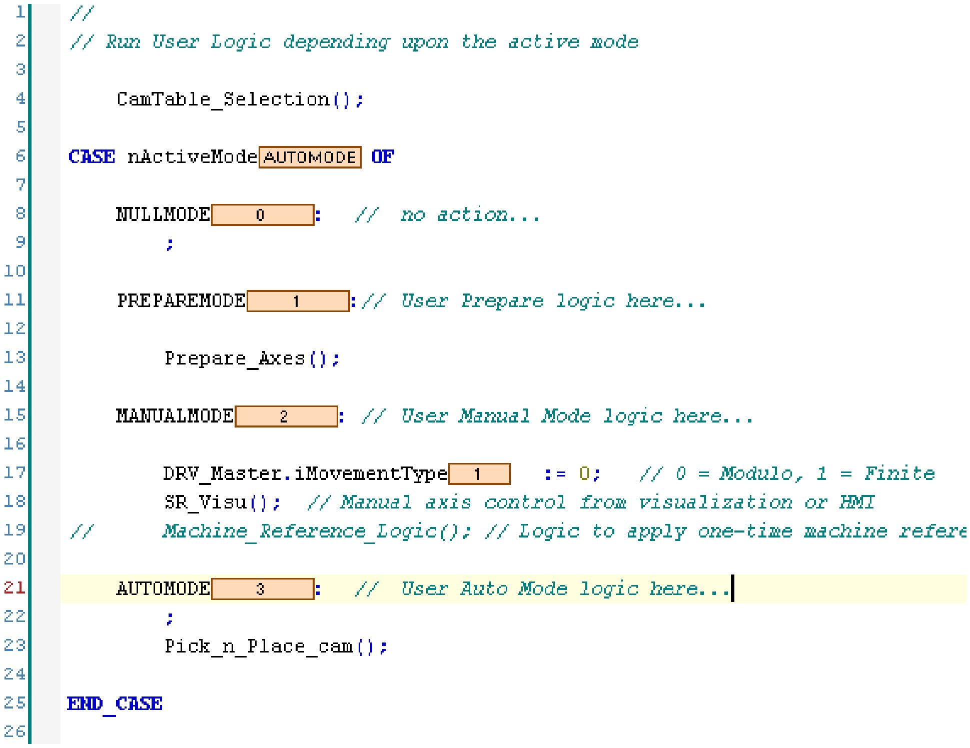
A **User_Logic** step is provided for User-specific (or Machine specific) code. The state machine for robot operation will be located here.



*A corresponding **User_Fast** step is provided within SR_SoftMotion for logic that must be solved on the fast Motionbus cycle time.*

Within **User_logic**, a decision-based CASE statement applies calls depending upon the current operating mode. The pre-defined operating modes are :

- NULLMODE
- PREPAREMODE
- MANUALMODE
- AUTOMMODE



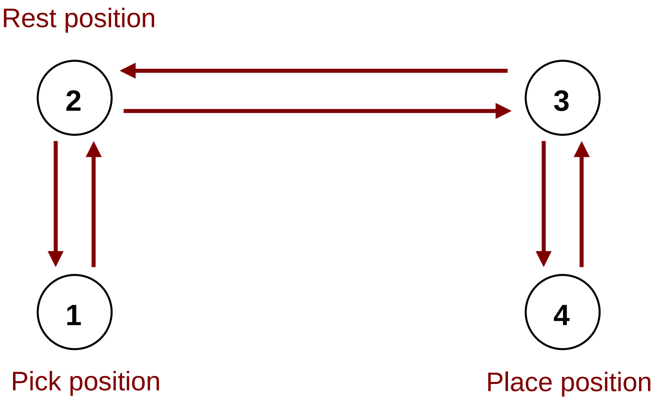
```
//
// Run User Logic depending upon the active mode

    CamTable_Selection();

CASE nActiveMode AUTOMODE OF

    NULLMODE 0 :    //  no action...
        ;

    PREPAREMODE 1 :// User Prepare logic here...

        Prepare_Axes();

    MANUALMODE 2 : // User Manual Mode logic here...

        DRV_Master.iMovementType 1   := 0;    // 0 = Modulo, 1 = Finite
        SR_Visu();  // Manual axis control from visualization or HMI
//      Machine_Reference_Logic(); // Logic to apply one-time machine refere

    AUTOMODE 3 :    //  User Auto Mode logic here...
        ;
        Pick_n_Place_cam();

END_CASE

```

The existing **Prepare_Axes** call is designed to enable and home all of the axes. The logic can be edited for a specific homing process or functionality as needed. For this application, it can be used as-is.

For the new robot application, the existing Pick-n-Place routine will be replaced by a new program created in the next exercise.

Robot Movement Path

The following path points will used to identify the robot target positions and exceptions.



Robot Operation

In order to build the state machine to control the robot, we have to know the operational sequence… provided by the machine builder. For our purposes, the operational sequence will be outlined as follows:

1. Machine is powered up
2. Axes are homed on command from Operator Screen (Prepare button)
3. Machine is placed into Auto Mode from the operator screen
4. Axes automatically go to the “rest” position 2.
5. Operator presses the start button
6. If “OK-to-pick”, then robot moves to position 1
7. Robot returns to position 2
8. Robot moves to position 3
9. If “OK_to_place” then robot moves to position 4
10. Robot returns to position 3
11. Robot moves to rest position 2
12. If “OK-to-pick”, robot continues to position 1
13. Repeat until Stop.
14. If a Stop command is given, the robot completes the current path back to the rest position 2.
15. Return to state 5.

Exception handling is indicated in red text above. The state machine to drive the operations wil include each of these process, as well as confirmation steps using output status from the axis modules.

In the next section, we will explore the basic concept and syntax of a state machine, and create a sequence-based state machine to drive these actions and exceptions.

What is a State Machine?

In the previous chapter, we controlled and monitored an axis using the **stAxisControl** variable structure. We *could* operate the robot manually by manipulating the interface variables and observing the result. We load parameters for speed, position, acceleration and deceleration for each axis. Then at each step of the operation, we make a conscious decision to execute an absolute movement of the appropriate axis in the correct sequence to move the gripper.

Problems...?

- We’re not very fast or efficient
- We’re not very reliable
- We’re too expensive

In this chapter, we will *automate* the functionality that we performed manually. The sequencing logic used to automate these functions is a “State Machine”, and all communication to the Axes will take place using the **stAxisControl** Interface.

CASE statement

Similar to **IF THEN ELSE**, a **CASE** statement is a form of ***multi-branch*** logic that provides an unlimited number of branch options.

The syntax is shown below.

diDayOfWeek is the test variable, and ***Sunday…Saturday*** are the test conditions. Each of the test condition is followed by a logic statement(s) to be performed if the test variable matches the particular expression.

```
CASE diDayOfWeek OF
        Sunday:
                RegularMenu();
                CloseShop();
        Monday:
                ClosedMonday() ;
        Tuesday:
                OpenShop() ;
                TuesdaySpecials() ;
        Wednesday:
                RegularMenu();
        Thursday:
                RegularMenu();
        Friday:
                WeekendSpecial();
        Saturday
                WeekendSpecials();
END_CASE
```

BRANCH CHOICES

Alternatively, these branch options can be driven chronologically to form an operational *sequence of events*.

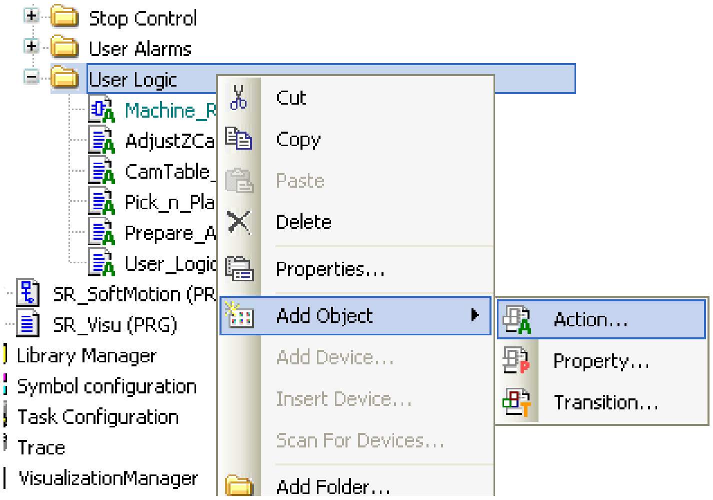
BRANCH SEQUENCE

```
CASE diSetupState OF
        0: // Wait
        10: // Load part
                IF xPartSensor THEN
                        diSetupState:= 20;
                END_IF
        20: // Jog to position
                JogControl() ;
                IF xPartInPlace THEN
                        dSetupState     := 30;
                END_IF
        30: // Reference:
                xStartHome      := TRUE;;
                diSetupState    := 40 ;
        40: // Confirm Reference completed
                If xHomingDone THEN
                        xStartHome      := FALSE;
                        diSetupState    := 50;
                END_IF;
        50: // Setup completed - Wait for Start button
                ;
END  CASE
```

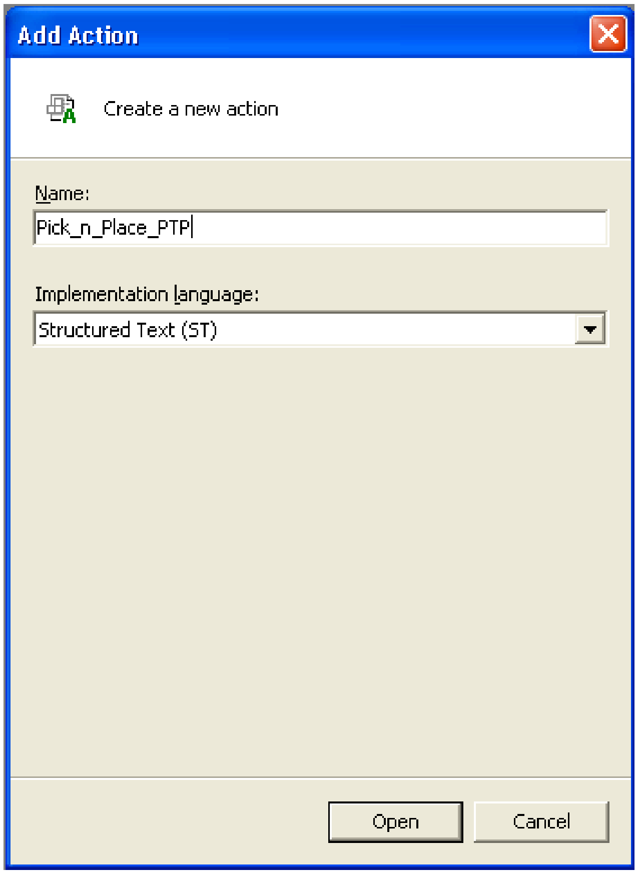
Exercise – Build the Robot State Machine

1. Add an “action” Object to the User Logic folder

i. From the project browser, right-click on the **User Logic** folder under **SR_Main**, Select **Add Object >> Action.**



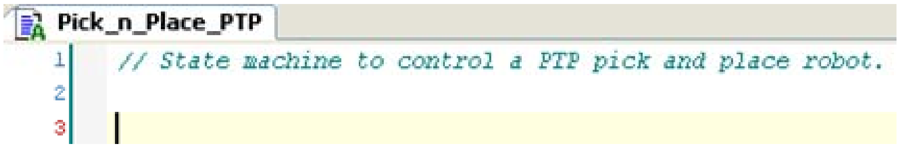
ii. Create the action “**Pick_n_place_PTP**” using the **Structured Text (ST)** editor as shown.



Name: Pick_n_Place_PTP

Implementation language: Structured Text (ST)

iii. Add a comment line at the top of the ST editor.



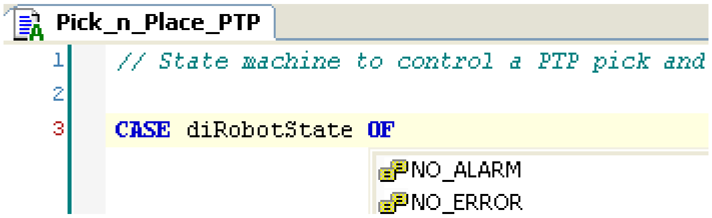
```
// State machine to control a PTP pick and place robot.

```

Comments can be created using the double slash //, or by surrounding the text by the comment delimiters (… *).*

---

2. Configure a CASE logic statement

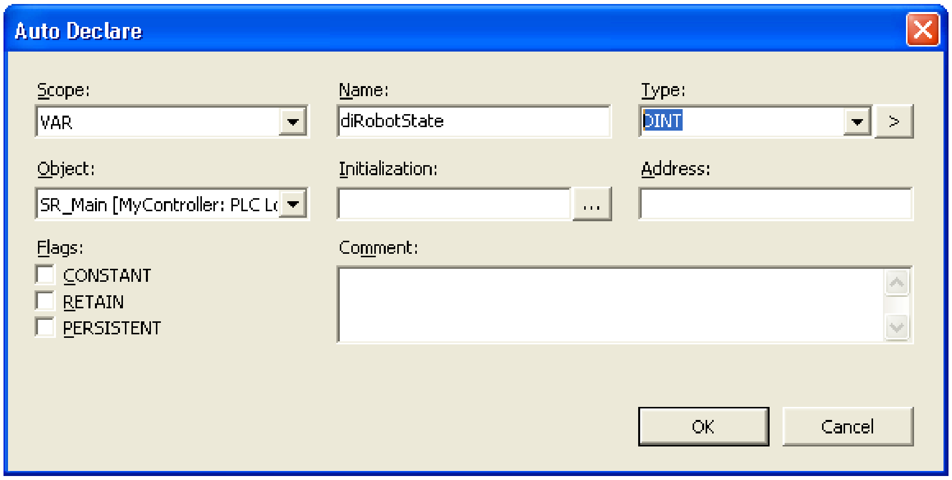
i. Create the CASE statement below the comment lines by typing the CASE syntax as shown.



```
// State machine to control a PTP pick and

CASE diRobotState OF
```

ii. Type **<Enter>** to complete the instance.

iii. Select (or type) **DINT** as the Auto Declaration data type for the CASE variable.



Scope: VAR

Name: diRobotState

Type: DINT

Object: SR_Main [MyController: PLC Lo

Initialization:

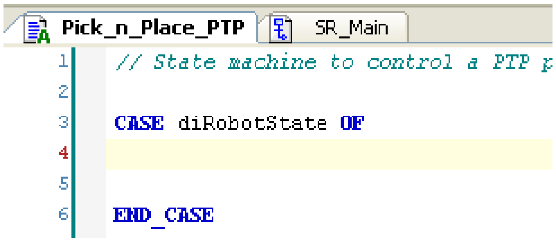
Address:

☐ CONSTANT

☐ RETAIN

☐ PERSISTENT

The completed CASE statement instance should look like the following.

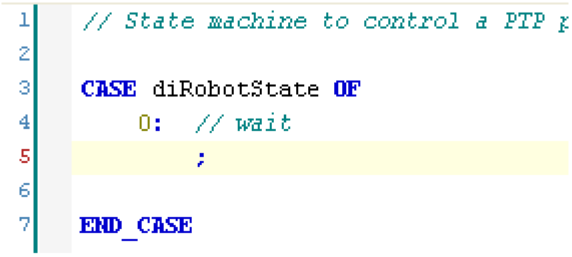


```
// State machine to control a PTP p

CASE diRobotState OF


END_CASE
```

3. Fill in the operating sequence for the robot.

In general it is useful to apply the sequence branches in multiples of 10. This makes it easier to insert lines later if needed. State “0” will be a “wait” state.

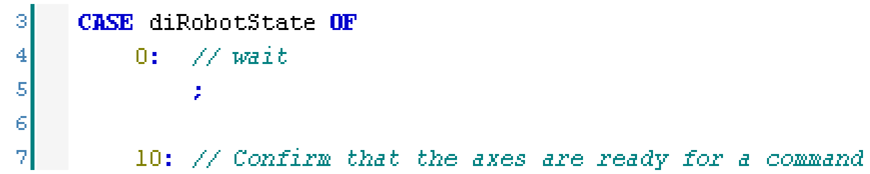
i. Create state 0 with a “wait” comment as shown. The semicolon is used as a “null” instruction.



```
// State machine to control a PTP p

CASE diRobotState OF
    0:   // wait
        ;

END_CASE
```

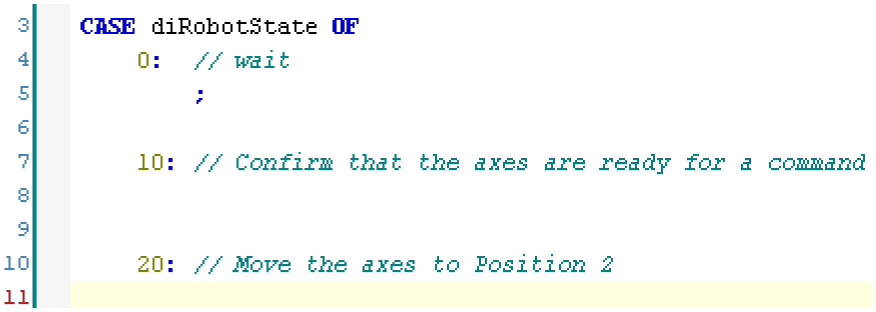
Use the *PLCopen* axis status confirmation, as well as the previous “operational sequence”, as guides to complete the state sequence objectives.

ii. Create the starting point (State 10) as a confirmation of the “ready” status of each of the axes.



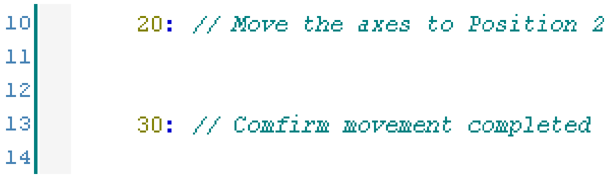
```
CASE diRobotState OF
    0:   // wait
        ;

    10: // Confirm that the axes are ready for a command
```

iii. Create state 20, to move the axes to the starting point “overpick” position 2.



```
CASE diRobotState OF
    0:   // wait
        ;

    10: // Confirm that the axes are ready for a command


    20: // Move the axes to Position 2

```

iv. Confirm that the movement has completed in state 30.



```
    20: // Move the axes to Position 2


    30: // Confirm movement completed

```

v. Edit the remaining sequence states as indicated:

40: // Wait for Start Input

50: // IF OK to pick, Move to Pick Position 1

60: // Confirm move completed

70: // Move back to Position 2

80: // Confirm move completed

90: // Move to Overplace Position 3

100: // Confirm move completed

110: // IF OK to Place, Move to Place Position 4

120: // Confirm move completed

130: // Move back to Position 3

140: // Confirm move completed

150: // Return to Rest Position 2

160: // Confirm move completed

170: // Return to check Start status, and repeat cycle

---

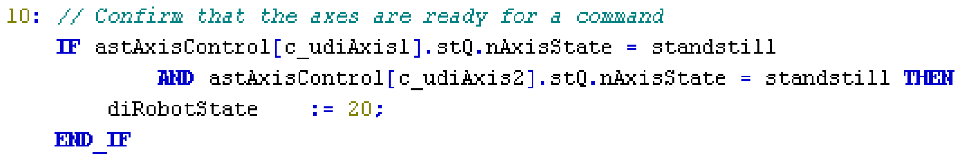
4. Assign the state machine sequence logic.

Control of the axes is managed through the components of the interface structure array variable, **astAxisControl [ ]**.

For example, an axis is ready to accept a movement command if it is PLCopen state “Standstill”.

astAxisControl[c_udi_Axis1].stQ.nAxisState = Standstill.

i. In step 10, apply logic to test that each of the axes are ready to accept a command. If so, jump to the next sequence step.

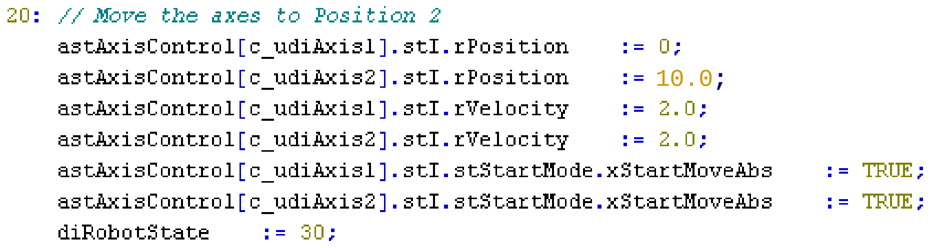


```
10: // Confirm that the axes are ready for a command
    IF astAxisControl[c_udiAxis1].stQ.nAxisState = standstill
            AND astAxisControl[c_udiAxis2].stQ.nAxisState = standstill THEN
        diRobotState    := 20;
    END_IF
```

The target positions are defined as follows:

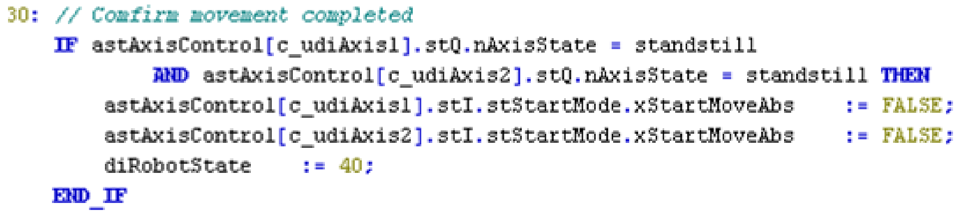
Position	Axis1 (X)	Axis2 (Y)
1	0.0	0.0
2	0.0	10.0
3	24.0	10.0
4	24.0	0.0

ii. In step 20, apply the initial movement parameters, and trigger an absolute move.



```
20: // Move the axes to Position 2
    astAxisControl[c_udiAxis1].stI.rPosition     := 0;
    astAxisControl[c_udiAxis2].stI.rPosition     := 10.0;
    astAxisControl[c_udiAxis1].stI.rVelocity     := 2.0;
    astAxisControl[c_udiAxis2].stI.rVelocity     := 2.0;
    astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbs    := TRUE;
    astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbs    := TRUE;
    diRobotState    := 30;
```

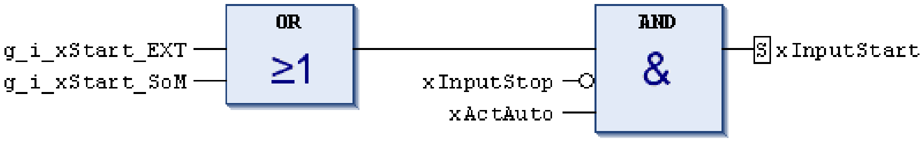
iii. In step 30, confirm that the movement has completed, and reset the **xStartMoveAbs** commands.



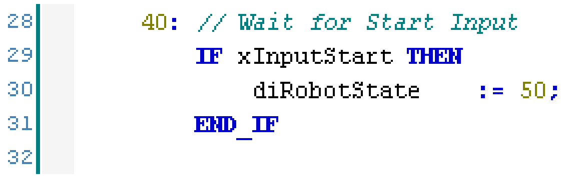
```
30: // Confirm movement completed
    IF astAxisControl[c_udiAxis1].stQ.nAxisState = standstill
            AND astAxisControl[c_udiAxis2].stQ.nAxisState = standstill THEN
        astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbs    := FALSE;
        astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbs    := FALSE;
        diRobotState    := 40;
    END_IF
```

Hint: Copy and paste to simplify !!!

For the Start input, we will make use of the predefined variable **xInputStart**, which is SET to TRUE within the **Inputs_active** logic.



iv. Wait for the **xInputStart** condition in Step 40.



```
    40: // Wait for Start Input
        IF xInputStart THEN
            diRobotState    := 50;
        END_IF

```

v. Complete the remaining state logic steps.

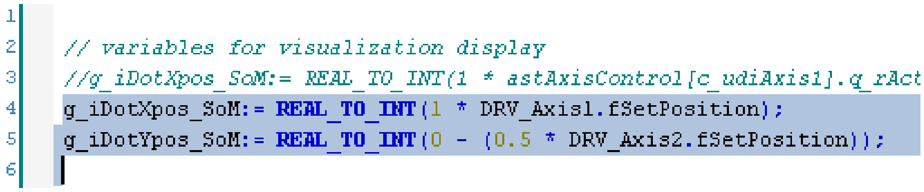
vi. Save and Rebuild the application to check for any syntax errors.

---

5. Create “position” values to be used in the visualization.

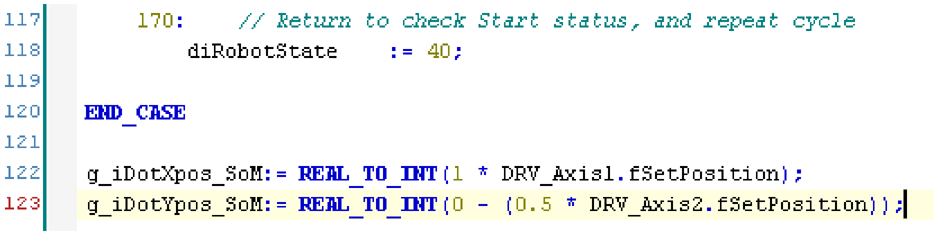
The Auto mode visualization screen displays the axis positions as a small blue box in the XY plane. The range of movement for the visualization screen is about 360 by 360 pixels. In this part of the exercise, we will copy the existing visualization position component from the old pick-n-place logic and modify it for use with the new robot.

i. Open the old **pick_n_place_cam** logic, and copy the highlighted “dot” position text as shown.



```

// variables for visualization display
//g_iDotXpos_SoM:= REAL_TO_INT(1 * astAxisControl[c_udiAxis1].q_rAct
g_iDotXpos_SoM:= REAL_TO_INT(1 * DRV_Axis1.fSetPosition);
g_iDotYpos_SoM:= REAL_TO_INT(0 - (0.5 * DRV_Axis2.fSetPosition));

```

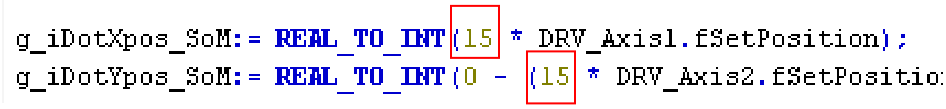
ii. Paste this into the new state machine logic below the **END_CASE** statement.



```
    170:    // Return to check Start status, and repeat cycle
        diRobotState    := 40;

END_CASE

g_iDotXpos_SoM:= REAL_TO_INT(1 * DRV_Axis1.fSetPosition);
g_iDotYpos_SoM:= REAL_TO_INT(0 - (0.5 * DRV_Axis2.fSetPosition));
```

Since the robot moves through a position range of about 24 inches. We need to multiply the visualization scaling by approximately 360 / 24, or 15 in order to be able to see comparable movement.

iii. Edit the logic to “amplify” the on-screen movement of the axes as shown.

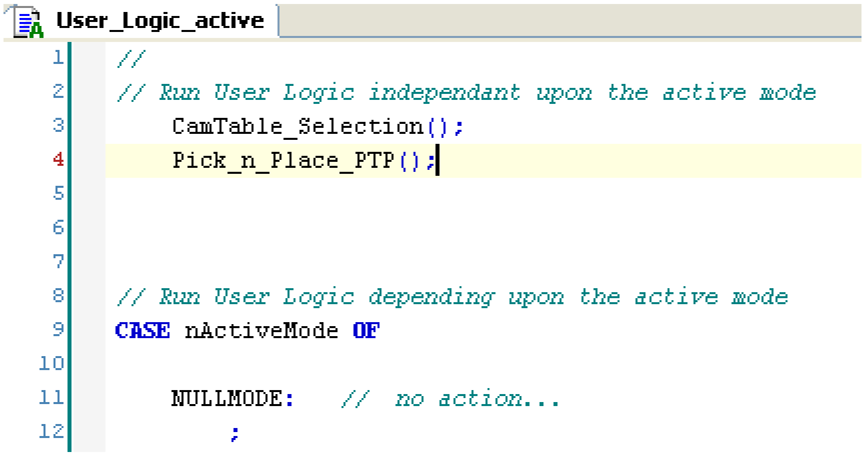


```
g_iDotXpos_SoM:= REAL_TO_INT(15 * DRV_Axis1.fSetPosition);
g_iDotYpos_SoM:= REAL_TO_INT(0 - (15 * DRV_Axis2.fSetPositio
```

---

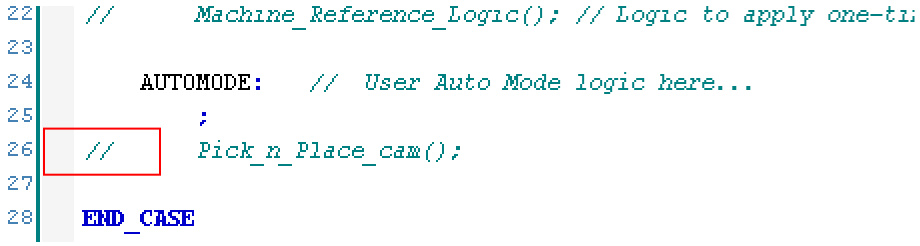
6. Assign the state machine call to the User Logic action.

i. From the browser, select **User_logic_action**, and add the new state machine logic outside of the CASE statement.



```
//
// Run User Logic independant upon the active mode
    CamTable_Selection();
    Pick_n_Place_PTP();



// Run User Logic depending upon the active mode
CASE nActiveMode OF

    NULLMODE:   //  no action...
        ;
```

ii. “Comment” the existing state machine call within the AUTOMODE branch to prevent the old logic from solving.



```
//      Machine_Reference_Logic(); // Logic to apply one-ti

    AUTOMODE:   //  User Auto Mode logic here...
        ;
//      Pick_n_Place_cam();

END_CASE
```

---

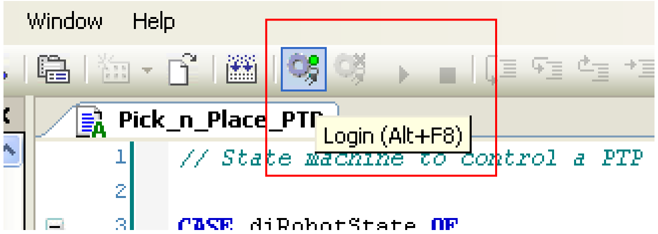
7. Save the project

i. **Build** the project to check syntax.

ii. **Save** the project.

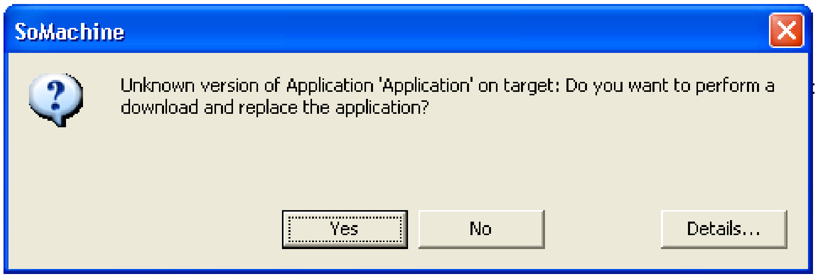
8. Download to the controller

i. Using an Ethernet connection, Select **Login** from the top level menu.



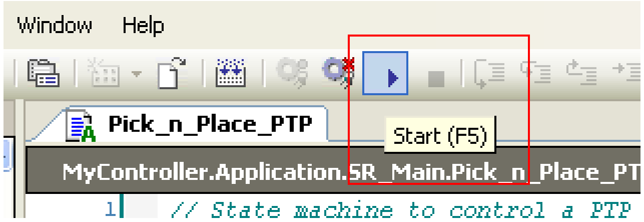
```
// State machine to control a PTP

CASE diRobotState OF
```

ii. Acknowledge any download or connection prompt.



Unknown version of Application 'Application' on target: Do you want to perform a download and replace the application?

iii. When the download has completed, select RUN from the top level menu.



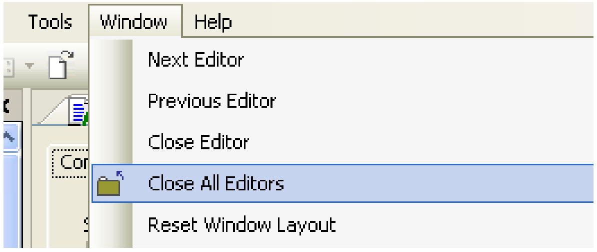
```
// State machine to control a PTP
```

This completes the exercise

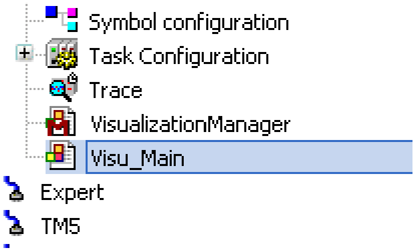
Exercise – Operate the Machine

1. Run the Project from VISU_Main

i. From the top level menu, select **Window >> Close All Editors**.



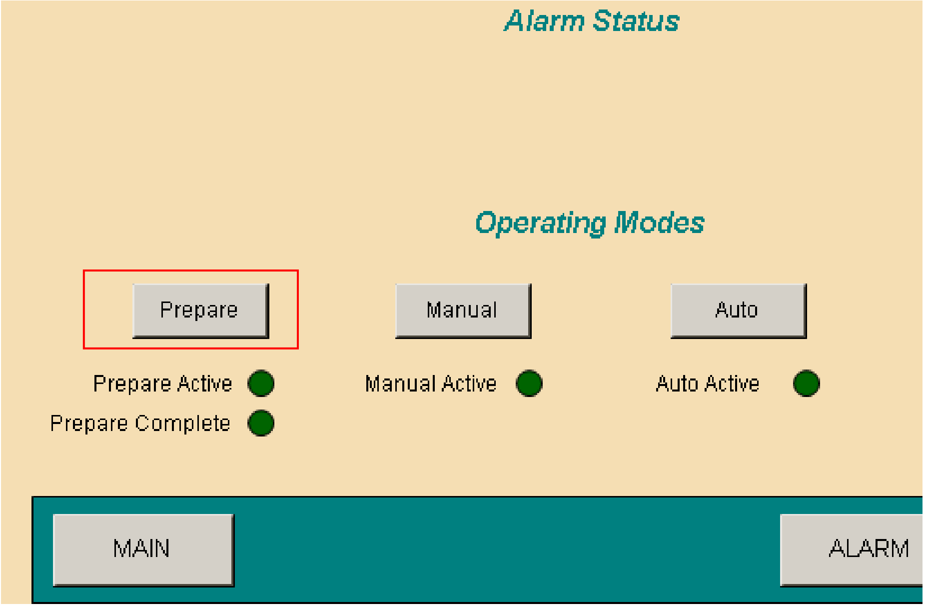
ii. From the Project browser, double-click to open the main visualization screen **Visu_Main**.



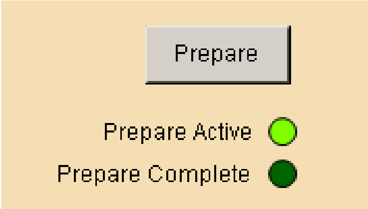
---

2. Prepare the machine

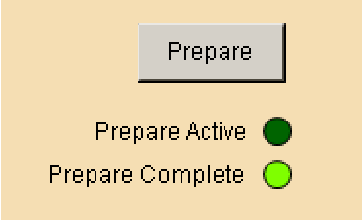
i. From the Main Screen, Press the **Prepare** button to initiate the axis *Homing* process.



ii. Observe the **Prepare Active** light turn ON. The axes will begin homing, in order, to the encoder Index mark.



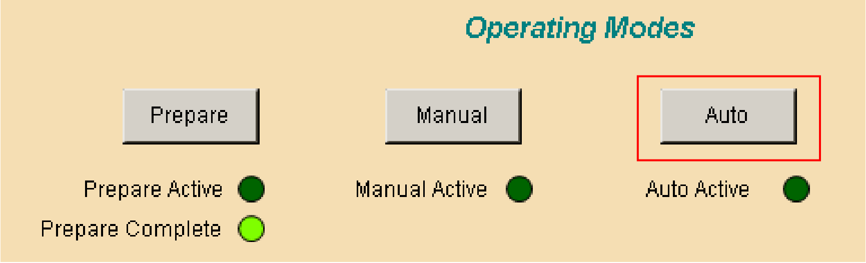
At the completion of the homing process, the **Prepare Active** light will turn OFF, and the **Prepare Complete** light will turn ON.



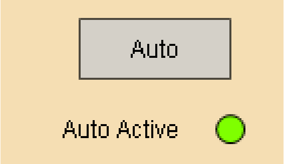
---

3. Select Auto Mode

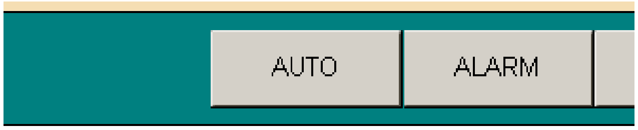
i. Press the AUTO button on the main screen to select the Automatic operating mode.



ii. If all permissives are met, and the machine transitions to Auto Mode, the Auto Active light will turn ON and the Auto screen button will appear at the bottom menu bar. Confirm the Auto Active light…



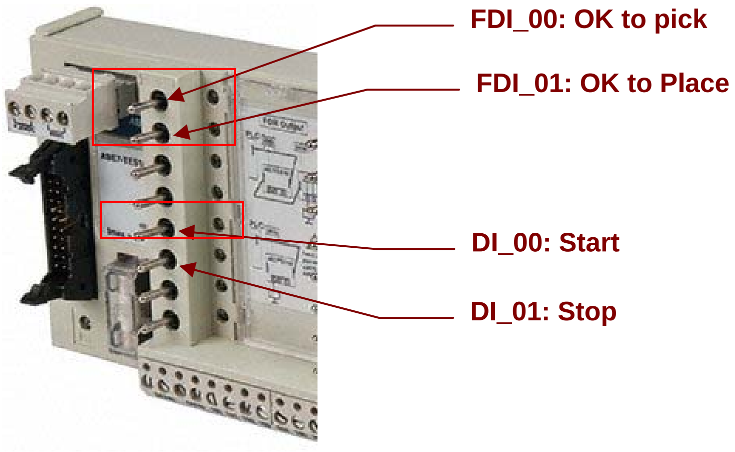
iii. … and select the **AUTO** operating screen from the bottom menu bar.



4. Start the machine and apply the exceptions

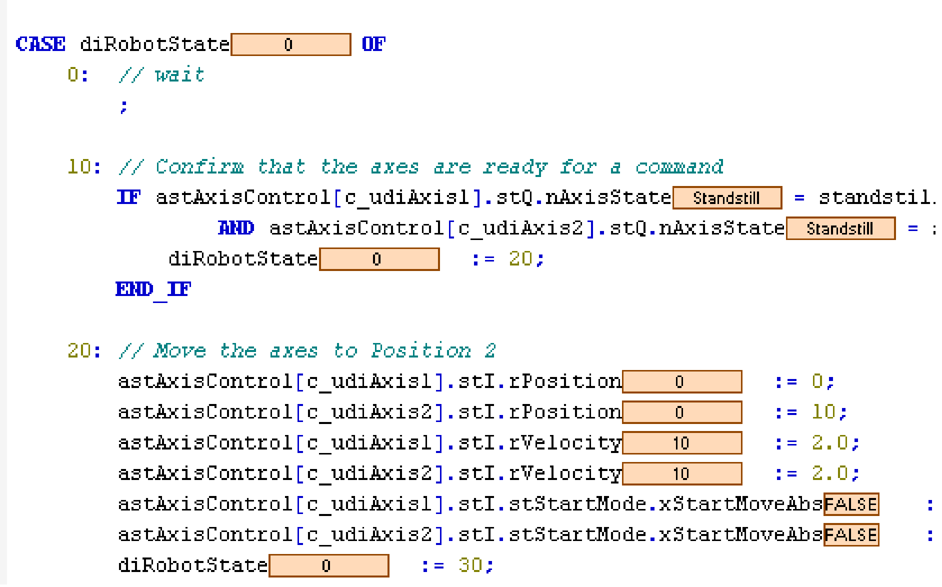
This will open the AUTO mode operating screen. Here you can Start and Stop the machine, and observe the physical movement of the robot axes.

i. Make sure that the **OK to Pick** and **OK to Place** inputs are ON as shown. Select the **Start** button, or Toggle the hardware **Start input** to start the machine.



Normally this would initiate the pick and place robot movement. However, in this case… there is no motion.

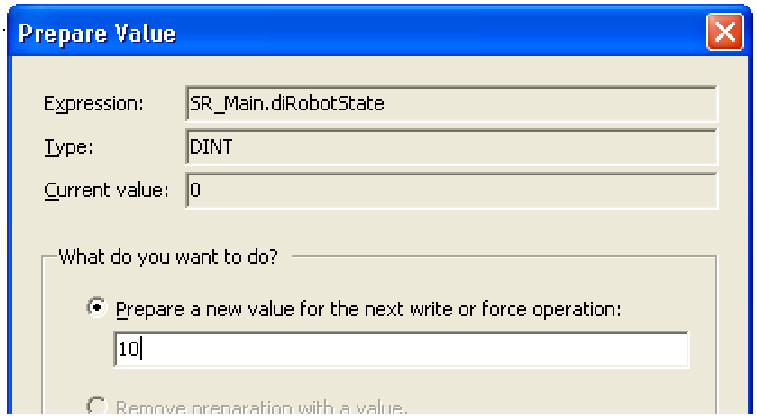
ii. Open the state machine to see which step is currently active,



```
CASE diRobotState  0  OF
    0:  // wait
        ;

    10: // Confirm that the axes are ready for a command
        IF astAxisControl[c_udiAxis1].stQ.nAxisState Standstill  = standstil.
                AND astAxisControl[c_udiAxis2].stQ.nAxisState Standstill  = :
            diRobotState  0    := 20;
        END_IF

    20: // Move the axes to Position 2
        astAxisControl[c_udiAxis1].stI.rPosition  0    := 0;
        astAxisControl[c_udiAxis2].stI.rPosition  0    := 10;
        astAxisControl[c_udiAxis1].stI.rVelocity  10   := 2.0;
        astAxisControl[c_udiAxis2].stI.rVelocity  10   := 2.0;
        astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbsFALSE    :
        astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbsFALSE    :
        diRobotState  0    := 30;
```

The state machine is “stuck” at diRobotState = 0. There is no logic mechanism to move the state variable to 10. We will do this manually at first, then add logic to automate the transition.

iii. Click on any of the available **diRobotState** fields, and enter the value 10 at the **Prepare Value** prompt.



Expression: SR_Main.diRobotState

Type: DINT

Current value: 0

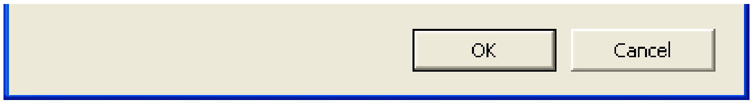
What do you want to do?

● Prepare a new value for the next write or force operation:

10

○ Remove preparation with a value.

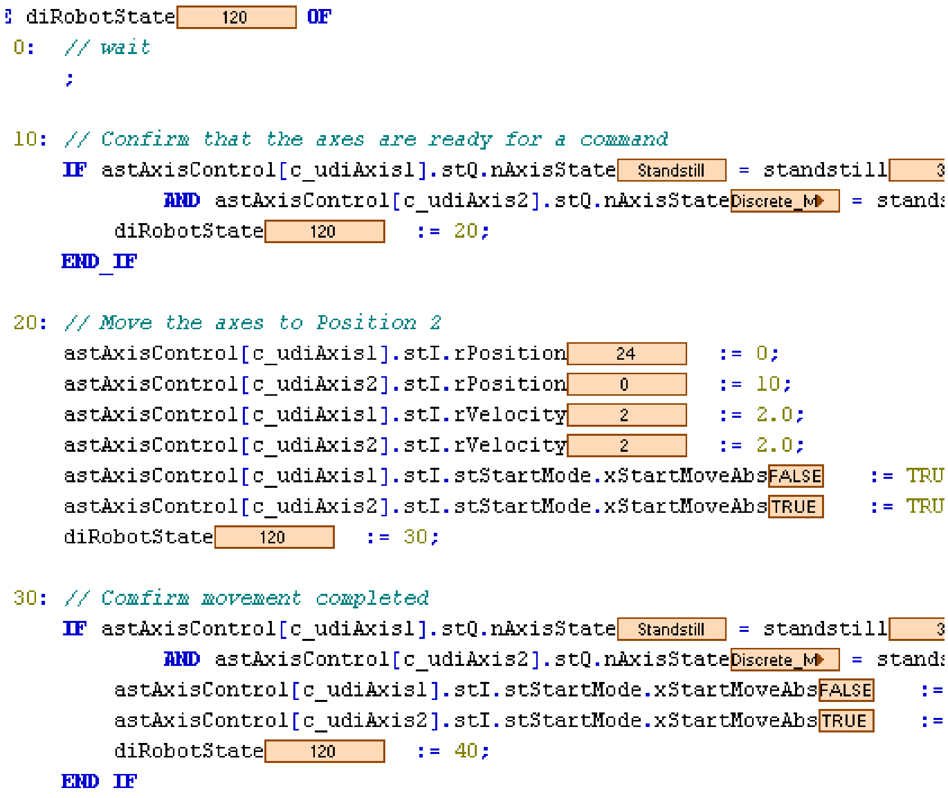
iv. Click **OK** to prepare the new value…



v. … then type **<CTRL><F7>** to apply the prepared value.

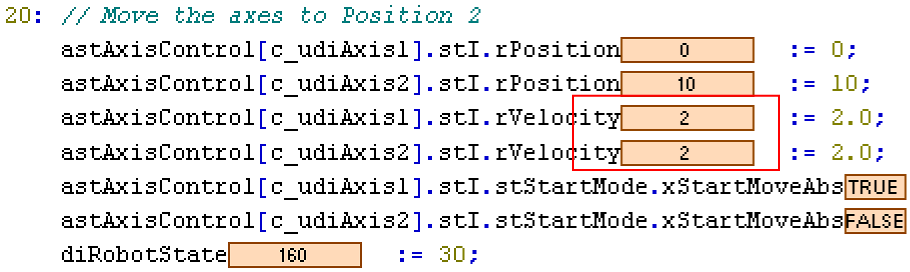
```
diRobotState 0 <10>
```

vi. The robot should begin running. Observe the online changes in the state machine.



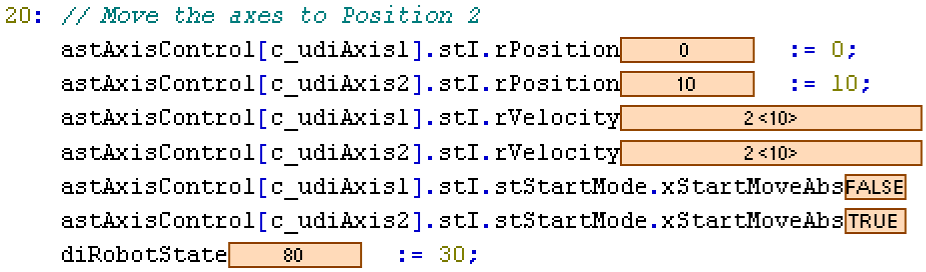
```
 diRobotState   120   OF
0:   // wait
     ;

10: // Confirm that the axes are ready for a command
    IF astAxisControl[c_udiAxis1].stQ.nAxisState  Standstill  = standstill   3
           AND astAxisControl[c_udiAxis2].stQ.nAxisStateDiscrete_M  = stand
       diRobotState   120    := 20;
    END_IF

20: // Move the axes to Position 2
    astAxisControl[c_udiAxis1].stI.rPosition   24    := 0;
    astAxisControl[c_udiAxis2].stI.rPosition   0     := 10;
    astAxisControl[c_udiAxis1].stI.rVelocity   2     := 2.0;
    astAxisControl[c_udiAxis2].stI.rVelocity   2     := 2.0;
    astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbsFALSE   := TRU
    astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbsTRUE    := TRU
    diRobotState   120    := 30;

30: // Comfirm movement completed
    IF astAxisControl[c_udiAxis1].stQ.nAxisState  Standstill  = standstill   3
           AND astAxisControl[c_udiAxis2].stQ.nAxisStateDiscrete_M  = stand
       astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbsFALSE   :=
       astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbsTRUE    :=
       diRobotState   120    := 40;
    END IF
```

vii. Speed up the robot by selecting the velocity parameters for Axis 1 and 2 as shown…



```
20: // Move the axes to Position 2
    astAxisControl[c_udiAxis1].stI.rPosition := 0;
    astAxisControl[c_udiAxis2].stI.rPosition := 10;
    astAxisControl[c_udiAxis1].stI.rVelocity := 2.0;
    astAxisControl[c_udiAxis2].stI.rVelocity := 2.0;
    astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbs
    astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbs
    diRobotState := 30;
```

viii. Change each of these to the prepared value 10 as before, then apply the change using <CTRL><F7>.



```
20: // Move the axes to Position 2
    astAxisControl[c_udiAxis1].stI.rPosition := 0;
    astAxisControl[c_udiAxis2].stI.rPosition := 10;
    astAxisControl[c_udiAxis1].stI.rVelocity
    astAxisControl[c_udiAxis2].stI.rVelocity
    astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbs
    astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbs
    diRobotState := 30;
```

ix. Apply exceptions using the OK to Pick and OK to place inputs.

x. Stop and restart the robot using the hardware inputs.

This completes the exercise

Managing the State Machine

In the previous exercise, we had to manually transition the state machine from the “wait” state into a “running” state. In this section we will apply branching conditions to automate this process.

Typically, we use the logic of an **IF THEN** branch to manage a state machine.

```
IF  <run conditions>  AND NOT One_Shot_Variable THEN
        Set the One_Shot_Variable
        Move the state machine into a specific run state (10)
ELSIF <reset conditions> THEN
        Move the state machine into the wait state (0)
        Reset the One_Shot_Variable
        Reset any control variables
END_IF
```

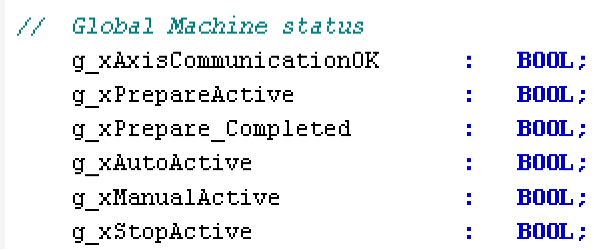
The logic must be applied “outside” of the CASE structure.

Run Conditions

For the Robot application, we will apply the Run conditions as follows:

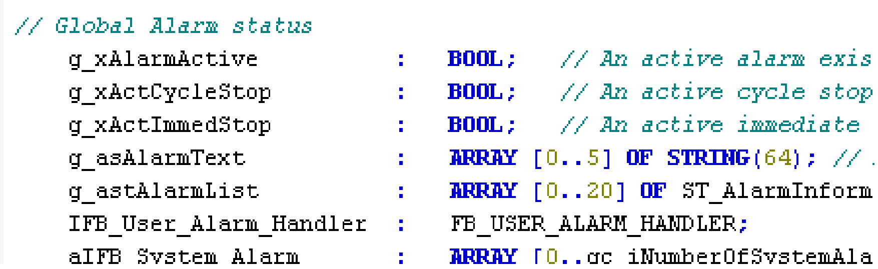
- Machine in Auto mode
- No active alarms
- Prepare completed (optional)

The template provides global variables that indicate the current machine status in the Global Variable list **GVL_Template >> GVL_Controller.**



```
//  Global Machine status
    g_xAxisCommunicationOK      :   BOOL;
    g_xPrepareActive            :   BOOL;
    g_xPrepare_Completed        :   BOOL;
    g_xAutoActive               :   BOOL;
    g_xManualActive             :   BOOL;
    g_xStopActive               :   BOOL;
```

Similarly, active Alarms are indicated by the BOOLEAN **g_xAlarmActive**.



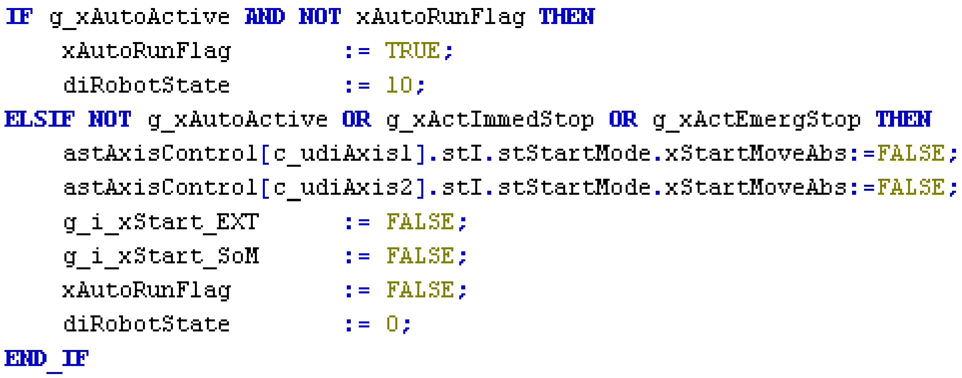
```
// Global Alarm status
    g_xAlarmActive          :   BOOL;    // An active alarm exis
    g_xActCycleStop         :   BOOL;    // An active cycle stop
    g_xActImmedStop         :   BOOL;    // An active immediate
    g_asAlarmText           :   ARRAY [0..5] OF STRING(64); // .
    g_astAlarmList          :   ARRAY [0..20] OF ST_AlarmInform
    IFB_User_Alarm_Handler  :   FB_USER_ALARM_HANDLER;
    aIFB System Alarm       :   ARRAY [0..gc iNumberOfSystemAla
```

Reset Conditions

Typical reset conditions could be:

- Active Alarms (requiring a IMMEDIATE stop)
- Transition out of Auto Mode

As an example, the following state machine manager could be used for the robot application.

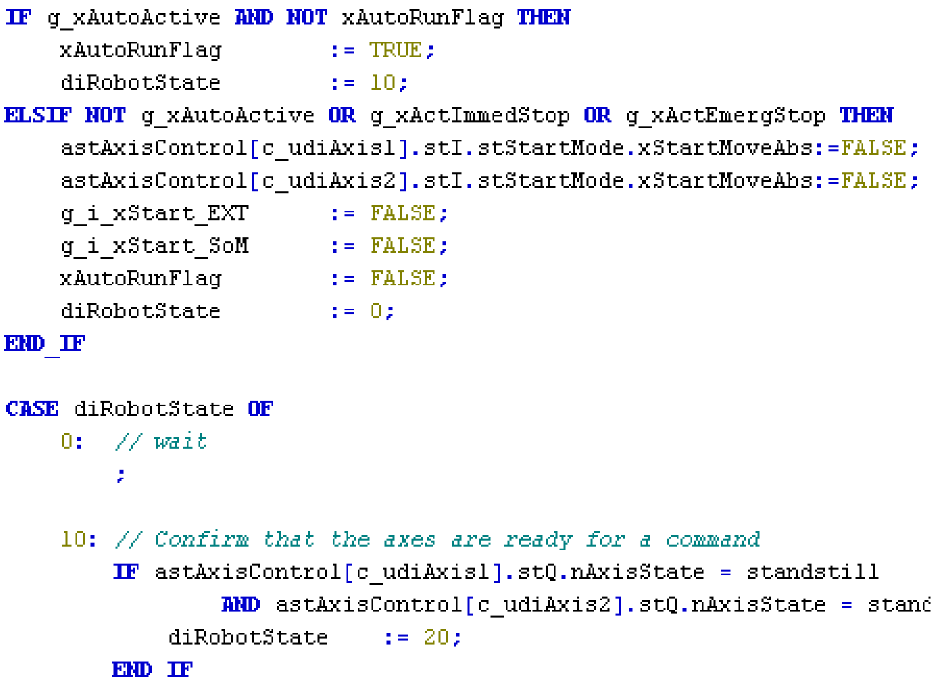


```
IF g_xAutoActive AND NOT xAutoRunFlag THEN
    xAutoRunFlag        := TRUE;
    diRobotState        := 10;
ELSIF NOT g_xAutoActive OR g_xActImmedStop OR g_xActEmergStop THEN
    astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbs:=FALSE;
    astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbs:=FALSE;
    g_i_xStart_EXT      := FALSE;
    g_i_xStart_SoM      := FALSE;
    xAutoRunFlag        := FALSE;
    diRobotState        := 0;
END_IF
```

Exercise – State Machine Manager

1. Apply the state manager code.

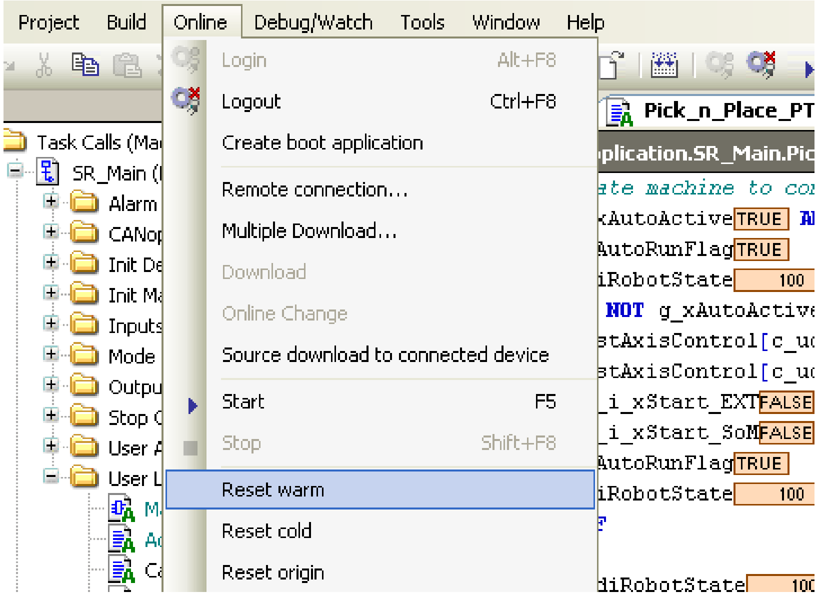
i. Copy and paste the state manager logic as indicated in the previous section into the state machine ***before*** the CASE statement.



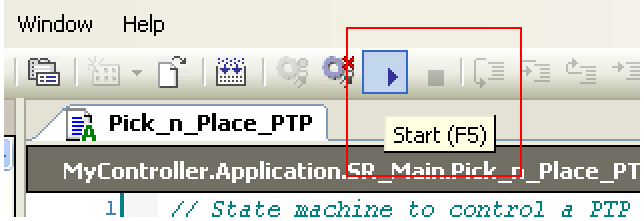
```
IF g_xAutoActive AND NOT xAutoRunFlag THEN
    xAutoRunFlag        := TRUE;
    diRobotState        := 10;
ELSIF NOT g_xAutoActive OR g_xActImmedStop OR g_xActEmergStop THEN
    astAxisControl[c_udiAxis1].stI.stStartMode.xStartMoveAbs:=FALSE;
    astAxisControl[c_udiAxis2].stI.stStartMode.xStartMoveAbs:=FALSE;
    g_i_xStart_EXT      := FALSE;
    g_i_xStart_SoM      := FALSE;
    xAutoRunFlag        := FALSE;
    diRobotState        := 0;
END_IF

CASE diRobotState OF
    0:  // wait
        ;

    10: // Confirm that the axes are ready for a command
        IF astAxisControl[c_udiAxis1].stQ.nAxisState = standstill
                AND astAxisControl[c_udiAxis2].stQ.nAxisState = stand
            diRobotState    := 20;
        END IF
```

ii. **Build** and **Save** the project as before. **Login** and download the change

iii. While In Run, apply a warm start to reset the controller and control variables



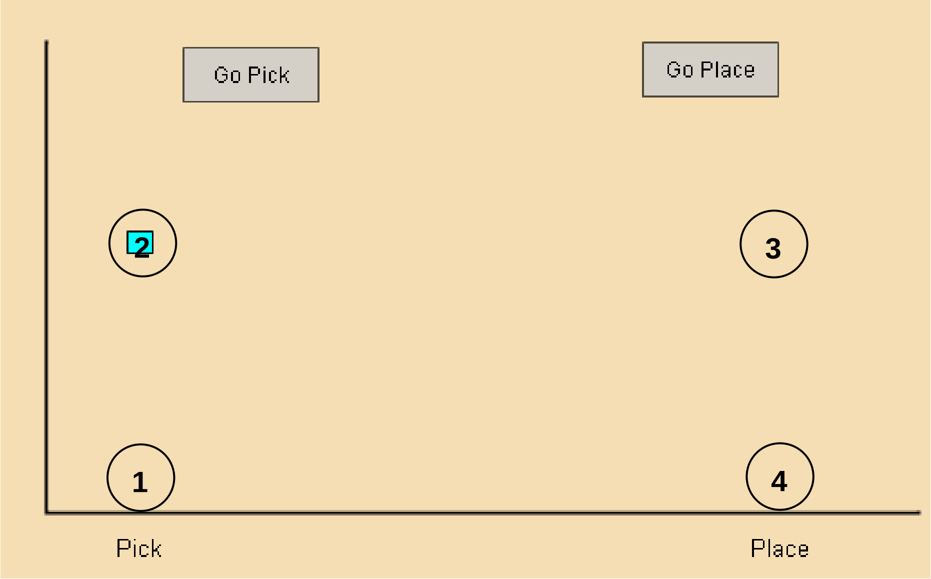
iv. Select Start once again to Run the program.



```
1 // State machine to control a PTP
```

v. **Prepare** the machine as before, and select Auto Mode.

This time, the robot will move to the rest position awaiting the Start button input.

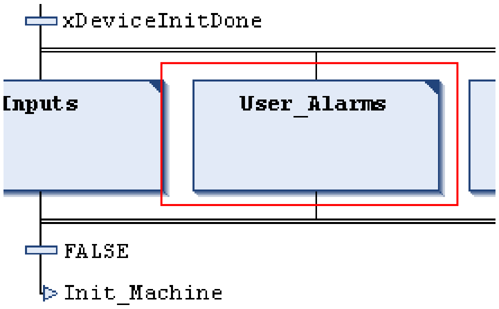


vi. Operate the robot as before testing the behavior with exception inputs.

This completes the exercise

User Alarms

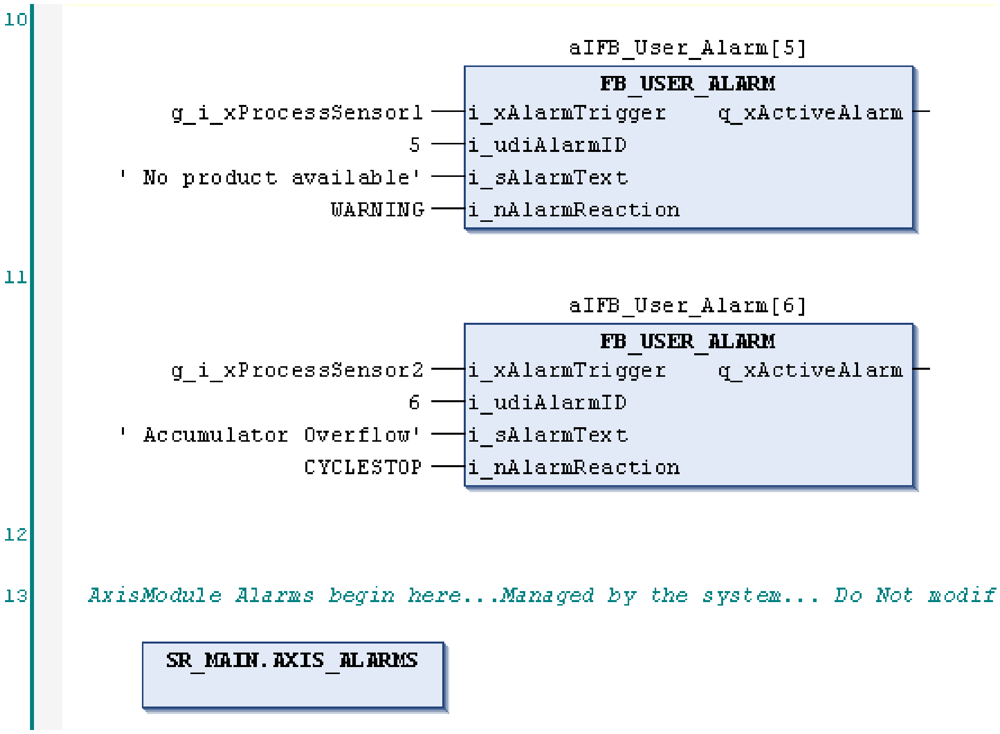
The template provides a convenient means of managing user alarms. Dedicated function blocks are located in the **User_Alarms** step, and can be copied and pasted as necessary to add, delete, or modify existing alarms.



The template automatically manages and displays axis alarms. Custom User alarms can be created using the **FB_User_Alarm** function block as shown.

Four input are required to create an alarm:

- An alarm trigger event (BOOL)
- An arbitrary (unique) ID number
- Text to display when the alarm occurs
- The required reaction to the alarm

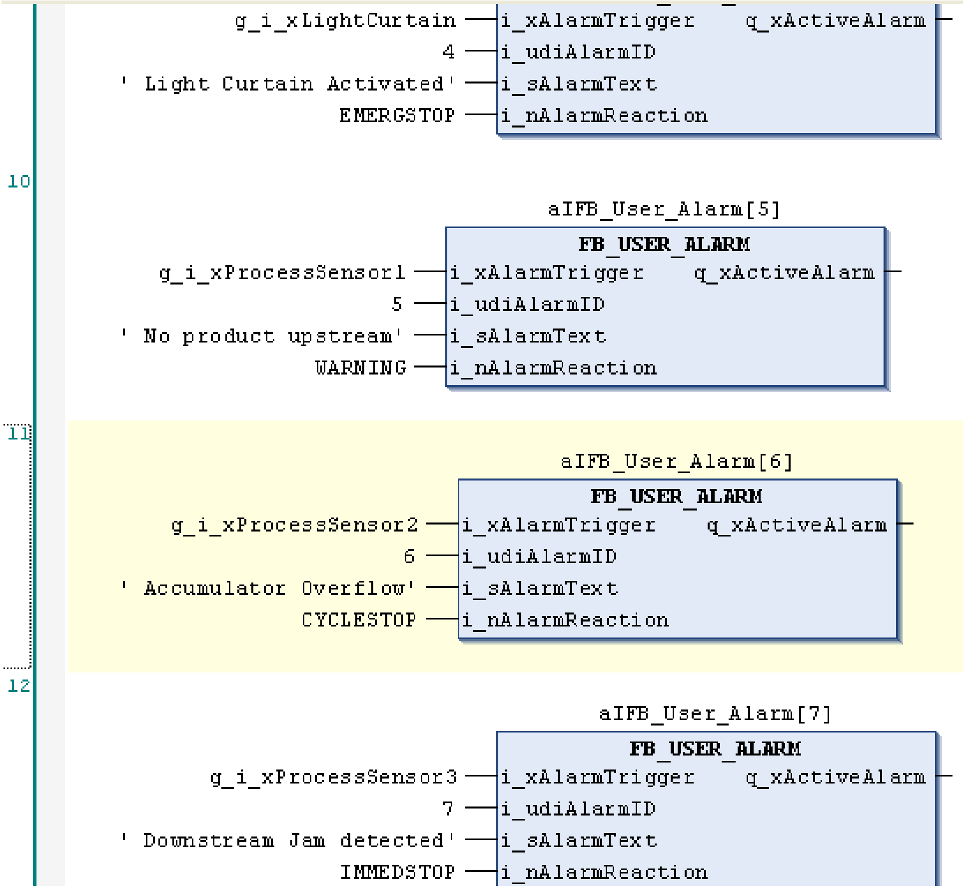


Alarm Reactions include:

Reaction	Description
WARNING	No system response other than display
CYCLESTOP	User –defined machine stop at end of cycle
IMMEDSTOP	Axes are stopped immediately, and Slave axes remain synchronized
EMERGSTOP	Axes are immediately and individually

The EMERGStop reaction would typically be managed by a Safety PLC or Enhanced Safety card for the LXM32 axis. An Emergency stop generates a category 1 controlled stop followed by removal of the STO (safe torque Off) inputs. It is included in the template for demonstration purposes only.

The predefined user alarms in the template can be used to illustrate a variety of Alarm conditions and reaction. In particular, User Alarms [4] through [7] illustrate each of the alarm reactions.



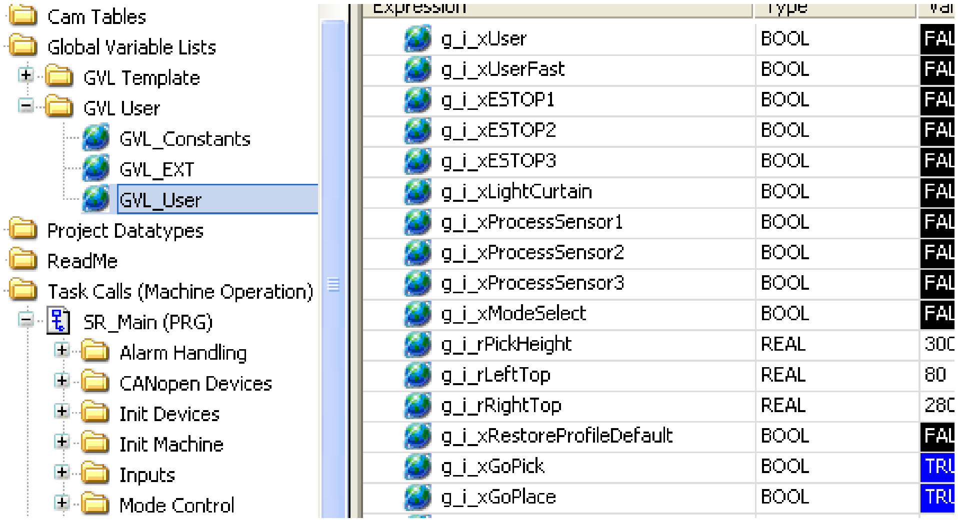
```
        g_i_xLightCurtain ─i_xAlarmTrigger    q_xActiveAlarm ─
                        4 ─i_udiAlarmID
' Light Curtain Activated' ─i_sAlarmText
                EMERGSTOP ─i_nAlarmReaction

                  aIFB_User_Alarm[5]
                     FB_USER_ALARM
     g_i_xProcessSensor1 ─i_xAlarmTrigger    q_xActiveAlarm ─
                       5 ─i_udiAlarmID
   ' No product upstream' ─i_sAlarmText
                 WARNING ─i_nAlarmReaction

                  aIFB_User_Alarm[6]
                     FB_USER_ALARM
     g_i_xProcessSensor2 ─i_xAlarmTrigger    q_xActiveAlarm ─
                       6 ─i_udiAlarmID
  ' Accumulator Overflow' ─i_sAlarmText
               CYCLESTOP ─i_nAlarmReaction

                  aIFB_User_Alarm[7]
                     FB_USER_ALARM
     g_i_xProcessSensor3 ─i_xAlarmTrigger    q_xActiveAlarm ─
                       7 ─i_udiAlarmID
' Downstream Jam detected' ─i_sAlarmText
               IMMEDSTOP ─i_nAlarmReaction
```

In the final Exercise, we will demonstrate how the template manages these Alarm types.

Exercise – Managing Alarms

1. Apply the pre-defined User alarms

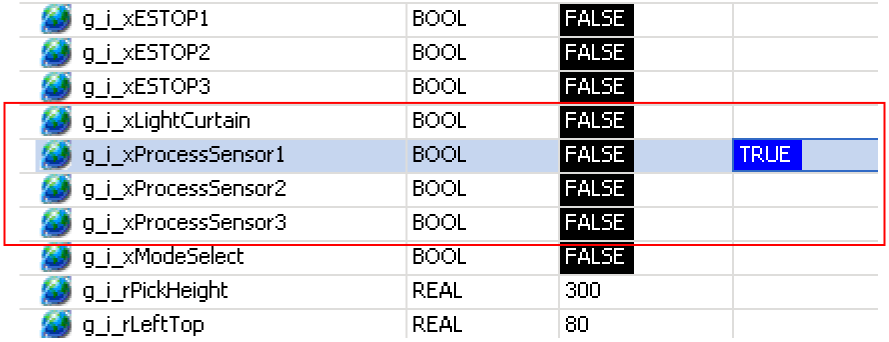
i. While operating the robot, navigate to the Global variable list **GVL User >> GVL_User**



- Cam Tables
- Global Variable Lists
  - GVL Template
  - GVL User
    - GVL_Constants
    - GVL_EXT
    - GVL_User
- Project Datatypes
- ReadMe
- Task Calls (Machine Operation)
  - SR_Main (PRG)
    - Alarm Handling
    - CANopen Devices
    - Init Devices
    - Init Machine
    - Inputs
    - Mode Control

Expression	Type	Va
g_i_xUser	BOOL	FAL
g_i_xUserFast	BOOL	FAL
g_i_xESTOP1	BOOL	FAL
g_i_xESTOP2	BOOL	FAL
g_i_xESTOP3	BOOL	FAL
g_i_xLightCurtain	BOOL	FAL
g_i_xProcessSensor1	BOOL	FAL
g_i_xProcessSensor2	BOOL	FAL
g_i_xProcessSensor3	BOOL	FAL
g_i_xModeSelect	BOOL	FAL
g_i_rPickHeight	REAL	30
g_i_rLeftTop	REAL	80
g_i_rRightTop	REAL	28
g_i_xRestoreProfileDefault	BOOL	FAL
g_i_xGoPick	BOOL	TRU
g_i_xGoPlace	BOOL	TRU

ii. Select any of the 4 user alarms as indicated, and prepare the value TRUE to trigger an alarm reaction.



g_i_xESTOP1	BOOL	FALSE	
g_i_xESTOP2	BOOL	FALSE	
g_i_xESTOP3	BOOL	FALSE	
g_i_xLightCurtain	BOOL	FALSE	
g_i_xProcessSensor1	BOOL	FALSE	TRUE
g_i_xProcessSensor2	BOOL	FALSE	
g_i_xProcessSensor3	BOOL	FALSE	
g_i_xModeSelect	BOOL	FALSE	
g_i_rPickHeight	REAL	300	
g_i_rLeftTop	REAL	80	

iii. Clear and acknowledge the alarm to restore operation.

iv. Repeat for the remaining user alarms.

This completes the exercise

This completes the Training Event !