MELSOFT to Logix5000 Application Conversion Guide

Application Solution

ALLEN-BRADLEY • ROCKWELL SOFTWARE

Rockwell Automation

LOGIX-AP009A-EN-P

Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. Safety Guidelines for Application, Installation, and Maintenance of Solid State Controls (publication SGI-1.1 available from your local Rockwell Automation sales office or online at http://literature.rockwellautomation.com) describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams. No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.

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

ATTENTION Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.

SHOCK HAZARD Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.

BURN HAZARD Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

Table of Contents

Preface

This document provides guidance for users and engineers who have used control systems based on one of these two platforms:

- MELSEC Q Series Controller
- Rockwell Automation Logix Programmable Automation Controller (PAC)

In addition, a user should have:

- A desire or need to take advantage of the PAC features, or are in the early stages of migrating from GX Developer to Logix.
- Specific GX Developer program code that they want to convert to effective, and efficient RSLogix 5000 code.

Use this document to help adopt good practices and to avoid common mistakes when converting the project to Logix.

Conversion versus Translation

The theme of conversion versus translation is one that is repeated in this application conversion guide. Simple translation is focusing only on the line of code and finding an equivalent in the Logix languages. To convert an application optimally, a user has to do more than just translate. For instance, a user may benefit from choosing a different programming language, using different programming techniques, and designing a different scheduling scheme to solve the same task. Conversion is performed in a context of a higher level design and knowledge of the strengths of the Logix system.

If a user has application code to convert, they need to understand the GX Developer program before they start the conversion – either by having been involved in its development or by reading the documentation of the program, and of the process that it controls. If the program or the process is unfamiliar or poorly documented, proper conversion will be difficult.

In some cases, if the documentation of the process and program is poor, it may be more efficient in terms of the overall project duration/cost to draw up a new specification and begin the Logix program with minimal time spent on translation from the old program.

Terminology

The Mitsubishi programmable logic controllers use the MELSOFT programming language. GX Developer is part of the Automation Software MELSEC suite. GX Developers support all MELSEC controllers from compact PLCs in the MELSEC Fx Series, to the modular PLCs, including the MELSEC System Q.

RSLogix 5000 software is used with Rockwell Automation Logix programmable-automation controllers. Logix is referred to as a programmable automation controller because it does so much more than a traditional general-purpose PLC. It provides an excellent control platform for multi-discipline control, a common namespace, a coordinated system time for truly scalable multi-CPU architectures, user-defined data types, and full NetLinx connectivity. NetLinx connectivity is the Rockwell Automation solution for networking technologies. The NetLinx networks include EtherNet/IP, ControlNet, and DeviceNet.

“Logix5000” refers to any of the ControlLogix, CompactLogix, GuardLogix, FlexLogix, DriveLogix, or SoftLogix controllers, or the RSLogix 5000 programming environment.

Additional Resources

Each section of this guide references other Rockwell Automation user manuals, selection guides, and documents where additional information can be found.

Publication Number	Publication Title
1756-SG001	ControlLogix Controllers Selection Guide
1769-SG001	1769 CompactLogix Controllers Selection Guide
1768-UM001	1768 CompactLogix Controllers User Manual
1769-SG002	Compact I/O Selection Guide
1756-RM094	Logix5000 Controllers Design Considerations Programming Manual
1756-PM001	Logix5000 Controllers Common Procedures Programming Manual
1756-RM003	Logix5000 Controllers General Instructions Reference Manual
1734-SG001	POINT I/O Selection Guide
1738-SG001	ArmorPoint I/O Selection Guide
1792-SG001	ArmorBlock MaXum I/O and ArmorBlock I/O Selection Guide
1794-SG002	FLEX I/O and FLEX Ex Selection Guide
NETS-SG001	NetLinx Selection Guide
VIEW-SG001	Visualization Platforms Selection Guide
IA-RM001	Integrated Architecture: Foundations of Modular Programming
6873-SG004	Encompass Program Product Directory
1756-PM010	Logix5000 Controllers Add-On Instructions Programming Manual
1756-RM087	Logix5000 Controllers Execution Time and Memory Use Reference Manual
IASIMP-RM001	IA Recommended Literature Reference Manual

PLC Logic Conversion Services Provided by Rockwell Automation

Service Features

Legacy products are often expensive to support and difficult to repair which can increase downtime and decrease production. For this reason, Rockwell Automation Customer Support now offers Program Conversion Services. These services are designed to reduce the cost and the time it takes to migrate from a legacy PLC to one of our current PAC or PLC-control platform families.

Program Conversion Services will convert legacy Rockwell Automation brand PLCs, or third-party programmable controller programs to run on a ControlLogix, SLC 500/MicroLogix, or PLC-5 programmable controller.

One-stop PLC Program Conversion Services

Migration to a current Rockwell Automation PLC platform from a legacy product will improve manufacturing processes, system reliability and flexibility. It will give more access to application processing power, and reduce equipment repair costs and spares inventory. With Program Conversion Services from Rockwell Automation Customer Support, existing programmable controller programs will be expertly and quickly converted to the new PLC family. Rockwell Automation Customer Support engineers can help in the migration of legacy equipment, or convert PLC systems to Rockwell Automation products while minimizing downtime and maximizing operational success.

Service Benefits

Specialists for each of the product platforms will be involved during the program conversion process. There are no hard-to-find anomalies in the logic caused by typing errors. In most cases, the entire datatable is reproduced with no data lost, and the original documentation is preserved with no re-typing of comments or symbols. Original Rockwell Automation brand programs can be in either 6200 Series format or AI Series format. New programs will be created in the appropriate RSLogix format.

Services Offered

Two program-conversion packages are available, as well as project-specific custom packages done on a case by case basis.

Basic Conversion Package

- The original programmable controller program will be converted to the appropriate Logix5000 controller.
- An error list is generated during the conversion and includes instructions that are not directly convertible, any addresses that may not have been converted which could include pointers, and indirect addressing.
- The program and error list will be returned to the customer for manual debugging and correction.

Conversion Plus Initial Clean-up Package

- The original programmable controller program will be converted to the appropriate Logix5000 controller.
- Rockwell Automation will correct and convert any instruction and/or addressing errors to the new processor family.
- The completed program will then be returned to the customer for final startup and debugging.

Additional Options

- Application-level telephone support during the start-up and debugging phase of the project.
- Consultation on system re-engineering, operator interface, architecture, and communication strategies.
- Training and on-site start-up is available as an added value from your local Global Sales and Solutions (GSS) office.
- Complete turn-key migration or upgrades are available from your local GSS/Engineered Systems Office.

Additional Program Conversions Available

- PLC-2 format to Logix5000 controller
- PLC-3 format to Logix5000 controller
- PLC-5/250 format to Logix5000 controller
- Modicon – Quantum, 984, 584, 380, 381, 480, 485, 780, 785 to Logix5000 controller
- Siemens – S-5, S-7 to Logix5000 controller
- TI – 520, 520C, 525, 530, 530C, 535, 560, 560/565, 565, 560/560T, 560T, 545, 555, 575 to Logix5000 controller
- GE Series 6 to Logix5000 controller

Program conversions of third-party programmable controllers to Rockwell Automation brand controller programs, is also available. Contact technical support for details.

To schedule a conversion project, or learn more about the Program Conversion Services, contact your local Rockwell Automation sales office or authorized distributor. Email us at raprogramconversion@ra.rockwell.com, or visit http://support.rockwellautomation.com and view Knowledge Base Document **G19154.**

IMPORTANT

Use consultation services for re-engineering, typically to expand the system functionality and not to change out hardware due to obsolete or related reasons. SLC to Logix format, and PLC-5 to Logix format conversions, and PCE comment generation are built into the RSLogix 5000 software.

Chapter 1: Hardware Conversion

Introduction

The objective of this chapter is to provide guidance to a user or engineer who needs to determine the correct Logix hardware as a replacement for the existing MELSEC equipment.

This chapter describes how to select controllers, local I/O, remote I/O, networks, and HMI. There is also a section on distributed controller architecture and hardware conversion examples of the most often used MELSEC modules.

MELSEC Q Controllers

The MELSEC System Q is a modular PLC with multiprocessor technology. The heart of a PLC consists of a base unit, a power supply module, and at least one CPU module. The CPU executes the instructions in the PLC program. Depending on the application more modules can be installed on the base unit, such as input and output modules (I/O modules), and special function modules. The power supply module supplies the voltage for installed modules.

Communication between the individual modules and the CPU is performed via an internal bus connection on the base unit. The base unit on which the CPU is mounted is called the main base unit. The base units of the MELSEC System Q are available in 5 versions with up to 12 module slots.

This table lists a relevant selection of MELSEC Q Controllers CPU modules.

QCPU Basic	QCPU High Performance	Process CPU	Redundant CPU	QCPU Universal Model
Q00JCPU	Q02CPU	Q12PHCPU	Q12PRHCPU	Q01UCPU
Q00CPU	Q02HCPU	Q25PHPCU	Q25PRHCPU	Q03UDCPU
Qo01CPU	Q06HCPU			Q04UDHCPU
	Q25HCPU			Q06UDHCPU

Mitsubishi Controllers Classification

A Series	Q Series	System Q
AnA/Anu AnAS/AnUS AnN AnS	QnA	Q- single processor types Q- multi processor types
CPU types	CPU types	CPU types
A2A, A2A-S1, A2U, A2U-S1, A3A, A3U A2AS, A2AS-S1, A2AS-S30, A2AS-S60, A2US, A2US-S1 A1, A2, A2C, A3M, A3N A1S, A1S-S1, A2S, A2S-S1	Q2A/Q2AS, Q2A-S1/Q2AS-S, Q3A, Q4A, Q4AR	Q00J, Q00, Q01 Q02, Q02H, Q06H, Q12H, Q25H PC-CPU-Module: PPC-CPU686 (MS)-64 PPC-CPU686 (MS)-128 Up to 4 multi processor type CPUs can be used in a multi-CPU system, sharing control and communications.

Features

Controller	I/O	Logic Instruction Cycle Period	Memory
Alpha 2	10-28	20us	200 Function Block
FX Family	10-384	.065-.55us	2-64K
AnSH	32-1024	.075-.2us	28-60K
System Q	32-8192	.034-.2us	28-525K

Program Capacities

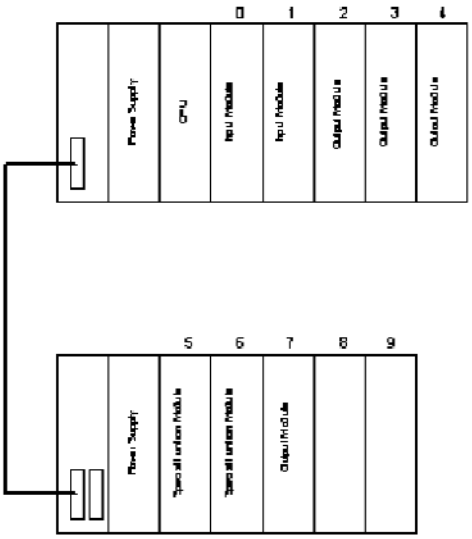
CPU	Memory
Basic Model QCPU	
Q00JCPU, Q00CPU	8K
Q01CPU	14K
High Perfomance Model QCPU	
Q02CPU, Q02HCPU	28K
Q06HCPU	60K
Q12HCPU	124K
Q25HCPU	252K
Process CPU	
Q12PHCPU	124K
Q25PHCPU	252K
Redundant CPU	
Q12PRHCPU	124K
Q25PRHCPU	252K
Universal Model QCPU	
Q02UCPU	20K
Q03UDCPU	30K
Q04UDHCPU	40K
Q06UDHCPU	60K

Also, the QnU series, which is the next level in Q performance, is available in the following models: **Q26UDH, Q13UDH, Q06UDH, Q04UDH, Q03UD and QO2U.**

Expandability

When more slots for additional modules are required on a main base unit, an extension base unit can be added. Extension cables connect the base units to one another. These cables also provide the installed modules with power if the extension base units do not have their own power supply. Up to seven extension base units can be connected to a main base unit. The total number of I/O and special function modules in all base units is 64.

Extension of Base Units



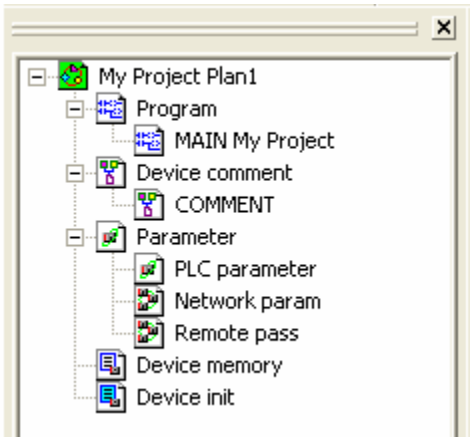
I/O Systems

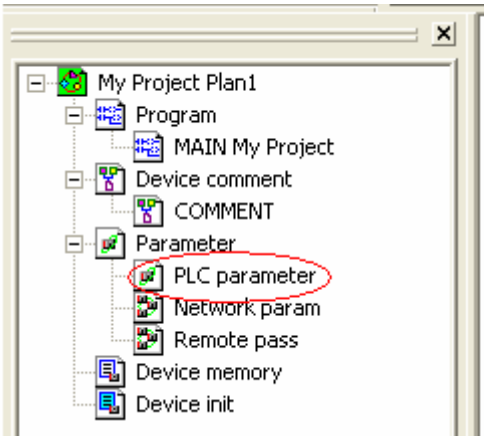
MELSEC Local I/O

A CPU of the MELSEC System Q automatically recognizes the slots available in main and extension base units and assigns addresses to the inputs and outputs accordingly. However, the assignment can also be done with the aid of the programming software. Therefore, slots can be left empty or addresses can be reserved for future extensions.

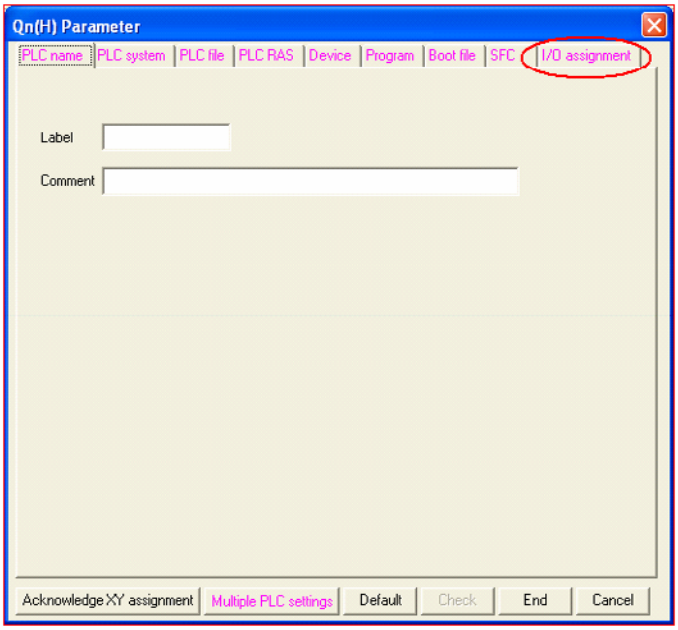
Before a program can be sent to the PLC CPU, an assignment list of the inputs and outputs must be built. This is required for the CPU to know the correct combination of modules in the rack so that it can interface with each module in the correct manner.

From the project **Data List** window, double-click the folder and files icon to open the parameter option.

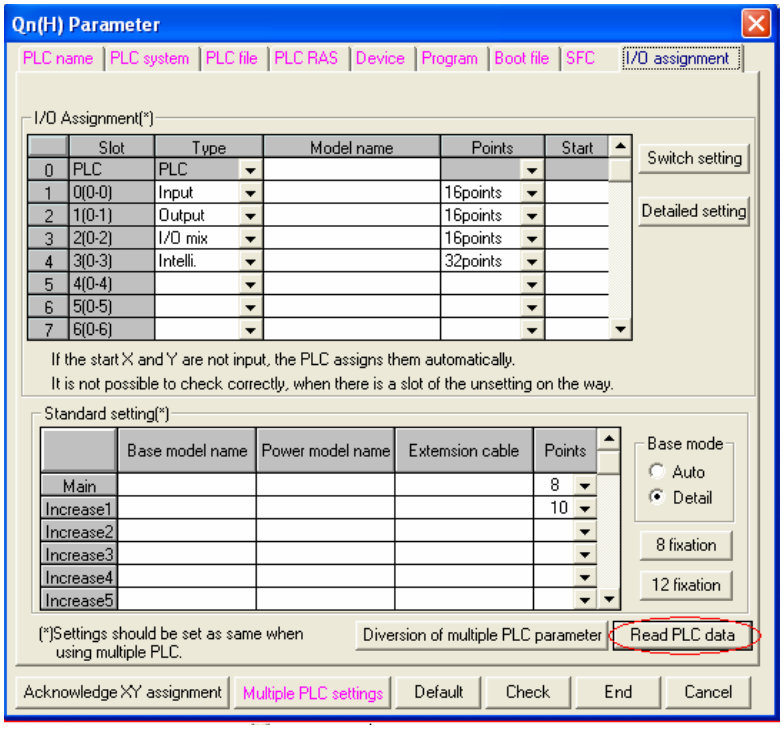




Click the **I/O assignment** tab.



Click **Read PLC Data.**



This causes the module configuration information to be read back to the screen.

Type the names of the modules in the rack by examining the module information on the front end of each unit.

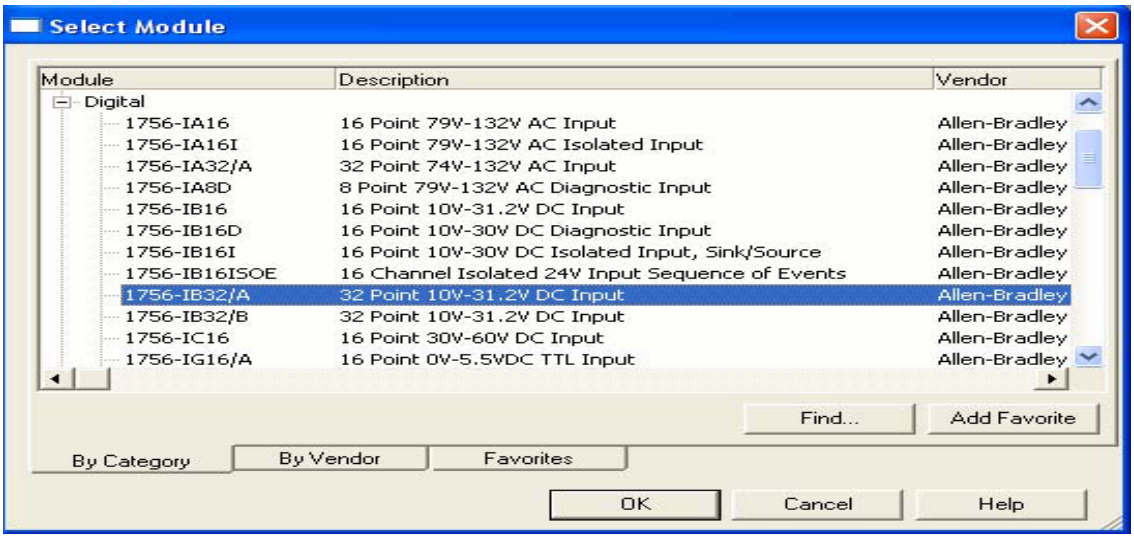
Logix Local I/O

A wide range of ControlLogix and CompactLogix I/O modules is available. The CompactLogix 1769 I/O is cost-optimized for just enough functionality as often requested by OEMs. The ControlLogix 1756 I/O family provides high feature/functionality for the most demanding applications that need to meet specific performance levels, as often requested by end users.

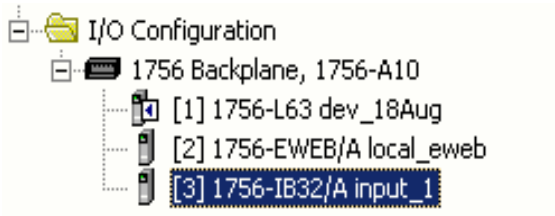
- CompactLogix modules are mounted to a standard DIN rail and a special coupling system secures electrical and mechanical connections to adjacent modules.
- MELSEC modules are fixed to the rail only and not to each other (other than by the electrical U-connector).
- ControlLogix modules are mounted in the 1756 racks.
- For 1769-L23 controllers, the maximum number of I/O modules attached to the controller is two. For 1769-L32C and 1769-L32E controllers, the maximum number of I/O modules attached to the controller is 16.
- For 1769-L35CR and 1769-L35E controllers, the maximum number of I/O modules attached to the controller is 30.
- For 1769-L43 controllers, the maximum number of I/O modules attached to the controller is 16.
- For 1769-L45 controllers, the maximum number of I/O modules attached to the controller is 30.
- For 1756 controllers, the number of slots in the rack defines the maximum number of I/O modules. It can be 4, 7, 10, 13, or 17.
- On both ControlLogix and CompactLogix platforms, further I/O can be networked via CIP networks. EtherNet/IP, DeviceNet, and ControlNet networks provide tight and seamless I/O integration.

Selection and Configuration of Logix I/O Components

From the I/O Configuration branch of your project tree, the Logix5000 library of device profiles can be accessed. These profiles provide full wizard-driven configuration for complete, easy-to-use integration into the datatable and intuitive programmable control over each module's functionality, such as scaling, alarming, and diagnostics.

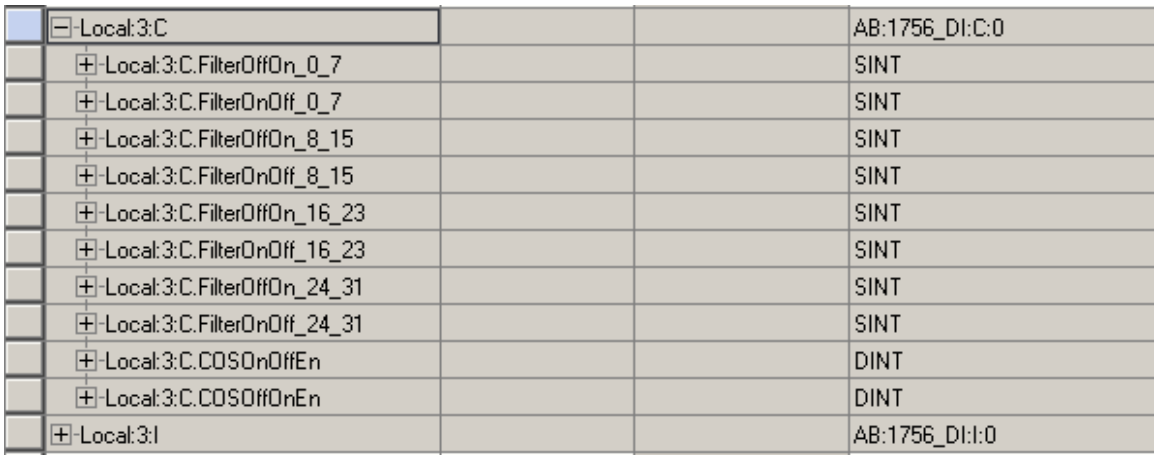


Select an item to add it to the I/O configuration.



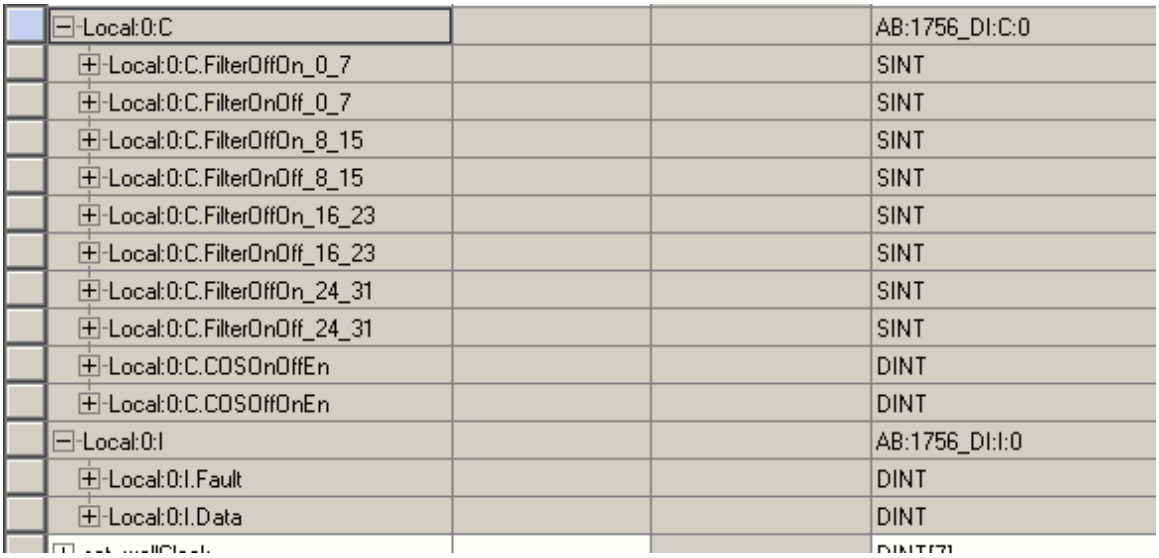
The device profile tags for the new I/O module have been added automatically to the controller scoped tag database.

The view below shows the tags partly expanded.



⊟ Local:3:C			AB:1756_DI:C:0
⊞ Local:3:C.FilterOffOn_0_7			SINT
⊞ Local:3:C.FilterOnOff_0_7			SINT
⊞ Local:3:C.FilterOffOn_8_15			SINT
⊞ Local:3:C.FilterOnOff_8_15			SINT
⊞ Local:3:C.FilterOffOn_16_23			SINT
⊞ Local:3:C.FilterOnOff_16_23			SINT
⊞ Local:3:C.FilterOffOn_24_31			SINT
⊞ Local:3:C.FilterOnOff_24_31			SINT
⊞ Local:3:C.COSOnOffEn			DINT
⊞ Local:3:C.COSOffOnEn			DINT
⊞ Local:3:I			AB:1756_DI:I:0

In addition to the I/O data, tags created in RSLogix 5000 software also contain descriptive configuration and fault data.



⊟ Local:0:C			AB:1756_DI:C:0
⊞ Local:0:C.FilterOffOn_0_7			SINT
⊞ Local:0:C.FilterOnOff_0_7			SINT
⊞ Local:0:C.FilterOffOn_8_15			SINT
⊞ Local:0:C.FilterOnOff_8_15			SINT
⊞ Local:0:C.FilterOffOn_16_23			SINT
⊞ Local:0:C.FilterOnOff_16_23			SINT
⊞ Local:0:C.FilterOffOn_24_31			SINT
⊞ Local:0:C.FilterOnOff_24_31			SINT
⊞ Local:0:C.COSOnOffEn			DINT
⊞ Local:0:C.COSOffOnEn			DINT
⊟ Local:0:I			AB:1756_DI:I:0
⊞ Local:0:I.Fault			DINT
⊞ Local:0:I.Data			DINT

MELSEC Remote I/O

To connect a remote I/O station and the system with the PLC CPU, only a network module and a network cable is required. Depending on the selected CPU type, up to 4096 local (on main and extension base units) and up to 8192 remote I/O points can be addressed. Remote communication is possible by using interface modules.

Ethernet Interface Modules - The modules QJ71E71/E71-100 and QD71E71-B2 are used on the PLC side to connect a host system, for example, a personal computer or workstation and the System Q via the Ethernet network. Up to 16 communication lines can be opened for concurrent data communication.

MELSECNET Modules - The modules QJ71BR11 and QJ71LP21 are used to connect the MELSEC System Q to a MELSECNET/10 or MELSECNET/H network. Communication with other PLCs, personal computers, or remote I/O is possible.

Master/Local Module for CC-Link - The QJ61BT11N module is applicable as a master or local station in a CC-Link system and manages the connection of remote inputs and outputs. The communication between the remote modules and the master module is performed automatically. With one master module, a system can be extended to up to 2048 remote I/O points.

Profibus/DP Interface Modules - The PROFIBUS/DP master modules QJ71PB92D and QJ71PB92V as well as the QJ71PB93D PROFIBUS/DP slave module enable PLCs of the System Q to communicate with other PROFIBUS devices. The master station can communicate with up to 60 slave units.

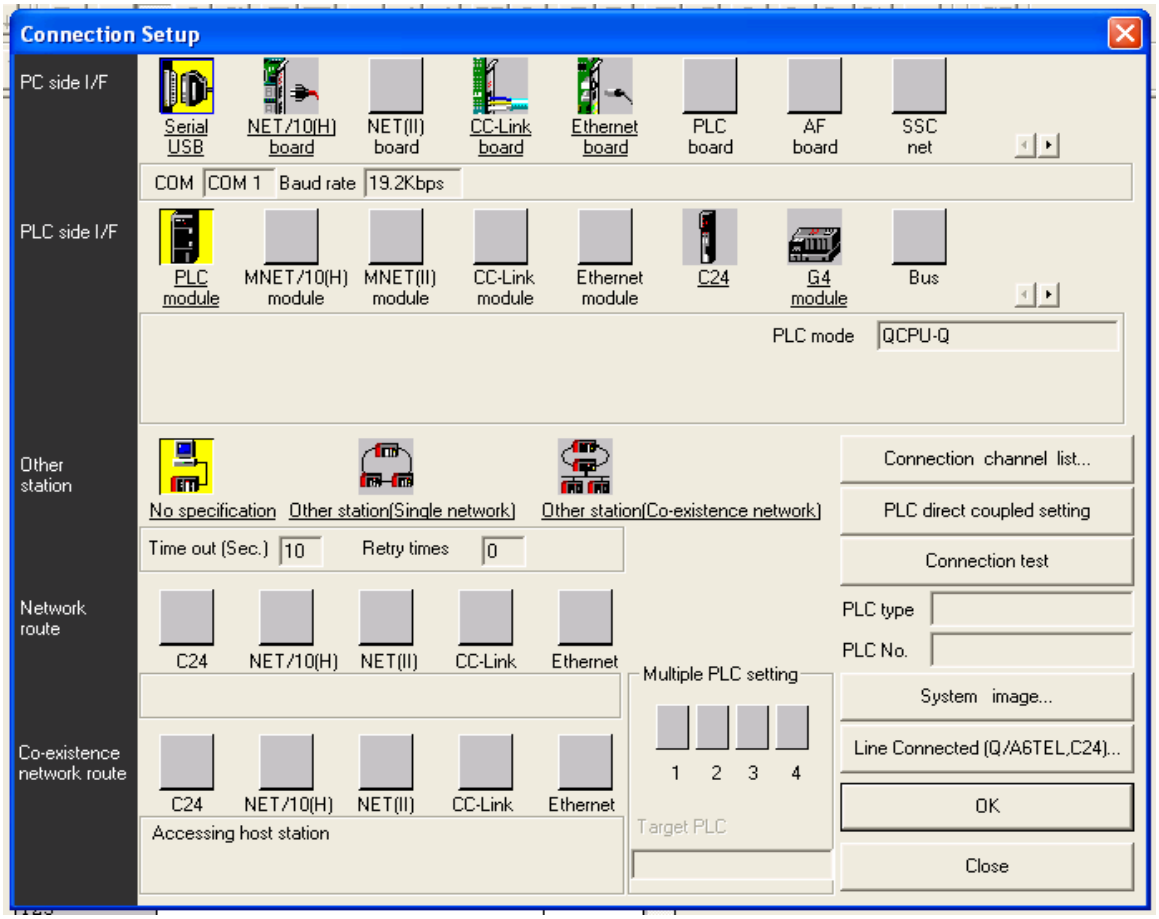
DeviceNet Master Module - The QJ71DN91 module connects a System Q series PLC with the DeviceNet network. Network integration of low-level terminal equipment and the positions of master and slave stations are user selectable.

Master Modules for AS Interface – The QJ71AS92 module is a master module for connecting System Q to the AS-interface system. Up to 62 slave units can be configured across 2 networks. Up to 496 digital inputs/outputs can be driven via the master.

Web Server Module - The web server module QJ71WS96 enables the remote control monitoring of a System Q series PLC.

Serial Communication Modules - The modules QJ71C24 and QJ71C24-R2 enable communication with peripheral devices via a standard serial interface.

Connection Setup to Go Online with the PLC



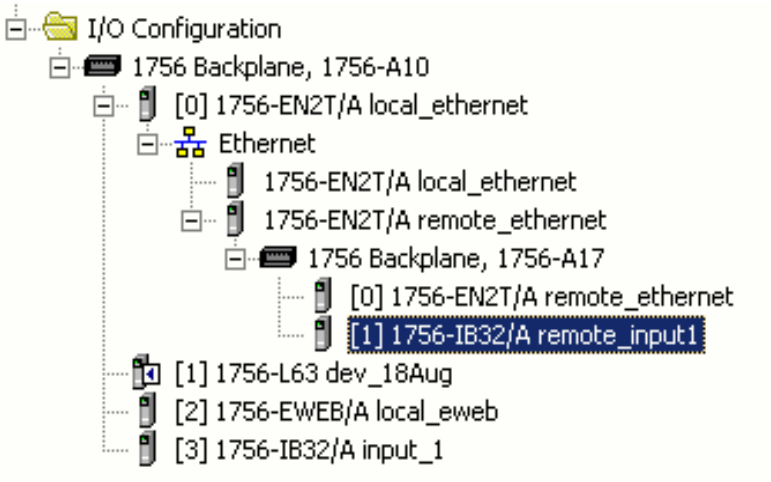
Logix Distributed I/O

Rockwell Automation Distributed I/O includes 1756, 1769 I/O modules and various Distributed I/O platforms, such as POINT I/O, FLEX I/O, ArmorPoint I/O, and ArmorBlock I/O systems.

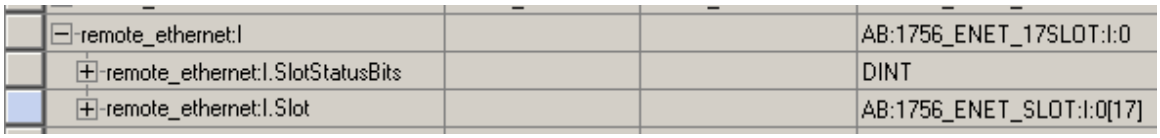
The I/O modules are connected to the network by using a communication module or communication adapter, or directly by using a built-in communication interface.

Configuration of Logix Distributed I/O

All I/O configurations are done in the project tree of RSLogix 5000 software. From the I/O Configuration branch, a communication module can be inserted for a chosen network type. This window shows an addition of a remote 1756-IB32 I/O module connected via an EtherNet/IP network.



Notice, that tags corresponding to the remote I/O module have been added automatically to the controller scope tag database.

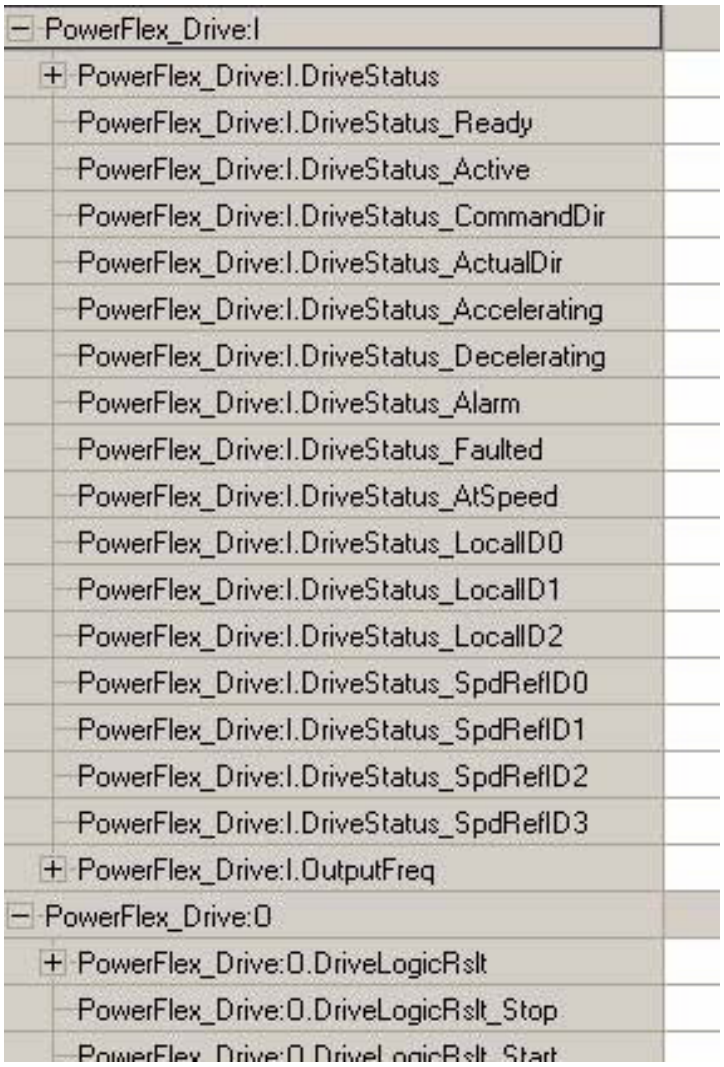


A networked variable-speed drive, such as PowerFlex drive, can be added in the same way.

Once the module has been added to the I/O Configuration, the module properties can be accessed by double-clicking on the device.

Again, RSLogix 5000 software will generate the new tags automatically for any device with a profile in RSLogix 5000 software and connect on an EtherNet/IP or ControlNet network. DeviceNet and GuardLogix Safety I/O are integrated in the same way. Other DeviceNet devices need to be set up by using the RSNetWorx configuration software.

The example below shows a PowerFlex drive device profile tag in RSLogix 5000 software. These types of tags are available for hundreds of Rockwell Automation devices.



Networks in MELSEC

Mitsubishi offers systems based on a three level network:

- Production Level
- Control Level
- Command Level

The above levels are also divided into:

- Open networks.
- MELSEC networks.

Production Level

A field network that links control devices such as a PLC with remote inputs and outputs, inverters, and operator terminals is in the lowest network level in production locations.

Control Level

A control network that links control devices for example, PLC and CNC, is in the middle network level in production sites. It is designed to transfer data directly related to the operations and motions of machinery and equipment between the control devices.

Command Level

The command level is in the highest network level in production fields in an information network. The level is designed to transfer production control information, quality control information, facility operating status, and other information between the PLC or facility controller and the production control computer.

Open Networks

Open networks are manufacturer independent, that is those networks that are also used by other manufacturers. Thus, the communication between a MELSEC PLC and devices of third-party manufacturers is possible.

Ethernet Network

The Ethernet network is the most widespread network for connection of information processors such as personal computers and workstations. The Ethernet network is a platform for a very wide range of data communication protocols.

TCP/IP provides logical point-to-point links between two Ethernet stations. Using the TCP/IP protocol, a process supervision system can request up to 960 data words per query if the MELSEC System Q module is used.

PROFIBUS/DP Network

The open PROFIBUS/DP network enables data exchange with a wide variety of slave devices including the following:

- Remote digital and analog I/O
- Frequency inverters
- Operator terminals
- A range of other devices from third-party manufacturers

CC-Link Network

The open fieldbus and control network CC-Link provides fast data communication with different devices. The following components from Mitsubishi Electric, among others, can be integrated:

- MELSEC PLC systems
- Remote digital and analog I/O
- Positioning modules
- Frequency inverters
- Operator terminals
- Robots
- Third-party devices like bar code readers

In addition to the cyclic transmission of word data, CC-Link systems handle transient transmission (message transmission) as well. This enables data communication with intelligent devices, such as display devices, bar code readers, measuring devices, personal computers, and PLC systems (up to 24 CPUs) as well as analog and digital devices.

DeviceNet Network

The DeviceNet network is a network for the integration of low-level terminal equipment. Up to 64 devices including a master can be integrated in 1 network.

AS Interface (AS-i)

The AS Interface is an internationally standardized interface for the lowest field bus level. It is suitable for controlling **A**ctuators, like solenoids, or indicators and **S**ensors, hence the name AS-i.

CANopen

CANopen is an “open” implementation of the **C**ontroller **A**rea **N**etwork (CAN). CANopen networks are used for connecting sensors, actuators, and controllers in industrial control systems, medical equipment, maritime electronics, railways, trams, and commercial vehicles. CANopen network modules are available for the MELSEC FX family of controllers.

MELSECNET/10/H

MELSECNET/10 and MELSECNET/H are high-speed networks for data exchange between MELSEC PLCs. Up to 255 MELSECNET/10/H networks can be linked together. The built-in router functionality allows easy data transfer from one network to another. In parallel to the cyclic data exchange, it is also possible for any station to send data to and read data from any other station, even across several networks.

MELSECNET/10 gives you a wide choice of cable types and topologies - from coaxial bus with a maximum of 500 m (1640 ft), over a coaxial duplex loop, to a fiber-optic duplex loop for distances of up to 30 km (18 mi).

Network Type	MELSEC Networks
Command	TCP/IP Ethernet
Control	CC-Link MELSECNET/10 MELSECNET/H
Production	CC-Link MELSECNET FX-PPN

Networking other than CC-Link and MELSECNET is very difficult. There are no tools like RSNetWorx for DeviceNet systems, so the processor automatically selects a configuration that can sometimes cause a problem.

Networks in Logix

NetLinx is the term identifying the Rockwell Automation solution in the area of networking technologies. The following are the primary networks used in Logix systems:

- EtherNet/IP
- ControlNet
- DeviceNet

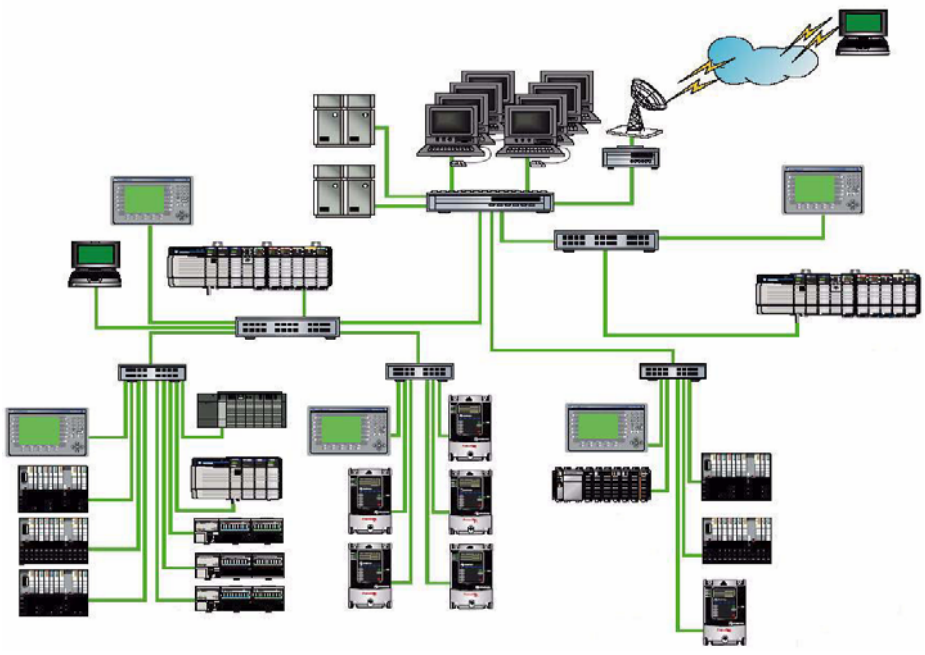
These networks have a variety of notable features. All are designed under the Common Industrial Protocol (CIP) which enables users to control, configure, and collect data over any of the NetLinx networks. As a result, data can flow between different networks without any need for protocol translation software or proxies.

EtherNet/IP Network

The EtherNet/IP network offers a full suite of control, configuration, and data collection services. It uses TCP/IP for general messaging/information exchange and UDP/IP for I/O messaging. It is most often used in these types of configurations:

- General I/O control
- Data exchange among controllers
- Connecting many computers
- Connecting many devices
- Connectivity to enterprise systems
- Integration of Safety devices
- Motion control (future)

Typical EtherNet/IP Example

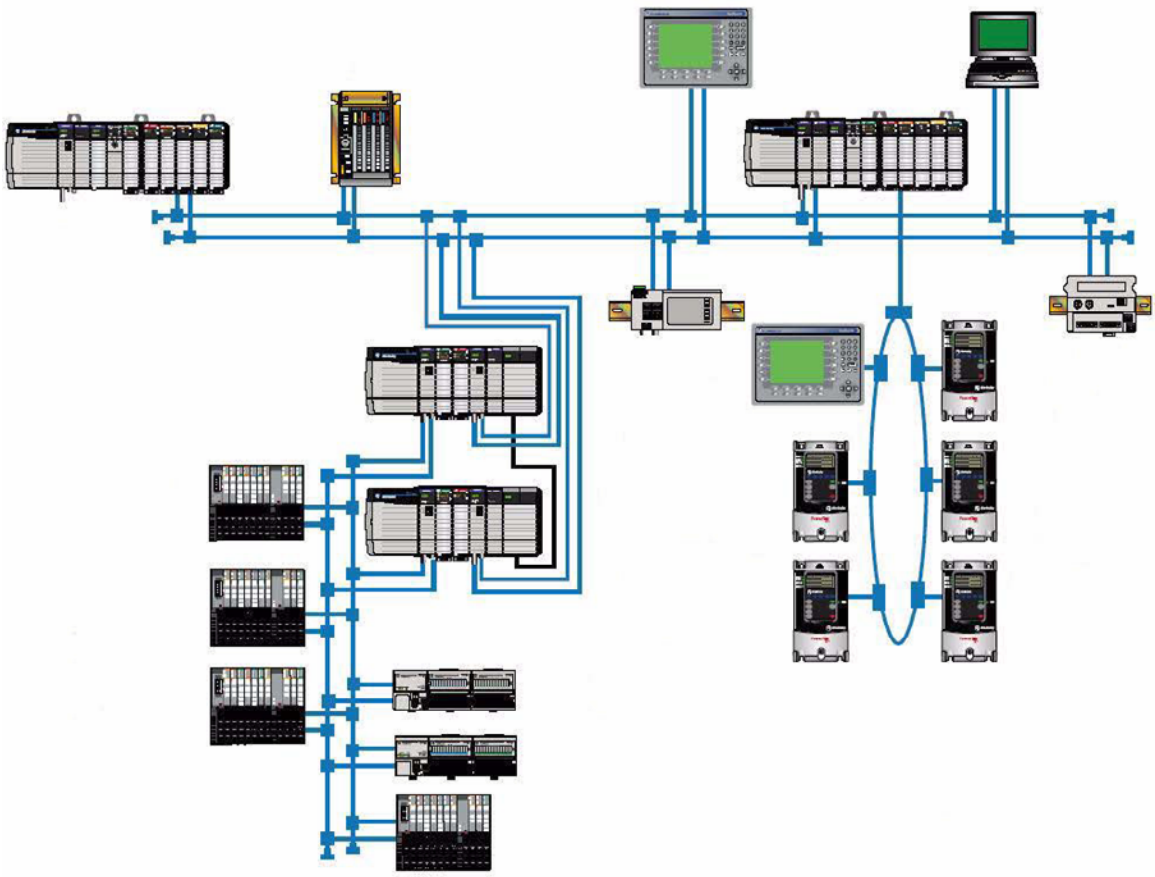


ControlNet Network

The ControlNet network is a real-time control network. It provides transport of both time-critical I/O and interlocking data and messaging data, including upload/download of programming and configuration data on a single physical-media link. It is most often used in these types of configurations:

- General I/O control
- Data exchange among controllers
- Backbone to multiple distributed DeviceNet networks

Typical ControlNet Example

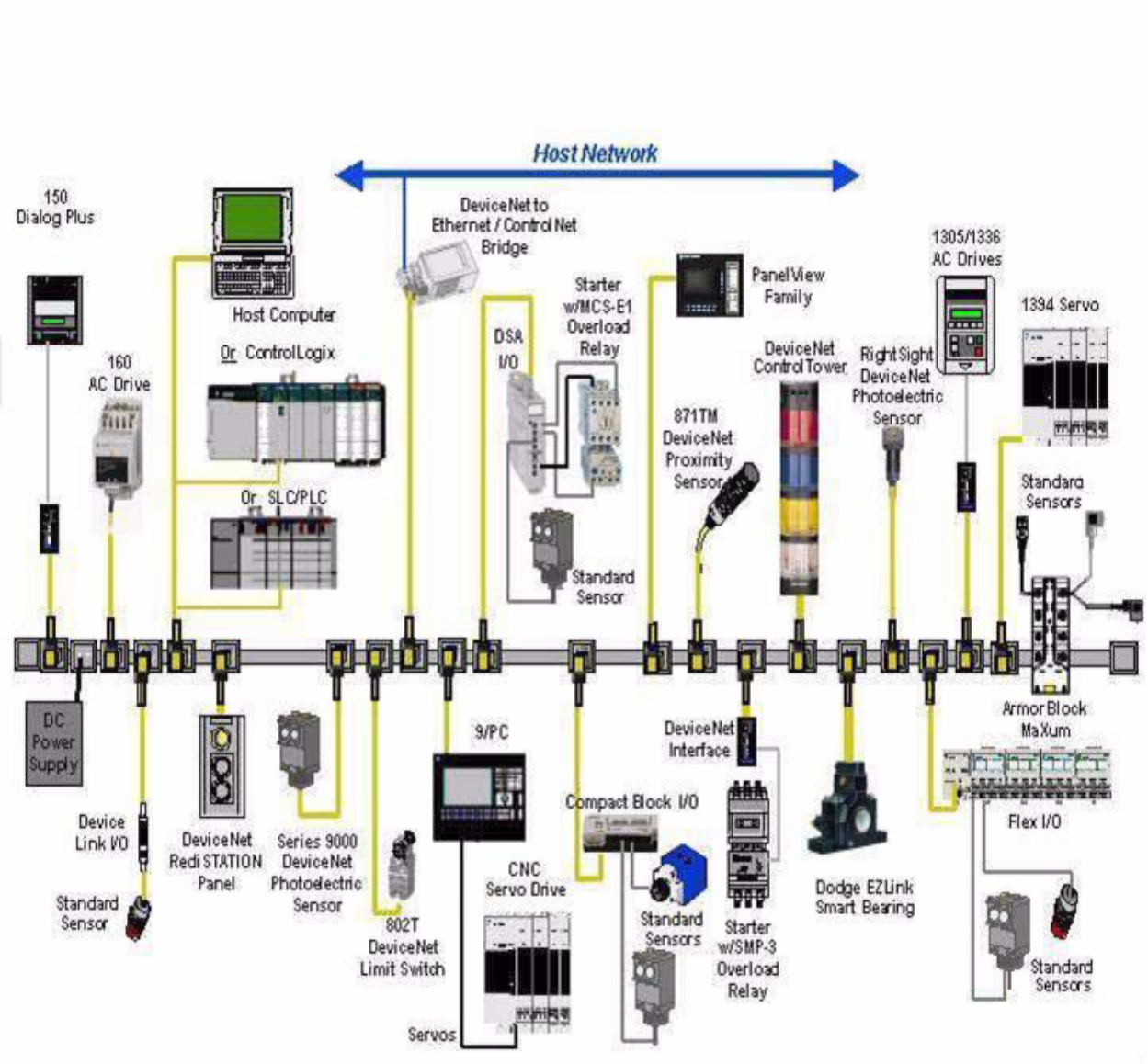


DeviceNet Network

The DeviceNet network is a solution for low-level industrial device networking. Designed for devices with a low data volume per device for real time operation, it is most often used in these types of configurations:

- Applications containing distributed devices with a few points
- Network of third-party drives and other “simple” third-party devices
- Systems in which devices need to be connected directly to the network with data and power in the same connection
- When advanced diagnostic information is required

Typical DeviceNet Example



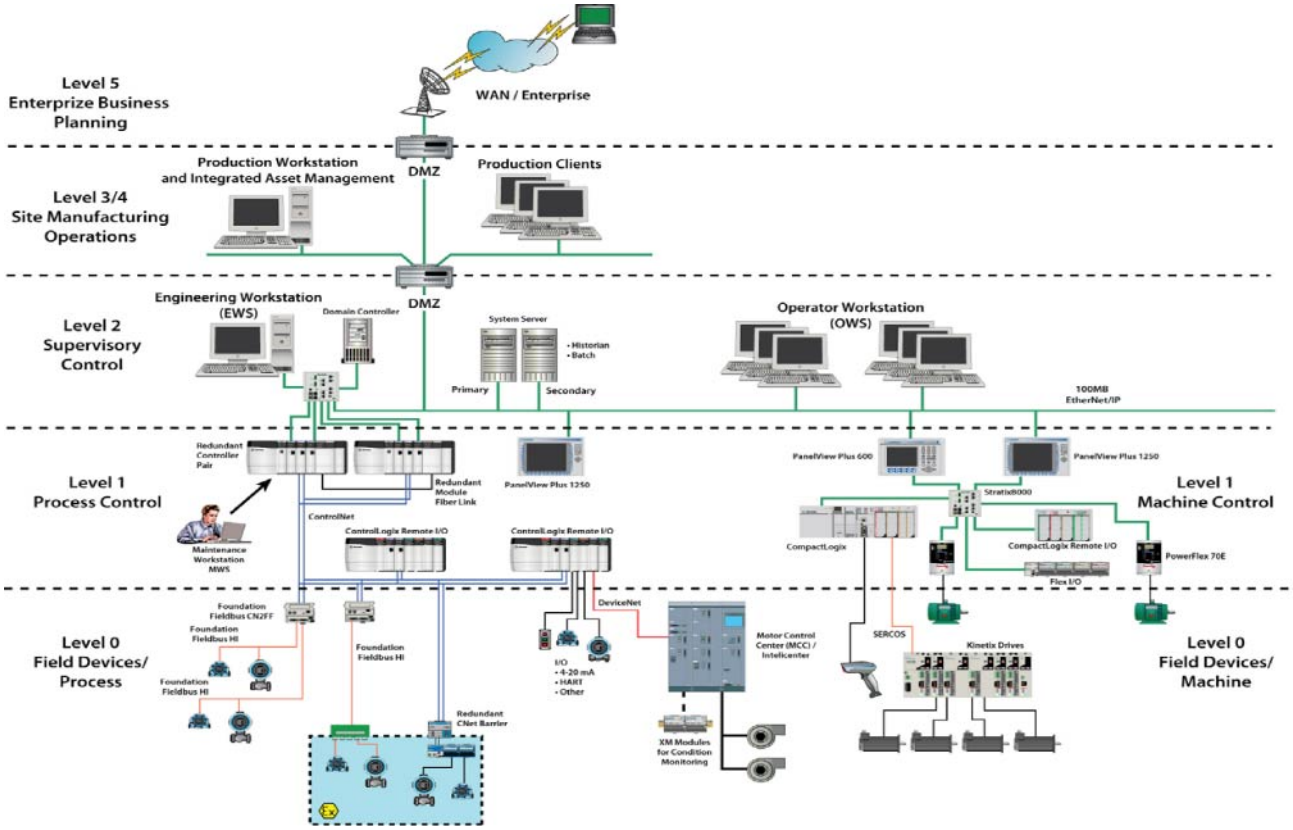
Interconnecting NetLinx Networks

There are two common ways to interconnect NetLinx networks:

- Through the communication backplane that allows multiple network links at once
- Through communication linking devices that link two networks together in a seamless fashion

No controller or programming is required in either of these approaches.

Example of a Control System Based on the NetLinx Networks



Chapter 2: Logix Features that May Not be Familiar to GX Developer Users

Introduction

This chapter describes Logix features that may not be familiar to GX Developer users.

Certain features of the Logix system are easier to use and maintain than the GX Developer. For example, data is organized into tag databases with no absolute addresses. In GX Developer, data items have absolute addresses that are selected by the programmer in defined memory areas.

In other respects, the structure of Logix is quite similar to GX Developer, but it is presented differently for instance, beneath the surface the task structure is similar to GX Developer's PoU.

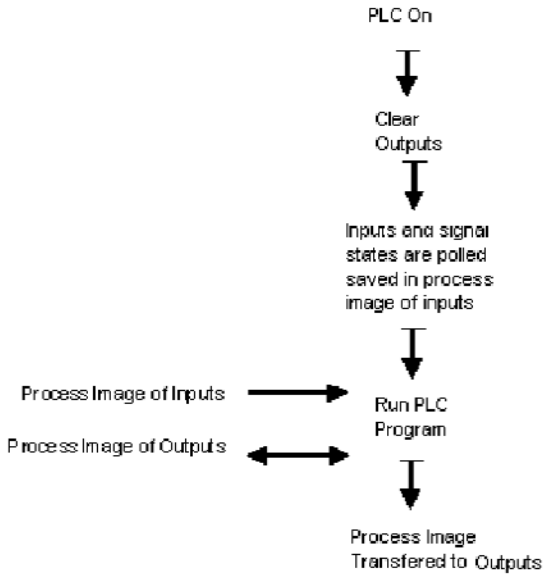
This Chapter contrasts those features that are different (such as tags) and compares those features that have underlying similarities (such as tasks).

The objective is to:

- Provide the GX Developer user converting to Logix with information that will make the design process easier and quicker.
- Demonstrate what Logix can do so users do not attempt to recreate what exists within controller firmware.

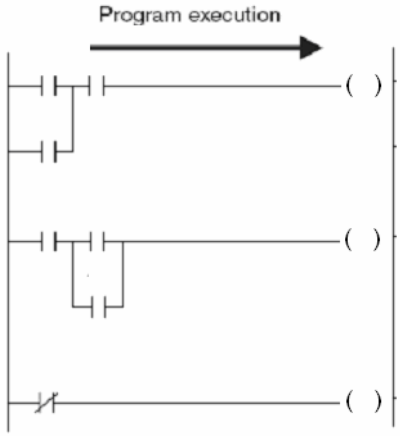
Software Structure of MELSOFT

Process image processing: The program in the PLC is not executed directly on the inputs and outputs, but on a process image of the inputs and outputs.



Input process image**:** At the beginning of each program cycle, the system polls the signal state of the inputs and stores them in a buffer creating a "process image" of the inputs.

Program execution**:** After this the program is executed during which the PLC accesses the stored states of the inputs in the process image. This means that any subsequent changes in the input states will not be registered until the next program cycle. The program is executed from top to bottom in the order in which the instructions were programmed. Results of individual programming steps are stored and can be used during the concurrent program cycle.

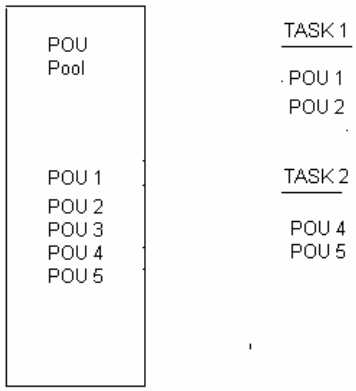


Output process image: Results of logical operations that are relevant for the outputs are stored in an output buffer – the output process image. The output process image is stored in the output buffer until the buffer is rewritten. After the values have been transmitted to the outputs the program cycle is rewritten.

GX Developer Organization Blocks Compared to Logix Tasks

The actual PLC program code is stored in the body of each PoU.

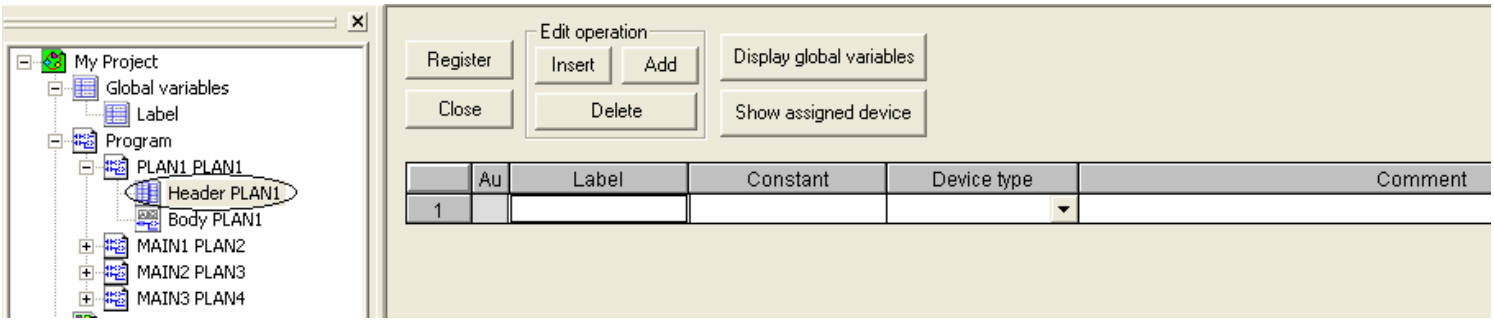
In IEC 61131-3, a program is divided into individual program modules called Program Organization Units (PoUs). A PoU is the smallest independent element of a sequence program.



PoUs are stored in pool. Program PoUs are grouped together in tasks. Individual tasks are grouped together to form the actual PLC program. Every PoU consists of a header and a body.

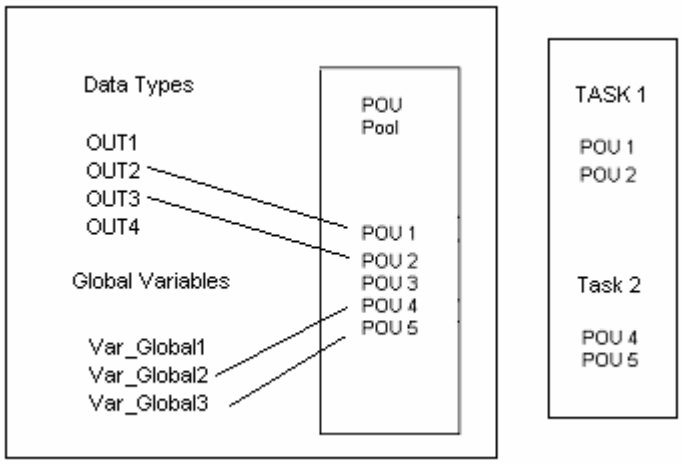
The header contains the variables that are available, declared, and used within this PoU.

The body is part of the project where the program is edited. It contains the executable PLC program code. Several languages are available for editing a program.



Tasks in GX Developer

Program Structure of GX Developer



The task pool is the “control center” for the execution of the programs. There should be at least one task for the project in the task pool. In turn, each task contains one or more PRG type PoUs. Each PoU can be defined only once in any one task in the task pool. The tasks in the task pool are polled cyclically.

The execution of the tasks is controlled by the following three parameters:

- Event
- Interval
- Priority

All tasks are either event-triggered or interval-triggered.

In order to address the SUB memory range with the A3 CPUs, a special task called MELSEC_SUB (prefix SUB) should be used**.**

Event Triggered Tasks

Events can have the following Boolean values – TRUE or FALSE. Tasks with the event value TRUE are always executed. The TRUE or FALSE parameter is automatically converted to uppercase letters.

Interrupt-triggered execution control is possible only with input modules that support this function. The task will be executed when the interrupt polled is set. The interrupt number, for example I20, is entered after Event in the task attributes in the Task Information dialog box.

CPU Series	Interrupt Range
A	10-131
FX	1000-1899
QnA	10-131
Q	10-1255

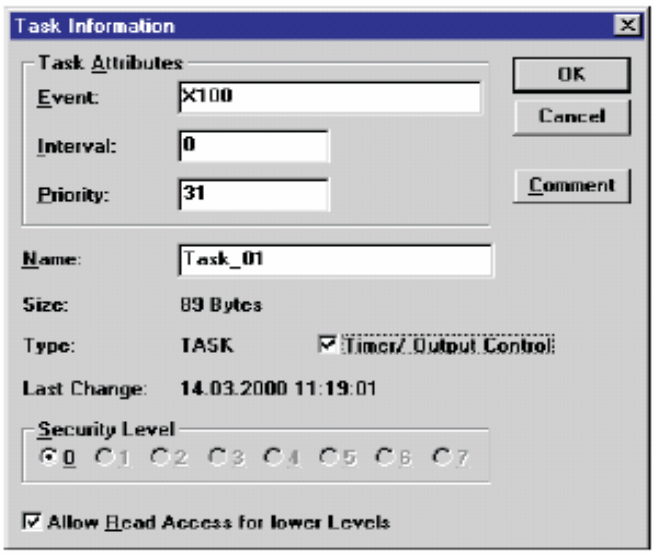
The EI instruction (Enable Interrupt) must be programmed in a MELSEC network, or the function EI_M must be used within the project when using interrupt-triggered tasks.

- Relays: The task is executed when the corresponding relay is set.
- I/O: The task is executed if the status conditions of the corresponding input or output are satisfied.
- Global Variables: All global variables of data type BOOL could be used to control a task. Elements of arrays or components of data unit variables could not be used.

Event Triggered Tasks with Timer/Output Control (MC-MCR) Execution

If a Boolean device is entered as an event, or the time interval, the timer/output control option can be selected.

Task Information



Task Processing Options

	Mode	Task Event	Execution Sequence	Event Active	Event Inactive
Timer/Output Control (Not Activated)	Normal execution	X10	LDI X10 CJ End of Task POU Code End of Task	If X10 is set the POU associated with the task is executed. If X10 is not set the program jumps to the end of the task.	The program jumps to the end of the task but some of the device status is preserved: Timers run (A, FX) or with Timer schedule for Q, QNa Timers stop running for Q, QnA without Timer schedule Ret/Accum Timers and Counters have status preserved Devices in OUT instructions have status preserved
Timer/Output Control (Activated)	MC-MCR execution	X100	LDI X100 ANI MC Relay CJ End of Task LD X100 MC N0 MC Relay POU Code MCR N0 End of Task	If X100 is set execution of the program sequence between the MC and MCR begins.	If X100 is not set the program sequence between the MC and MCR is executed once with the MC relay set to false. If the MC relay is not set, the program jumps to the end of the task and the device status is reset. Timer count value is set to 0 and contacts remain off. Ret/Accum timers and counters count value and input contact status is preserved. Output contacts are reset. OUT instruction outputs are reset. SET, RST, SFT instruction status is preserved

Interval Triggered Tasks

The intervals are entered as time constants. If a task is to be executed at intervals a user must enter FALSE for the Event parameter in the Task Information dialog box.

Syntax:	`T#[Time]`
[Time]	This is the value for the interval duration. Example: 12h30m, 5s, 100ms

The global variable of data type TIME can be used as an interval of a task. Elements of arrays or components of data unit variables cannot be used.

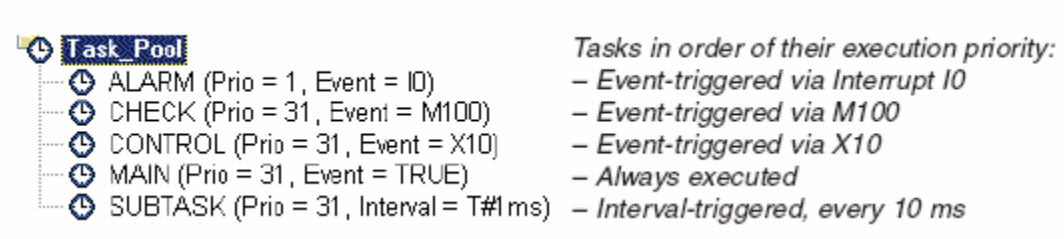
Priority-controlled Tasks

The task with the lowest priority **value** has the highest priority **execution** and it will call its associated programs first. Priority values between 0 and 31 can be assigned.

To define Task execution attributes follow these steps:

1. In the Project Navigation window, click the name of the task to edit.
2. Press the key combination Alt Enter to open the Task Information dialog box.

3. Enter the values for the task attributes.
4. Confirm with OK.



Special Task MELSEC_FIRST

If a task is named MELSEC_FIRST, it is handled with its assigned PoUs in a special way. The assigned PoUs will be the first ones in the Main sequence program. All internal code parts from the code generator that must be positioned at the beginning of the MAIN sequence program will be positioned behind this task.

The following error checks are performed during the code generation and will cause an error on occurrence:

- Interrupt events are not possible for this task.
- Because SFC PoUs must be initialized in the first scan before they are used, they could not be assigned to this task.
- The SFC control functions SFC_CTRL and SFC_PAUSE use a semaphore that is reset at the beginning of the Main sequence (now after MELSEC_FIRST). Therefore, this function cannot be used directly by programs assigned to MELSEC_FIRST or within their call tree.
- The priority task is ignored for MELSEC_FIRST.

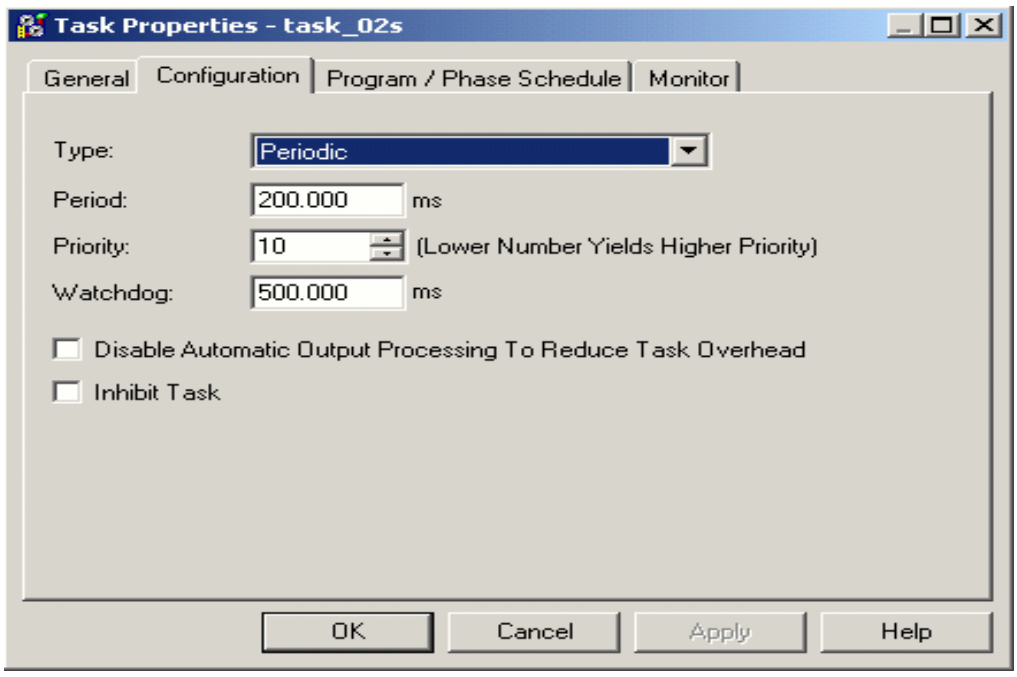
Tasks in Logix

Tasks are called by the operating system. A task provides scheduling and priority for one or more programs. Each program contains a data section and one or more code routines. The tasks may be periodic, event-based, or continuous. Each task may be assigned a priority. The continuous task, if present, is always of the lowest priority.

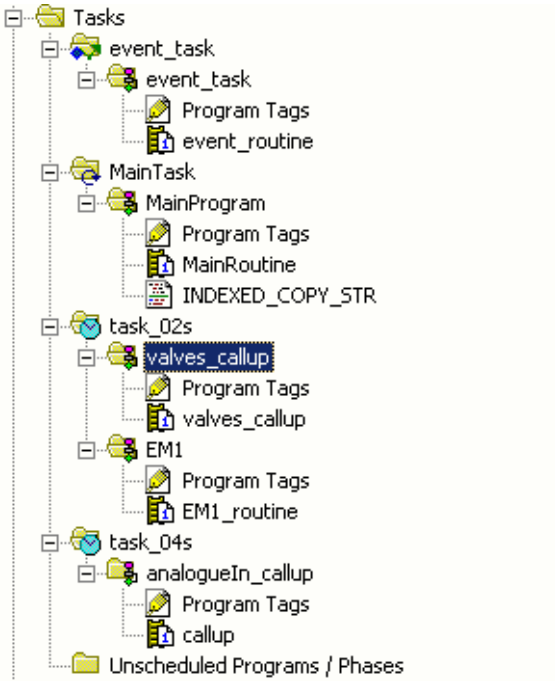
A Logix project will have one task whose default name is Main Task. This task can be continuous, event-based, or periodic. The name can be changed.

Task and Program Structure in Logix

This example from the RSLogix 5000 project tree helps illustrate how tasks and programs are configured.



The Tasks folder contained in the configuration tree below shows examples of event, periodic and continuous tasks.



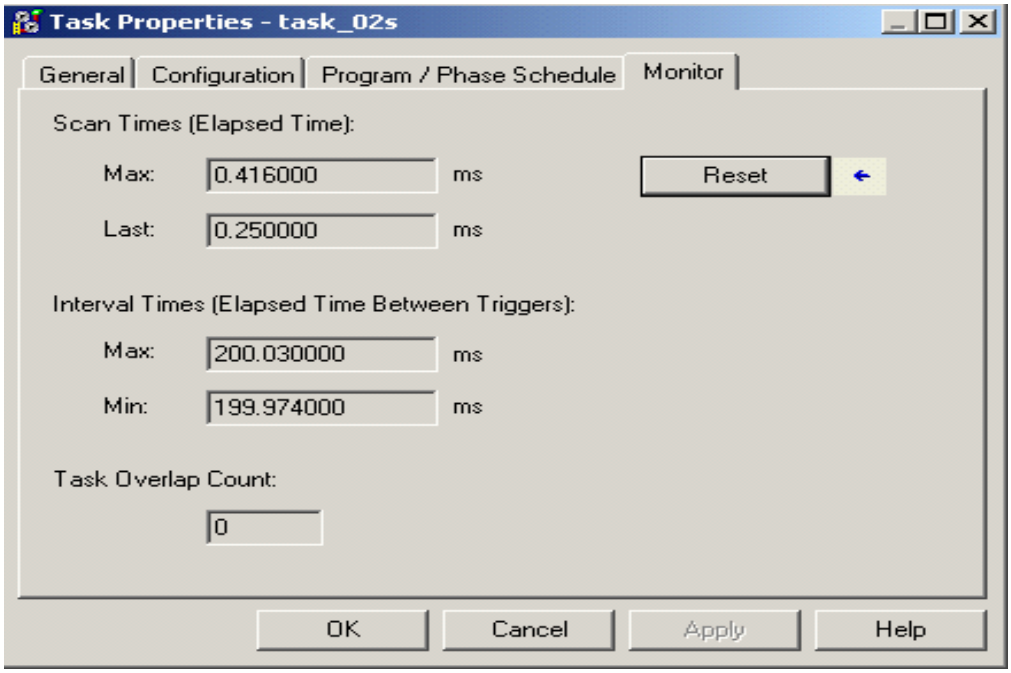
Scheduling of Periodic Tasks

Periodic tasks will trigger at a constant configured interval.

The purpose of the Task system is to:

- Allow the programmer to choose appropriate frequencies for the execution of programs. By executing code no more frequently than is needed, the controller CPU power is used more efficiently for application priorities.
- Use the priority system to allow critical tasks to interrupt lower priority ones. Therefore giving the programmer a better chance of executing at the intended frequency.

It is easy to check these times from the Monitor tab of the Task Properties.



If a trigger occurs while a task is running:

- If the new trigger is for a task with a higher priority than the one running, the running task will be interrupted by the new one, and will resume when the higher priority task is complete.
- If the new trigger is for a task with a lower priority than the one running, the running task will continue, and the new task will wait until no task of a higher priority is running.
- If the new trigger is for a task with a same priority as the one running, the controller will run both tasks by switching between them at 1 ms intervals.
- If the new trigger is for the same task as the one that is running, the new trigger will be discarded. This is an **overlap** condition. The number of overlaps that occurred since the counter was reset is shown in the task properties dialog box. A non-zero number indicates that the interrupt period needs to be increased.

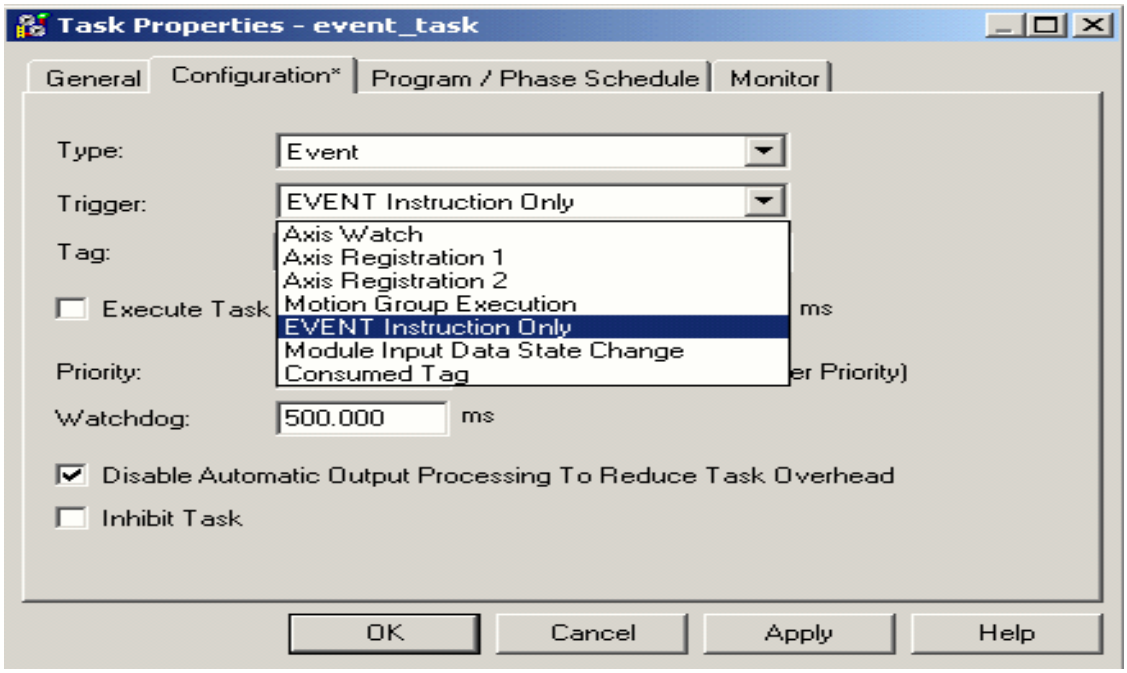
When periodic interrupts are programmed in Logix, note these similarities and differences with GX Developer:

- In GX Developer, calls will be made from the task pool which is the control center for the execution of the programs. At least one task must be defined for the project in the task pool. In turn, each task contains one or more PRG type PoU. Each PoU can be defined once in any one task in the task pool. The tasks in the task pool are polled cyclically. In Logix, insert programs and routines in the project tree under the task.
- In both GX Developer and Logix the actual application code will not differ greatly from the code in a continuous execution task. Note, the constant and known frequency of a periodic task gives programmers the opportunity to turn a simple variable increment into a timer.
- In both systems, checks for overlaps are needed as code is developed and tested. The execution time of the program cycle period or task must be much less than its execution period.
- Checking the execution time for Logix tasks is easy. Use the task properties dialog box. In GX Developer calculate the cycle period of the task by using the instruction execution times listed in the controller manual.
- In a MELSEC controller defining intervals shorter than the program cycle period can lead to unpredictable results. Logix is less strict and merely counts the number of overlaps.

Event Tasks

Event tasks will execute when a configured trigger event occurs. Normally they would be given higher priority than periodic tasks.

An event task is configured by opening the Task Properties dialog box, and choosing Event from the Type pull-down menu. Different types of event task triggers can be used for different Logix controllers.



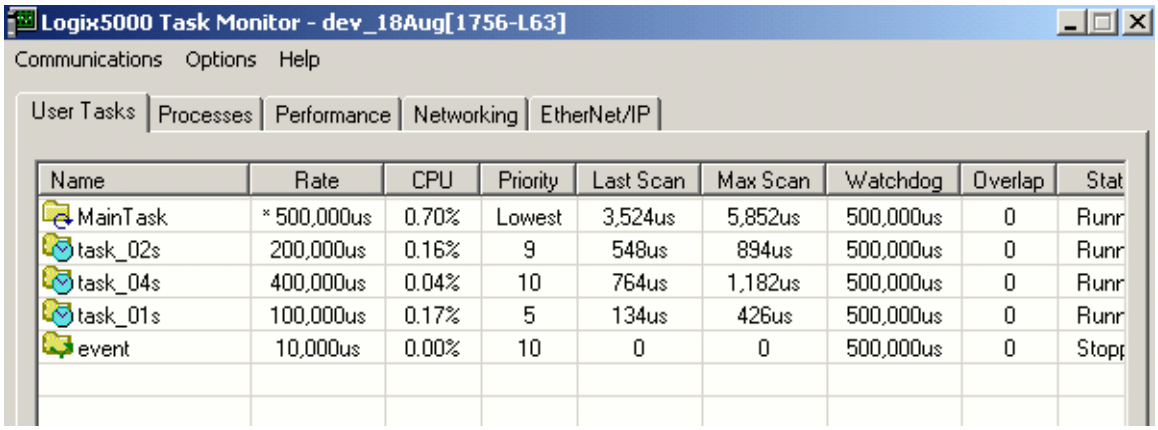
Continuous Task

A Logix controller supports one continuous task, but a project does not have to include the continuous task. The entire program can be run under Periodic and Event-based tasks. A user can configure whether the continuous task updates outputs at the end of its execution. The percentage of CPU time that is spent on unscheduled communication as a percentage of time dedicated to the continuous task can also be adjusted.

Task Monitor

RSLogix 5000 software includes a tool called Task Monitor that can help with analyzing scheduled tasks and much more.

The window below shows how information in the controller's tasks can be viewed in one table.



The other tabs provide a wealth of system-level information on the controller's performance.

Logix Tags vs GX Developer Addresses

One of the first major differences that a GX Developer user will notice when starting to work with Logix is that data does not have addresses. Data items are created in a tag database, and RSLogix 5000 software allocates addresses "behind the scenes". This makes it unnecessary for users to understand and manage memory addresses. This section describes data allocation in the two systems.

In GX Developer, addressing between modules most often must be done with "buffer memory" in the special function cards. Those who are not familiar with GX Developer find it impossible to program without a manual describing the buffer memory values for the individual module. You must pass values with a MOV statement between the module and the processor. For instance, if a user has a motion card and wants to change a position, they must first know the module location in the rack and the buffer memory location of the given position. They would move the value into the position with a MOV command. None of the data locations are addressed with anything other than position of the memory (a number, no description).

Much of Mitsubishi still uses hexadecimal addressing for physical addresses. For instance, if all input cards are allocated 32 bits of data, the first input of the sixth card would be addressed XA0. FX processors use octal numbering for addresses.

GX Developer Data Areas

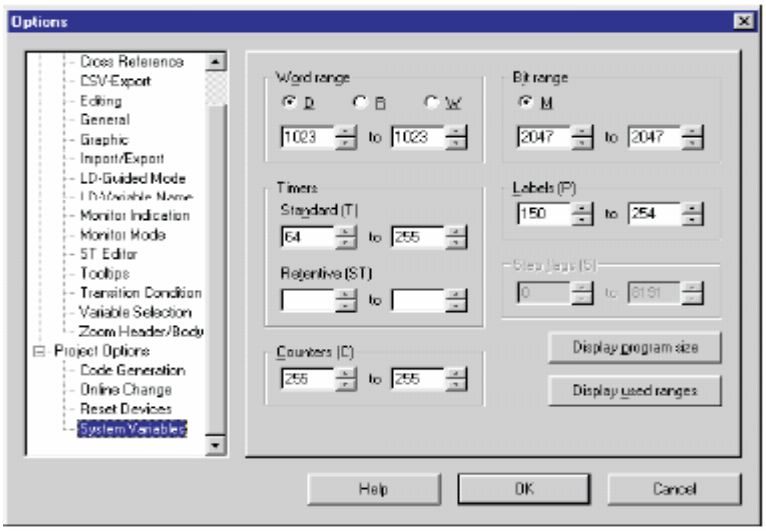
Address declaration for GX Developer is as follows:

X	**Bit Inputs**
Y	**Bit Outputs**
M	**Bit Internal Relays**
L	**Bit Latch Relays**
S	**Bit Step Relays**
B	**Bit Link Relays**
F	**Bit Annunciators**
T	**Bit Timer Contacts**
T	**Bit Timer Coils**
C	**Bit Counter Contacts**
C	**Bit Counter Coils**
T	**Timer Values**
C	**Counter Values**
D	**Data Registers**
W	**Link Registers**
R	**File Registers**

In GX Developer the hardware configuration tool will assign addresses to an I/O card when it is added to the system. For example, a digital input card might be assigned address bytes. Then the programmer will identify the bit address for each input and enter a name against it in the symbol table

System Variables

System variables vary for the actual project.



Word Range

D, R, and W devices are used as word system variables.

The word range is from/to the PLC type dependent, as defined in the parameters.

Timers

Standard (T) is from/to PLC type dependent, as defined in the parameters.

Retentive (ST) is from/to PLC type dependent, as defined in the parameters.

Counters

Counters are from/to PLC type dependent, as defined in the parameters.

Bit Range

M devices are used as system bit variables.

The bit range is from/to PLC type dependent, as defined in the parameters.

Labels (P)

Labels are from/to PLC type dependent, as defined in the adequate CNF file.

Step Flags(S)

Step flags are from/to PLC type dependent, as defined in the adequate TYP file.

Display Program Size

A summary of the used program size is displayed on a separate program size. If the program is not compiled, the dialog box shows a '?' character instead of the program size. If SFC or SUB programs are not available for this CPU, the corresponding line will be unavailable.

Display Used Ranges

A summary of the used system-variable ranges is displayed on a separate dialog box.

Data Types

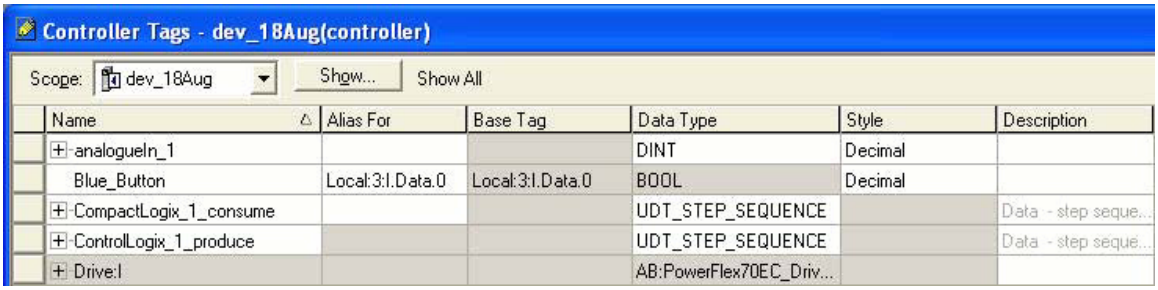
The data type determines the number and processing of bits as well as the value range of the variables.

The following data types exist.

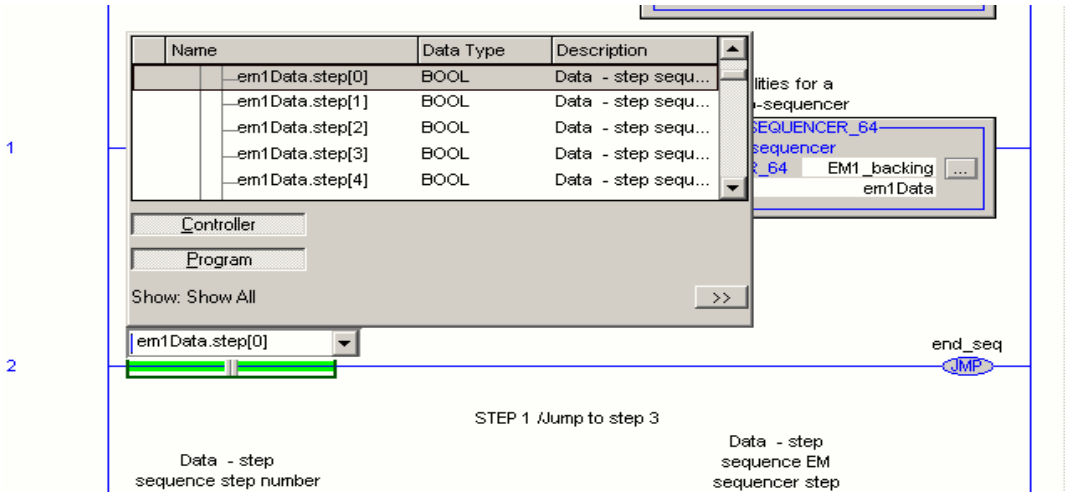
Data Type	Series CPU	Range	No. of Bits
BOOL	A, Q, System Q	0 (false), 1 (true)	1
INT	A, Q, System Q	-32768 thru +32,768	16
DINT	A, Q, System Q	-2,147,483,648 thru +2,147,483,647	32
WORD	A, Q, System Q	0 thru +65,535	16
DWORD	A, Q, System Q	0 thru +4,294,967,295	32
REAL	A, Q, System Q	3.4+/-38 (7 digits)	32
TIME	Q, System Q	T#-24d-0h31m23s648.00ms thru T#+24d-0h31m23s647.00ms	32
STRING	Q, System Q	50 characters (max.)	

Logix Data Areas

In the RSLogix 5000 programming environment, data is set up in a tag database. Memory addresses are hidden from the programmer, which makes things easier for the programmer.



Tags can be selected from the tag database using the pull-down menus while programming your logic.



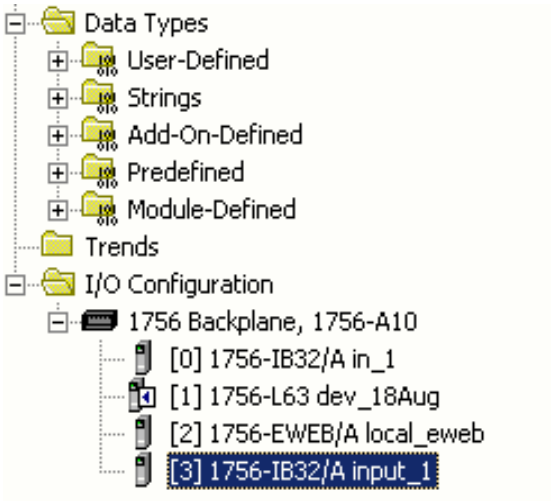
In Logix, there is a **controller-scope tag database** and **program-scope tag databases** associated with each program.

Tags in the controller-scope database are global and can be accessed by routines in any part of the program. Program-scope tags can be accessed only by routines in that program.

Logix I/O and Alias Tags

An alias tag lets you represent another tag, while both tags share the same value. One of the purposes of aliases is to reference the I/O tags as described below.

I/O modules can be added to a project by adding the module to the controller backplane in the project folder.

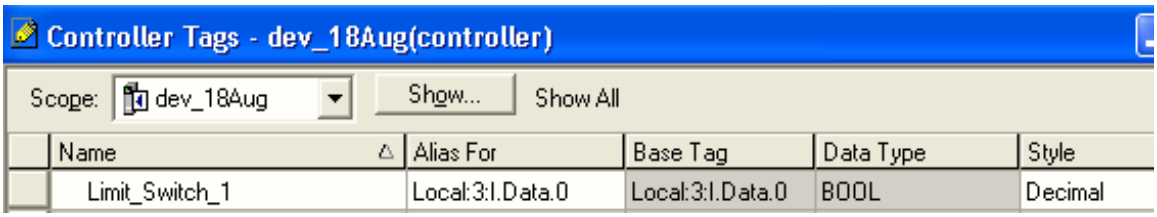


In this case, a 32-point input card has been added at slot 3. The slot number is in square brackets at the beginning of the line. “1756-IB32/A” is the catalog number of the card. The name for the card that is configured when the card is first added to the rack is “input_1”.

Having added the card, RSLogix 5000 software automatically generates the relevant device profile tags to the Controller-scope tag database. They are the **Local: 3:I** tags below.

Name	Alias For	Base Tag	Data Type	Style
Local:3:I			AB:1756_DI:I:0	
Local:3:I.Fault			DINT	Binary
Local:3:I.Data			DINT	Binary
Local:3:I.Data.0			BOOL	Decimal
Local:3:I.Data.1			BOOL	Decimal
Local:3:I.Data.2			BOOL	Decimal

Create a new alias tag with a more descriptive name. For instance, an alias for the first input can be called Limit_Switch_1, which physically describes this input.



Programming Languages

This section describes the programming languages that are available with GX Developer and RSLogix 5000 software. Selection of the Logix language most suitable to the task will result in easier program design, more rapid coding, and a program that is easier to understand.

In Logix all of the languages are “native” languages in the controller. That is, each is compiled without reference to any of the others. The benefit of this is that when a program is uploaded from the controller it can be viewed in the language in which it was written.

GX Developer has the following languages:

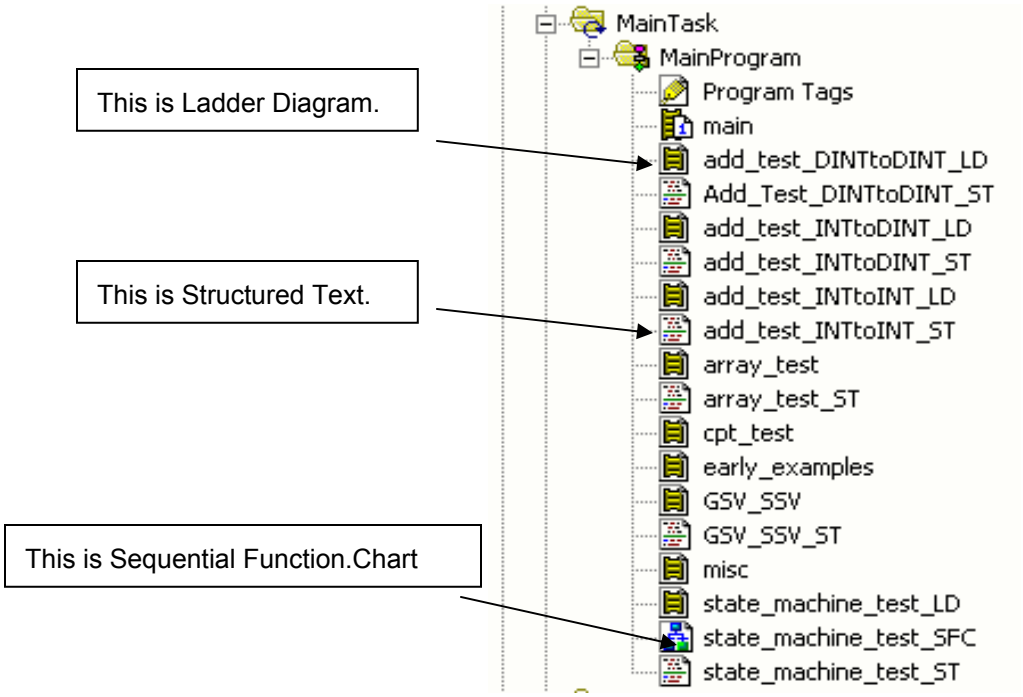
- **Instruction List** (IL): A work area is a simple text editor where the instructions are entered directly. Each instruction must contain an operator (function) and one or more operands. Each instruction must begin on a new line.
- **Structured Text** (ST): The structured text editor is compatible with the IEC61131-3. All the requirements are fulfilled.
- **Ladder Diagram**: A ladder diagram consists of input contacts and output coils, but also function blocks and functions.
- **Function Block Diagram** (FBD)
- **Sequential Function Chart**: It is a structured language that allows clear representation of complex processes.

A program can consist of function blocks and functions written in different languages.

Logix:
RSLogix 5000 software has four programming languages:

- **Ladder Diagram (LD)**
- **Structured Text (ST)**
- **Function Block Diagram (FBD)**
- **Sequential Function Chart (SFC)**

A routine, the basic section of code in Logix, can be in any of these languages. A program can be made of routines written in different languages. The following windows provide an example.



Logix Ladder Diagram

Traditionally, Ladder Diagram is used for implementing Boolean combinational logic. In Logix it can also be used for sequential logic, motion data manipulation, and mathematical calculations although other languages may be more convenient for these tasks.

Logix Structured Text

Structured Text is a high-level procedural language that will be easy to learn for anyone with experience in Basic, Pascal, or one of the 'C' families of languages. It is used primarily for data manipulation and mathematical calculations, although motion, combinational, and sequential logic can also be easily programmed in ST.

Logix Function Block Diagram

Function Block Diagram graphically describes a function (Boolean or mathematical) relating input variables and output variables. Input and output variables are connected to blocks by connection lines. An output of a block may also be connected to an input of another block.

It is good practice to program PID loops in FBD. It is the most convenient language for process control.

Logix Sequential Function Chart

SFC is a graphical tool for describing sequential logic as a set of states and transitions. Outputs may be assigned to a state and Boolean conditions for transitions to other states defined.

Conversion of GX Developer Code to Logix

In order to convert GX Developer ladder logic code to Logix, LD should be the first choice. The meaning of LD is similar in both systems.

To convert a MELSOFT function block diagram to Logix, FBD should be the first choice.

Note that the standard Logix FBD is more advanced than MELSOFT FBD.

To convert GX Developer IL code to Logix, the most suitable language will depend on the nature of the STL block. If the STL block contains mainly Boolean evaluations, LD would probably be the best Logix language to convert to. If the STL block contains pointers to access and manipulate data or executes mathematical calculations, structured text would probably be the best Logix language to convert to. If the STL block contains sequential logic, sequential function chart should be considered, although sequential logic can also be easily implemented in structured text and ladder diagram.

GX Developer Programming Arrays

An array is a collection of variables of the same data type. GX IEC Developer supports arrays with up to a maximum of three dimensions.

Declaration

Arrays are declared in the header of the program POU. They can be declared as local or global variables.

Example ▽ This example shows the definition of a one-dimensional array called `ArrSingle` with three elements of the type INT, and a two-dimensional array called `ArrDouble` with three elements of the type INT in the first dimension and five elements of the type INT in the second dimension.

① Open the header of a POU.

② Add a new line to the table with the following entries:

P_12 [PRG] Header

	Class	Identifier	Type	Initial	Comment
0	VAR	ArrSingle	ARRAY [0..2] OF INT	3(0)	One_dimensional
1	VAR	ArrDouble	ARRAY [0..2, 0..4] OF	15(0)	Two_dimensional

Fig. 6-62: Declaring arrays in the program header

- VAR: Definition of the array as a local variable in the program organization unit.
- Arr Single/Array Double: Name of the arrays.
- ARRAY [0..2] OF INT: A one dimensional array has three elements (0…2) of the type integer INT.
- ARRAY [0..2,0..4] OF INT: A two dimensional array. The second dimension is separated by a comma between the braces. In this example, the first dimension of ArrDouble has three elements (0…2) and the second dimension has elements (0…4).
- 3(0): Number of elements is 3 with the default value 0.
- 15(0): Number of elements is 3 (3x5=15) with the default value 0.

Logix Add-On Instruction Summary

Add-On Instructions in Logix are the equivalent of PX Developer Function Blocks, with private data and advanced parameter choices. In particular, the INOUT parameter type or “pass by reference” makes it possible to efficiently pass data structures to the code.

Tip: To program in FBD (GX Developer) additional PX Developer software is needed.

Because the Add-On Instruction is so similar to the PX Developer Function Block, it is likely that the MELSOFT programmer who is converting to Logix will make use of it quite readily.

Comparison between Function Blocks and Add-On Instructions include the following:

- Both can be called as named functions from anywhere in the program.
- Both contain a private data area of static data.
- In the Add-On Instruction, local static data will do the same.

Both have three types of parameters–input (pass by value), output (pass by value), and in-out (pass by reference). The pass by reference parameter is a considerable benefit because it allows large data structures to be passed efficiently.

The Add-On Instruction automatically maintains a change history by recording a timestamp and the Windows user name at the time of the change.

With the Add-On Instruction a pre-scan routine can be configured to run when the controller goes from Program mode to Run mode, or powers up in Run mode. Under these conditions the pre-scan routine will run once and can typically be used to initialize data. In GX Developer, the PoU does the same but the pre-scan code cannot be specifically attached to a FB.

If the Add-On Instruction is called from a SFC step and the SFC is configured for Automatic Reset, a post-scan routine defined in the Add-On Instruction will execute once when the SFC exits that step. It could be used for resetting data. A PX Developer FB has no built-in equivalent (although it is easy to program).

An Add-On Instruction can have an EnableInFalse routine which will be called (if present) when the rung condition at the Add-On Instruction call is false. In this case, the input and output parameters will pass values.

Function Block Description

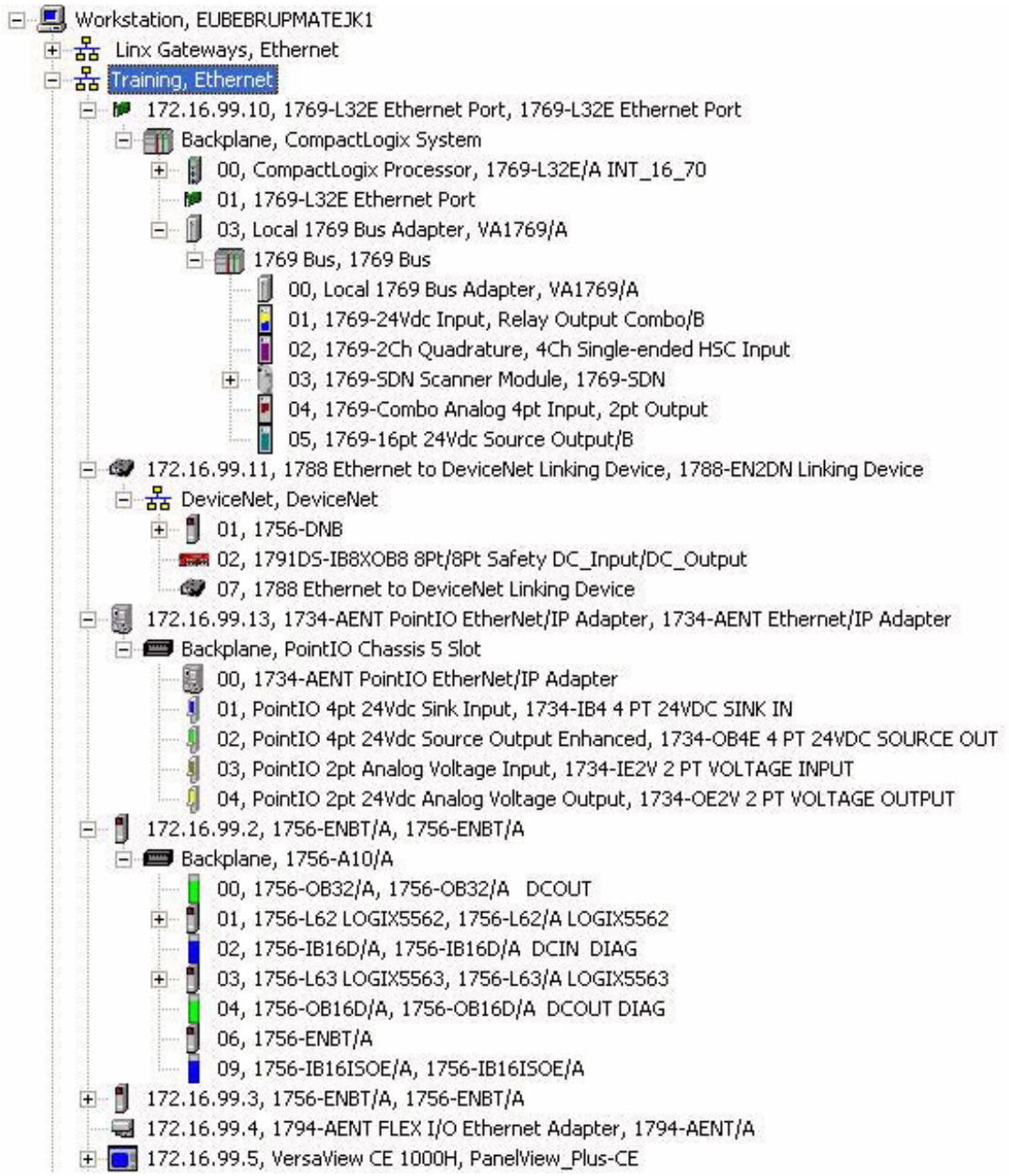
Type	Description
Input variable	Variable input from the FB outside. Up to 24 pcs. including I/O variables, at least 1 pc.
Output variable	Variable output to the FB outside. Up to 24 pcs. including I/O variables, at least 1 pc.
I/O variable	Variable having the functions of both the input and output. Up to 24 pcs.
Internal variable	Variable used only in the FB inside. {500 - (input variables + output variables + I/O variables)} pcs.

Backing Tags

Many instructions and data types use backing tags. These tags are created specifically for the instance of the instruction or data types that are being instantiated. Add-On Instructions, timers, counters, messages, and PID control all use backing tags. RSLogix 5000 software generates the corresponding structure of elements anytime a tag is created.

Viewing the Network in Logix

GX Developer users may find the Logix network configuration and management of data quite different and unique. For example, the tree below shows the devices actually connected to the system. This tree was produced by going online and nothing was preconfigured.



GX Developer Communication Protocol

GX Developer Communication Protocol is referred to as the MC protocol. The MC protocol is the name of the communication method for the Q-Series PLC. It is used by external devices to read/write device data and programs of the PLC CPU via the Q-Series C24 or Q Series E 71.

Communication through the MC protocol can be performed if the device can incorporate application programs and send/receive data by using the control procedures of the MC protocol.

The message formats and control procedures for an external device to access the PLC CPU are defined separately for the Q Series C24 and Q Series E71. The message formats and controls procedures during data communication by using the MC protocol. It is the same as when accessing the PLC CPU via conventional A/Qn Series modules.

At the PLC CPU side, the Q-Series, C24/Q Series, and E71 sends/receives data according to the commands from an external device. Thus, a sequence program for data communication is not required at the PLC CPU side.

Common Industrial Protocol (CIP)

Logix uses three main networks, EtherNet/IP, ControlNet, and DeviceNet. Each has characteristics suitable for different areas of application. The three network types share a protocol, the “Common Industrial Protocol”.

The CIP makes it possible to transfer data via any of the three types of networks supported by Logix with a nearly identical configuration and programming interface for all three. Also, data can be transferred through a network built from more than one of the three network types without any need for the programmer to translate protocols.

In “traditional” GX Developer, Ethernet interface modules are used on the PLC side to connect a host system, for example a personal computer or workstation and the system Q via the Ethernet network. MELSECNET modules and master/local modules for CC-Link are also used.

Data Exchange Between MELSEC Controllers

Data Read/Write Ranges

A QnA(S) CPU can be used with the network systems MELSECNET (II)/B/10. A CPU of the system Q supports the network systems MELSECNET/10 and MELSECNET/H.

Via the data link instructions, the CPU is able to exchange data with other stations connected to the MELSECNET and MELSECNET/10.

The data link instructions are divided into the following four categories:

- Data Refresh Instructions: These instructions refresh data in the designated network modules.
- Dedicated Data Link Instructions: These data link instructions are applied in combination with a QnA CPU or system Q CPU. For data communication, multiple channels of the network modules are used.
- A Series Compatible Link Instructions: These instructions are identical to the dedicated ACPU instructions.
- Read/Write Routing Information: These instructions read and write routing parameters from and to relay and routing stations.

For the MELSECNET and MELSECNET/10 systems, only specific data link instructions can be applied within the MELSECNET /10. Furthermore, this depends on whether the object station is a system Q CPU, an A CPU, a Qn CPU, or a remote I/O station.

Instruction Category Meaning
Network Refresh
Instructions for data refresh option in network modules
Dedicated data link
Read/write CPU data to and from object stations in object networks
Send data to network modules in object stations in object networks
Read CPU data sent via SEND instruction
Data requests to different stations (read/write operations with clock data, Run/Stop operations.)
Read/write data to and from special functions modules in remote I/O stations.
A series compatible data link
Read/write CPU data to and from object stations in different networks.
Read/write CPU data to and from local station (at master stations only.)
Read/write data to and from special function modules in remote I/O stations.
Read/Write routing
Read/write routing parameters (network number and station number of relay station, station number of routing station).

Logix Data Exchange

Produced/Consumed Tags

Produced and consumed tags are used to transfer critical data between networked Logix controllers at a defined time period. Produced and consumed tags can transmit over EtherNet/IP or ControlNet networks and on the backplane of ControlLogix controllers.

Produced and consumed tags need to be configured as produced or consumed when they are created. If a tag is marked as produced then its value will be multicast to an EtherNet/IP or ControlNet network that the controller is connected to. If it is marked as consumed then the controller that the tag requires data from will be identified as part of the configuration, and the consumed tag will receive its value from the equivalent produced tag in that controller.

There are separate channels for send and receive. Changing the value of a consuming tag will have no effect on the producing tag. No programming is required to set up produce/consume connections.

User-Defined Data Types

Data Unit Types in GX Developer

Data Unit Type (DUT) is structured, derived data types containing a collection of variables that can be of different data types. Data Unit Type can be declared as global or as local variables.

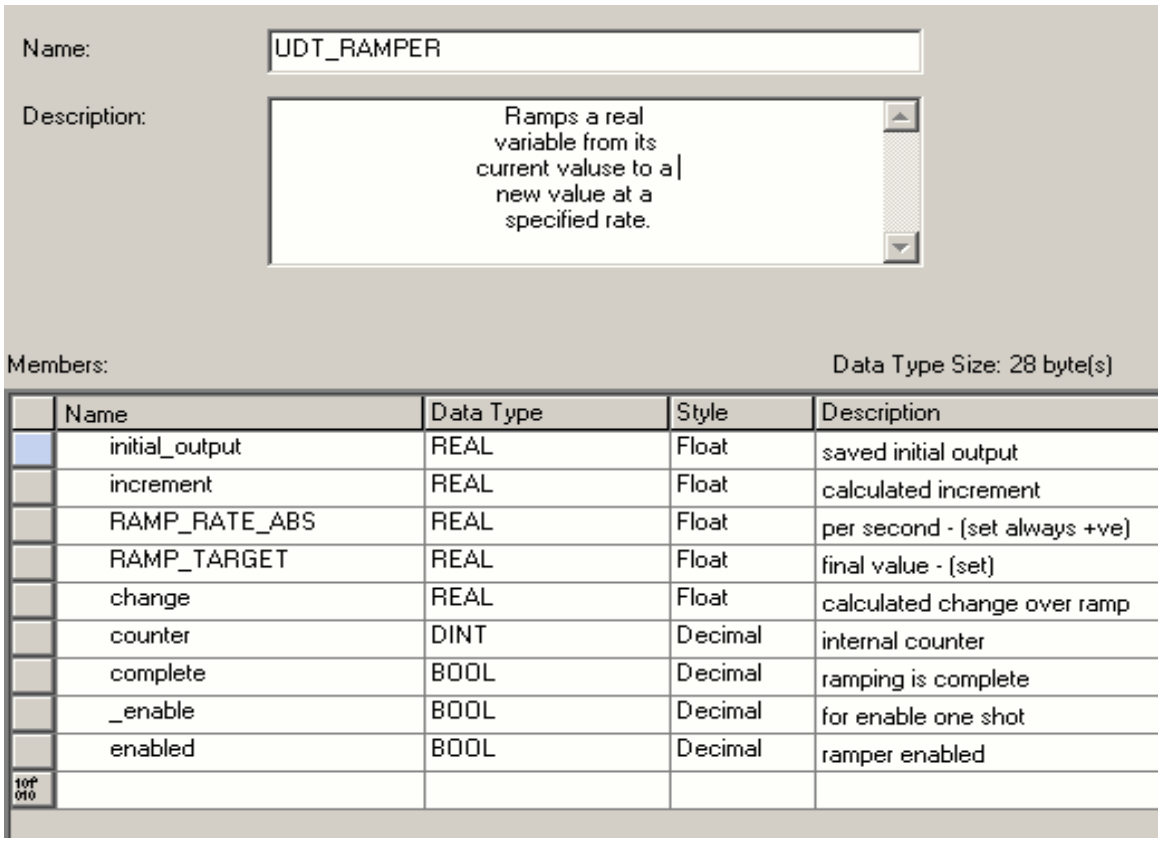
Important: Data Unit Type can be used only in the IEC editors. DUT components or arrays cannot be used in the MELSEC IL.

Data Unit Types must be defined in the DUT_Pool. All the variables in the DUT structure must be declared in a special declaration table.

Logix User Defined Data Types

In Logix, user-defined data types can be configured. This allows the structure of a complex data type to be declared as a type. Instances of that type can then be defined in the program.

Logix user-defined data types have a very similar configuration and usage to GX Developer User-Defined Data Types.



Name	Data Type	Style	Description
initial_output	REAL	Float	saved initial output
increment	REAL	Float	calculated increment
RAMP_RATE_ABS	REAL	Float	per second - (set always +ve)
RAMP_TARGET	REAL	Float	final value - (set)
change	REAL	Float	calculated change over ramp
counter	DINT	Decimal	internal counter
complete	BOOL	Decimal	ramping is complete
_enable	BOOL	Decimal	for enable one shot
enabled	BOOL	Decimal	ramper enabled

.

Logix I/O Control Method vs GX Refresh Mode

In GX Developer, the Link Refresh Instruction refreshes data at input/output interfaces or data transfer procedures.

This table gives an overview of the instructions.

	Q Series and System Q I/O Partial Refresh	A Series I/O Partial Refresh	Link and Interface data Refresh Instruction	Link and Interface Data Execution condition of refresh instruction
Instruction in MELSEC Editor	RFS RFSP	SEG	COM	EI DI
Instruction in IEC Editor	RFS_M RFSP_M	SEG_M	COM_M	EI_M DI_M

Asynchronous I/O Updating in Logix

In Logix systems, I/O is updated asynchronously with respect to program execution periods. The Logix programmer needs to consider whether there is any need to buffer input data so that its value remains constant during program execution. It is quite common to “consume” inputs once only by passing them as parameters to a code module. The inputs will not be used anywhere else in the program. This removes any need for buffering

Logix DINT Data Type

Logix controllers operate on DINT (32 bit integer) tags more efficiently than on INT (16 bit integer) or SINT (8 bit integer). Use DINT whenever possible even if the range of values you are working with would fit in an INT or SINT. These data types are provided for IEC61131-3 compatibility reasons but are internally converted to DINTs before being used by the program, so code will execute more efficiently in most situations.

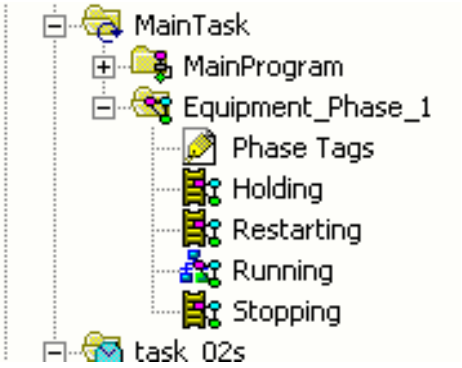
Logix PhaseManager Utility

In an S88 Equipment Phase, there are specified states of the phase as well as the transitions between these states. The PhaseManager utility is a functionality of RSLogix 5000 software that allows you to do three things:

- Allocate the code for each phase state to a different routine.
- Run a state machine 'behind the scenes' that handles the transitions between states.
- Manage the running of the phase by using a set of Logix commands.

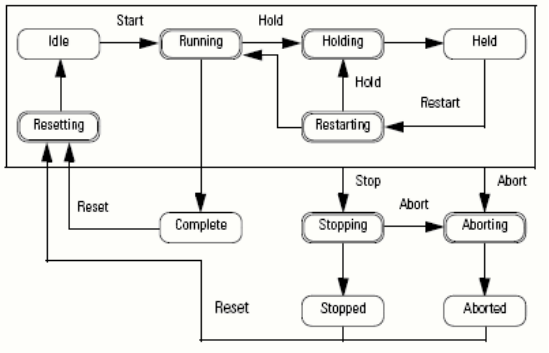
It is used in a variety of application spaces, including, but not limited to, process control and packaging, because it allows for clean separation of device/equipment control and of procedural control. This makes code far more modularized and efficient, especially for larger systems with standardization.

Equipment Phase in the Project Tree



The code for each state of the phase can be written in any of the Logix languages.

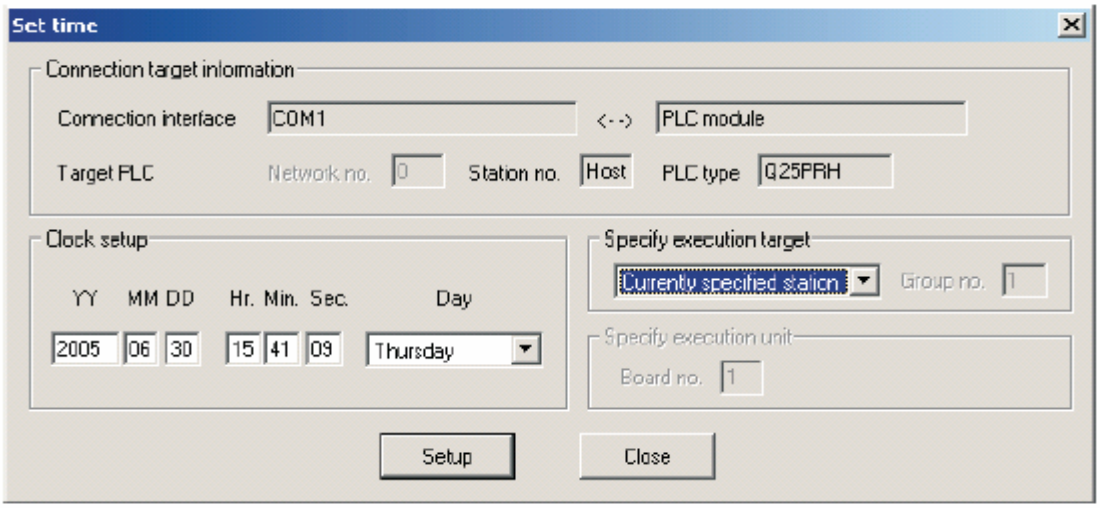
This is the phase state machine. It is almost identical to the S88 state model.



Coordinated System Time (CST)

Setting the PLC Time in GX Developer

In GX Developer, this function sets the time on the connected PLC or on all PLCs in a network.



Set time dialogue

Connection target information

Shows the actual connection.

Clock set up

Set date, time, and day.

Specify Execution Target

You can select between currently specified stations, specify all stations, and specify groups in connection with the group number as the target for setting the internal clocks.

Specify execution unit

Set the board number.

Setup

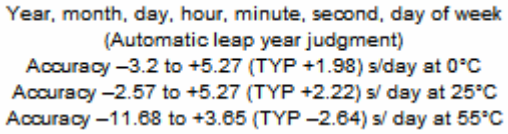
Set up the time set information.

Close

Close the dialog box.

Clock Function Details

GX Developer has a clock function that reads the year, month, day, minute, second, and day of week from the clock element of the CPU module and stores it as BIN value to the device designated by Ⓓ or later device



Logix Coordinated System Time

Logix uses coordinated system time (CST), which is a 64-bit number that measures the number of microseconds since the controller was last started. It can be measured by making calls to the operating system to get the CST value. It provides the foundation for clock synchronization for multi-CPU systems, accurate motion-control functionality, scheduled output switching to 100 microseconds accuracy, input event-time stamping, scheduled analog sampling, safety I/O monitoring and communication, motion cam-position calculations, and wall clock time.

Logix Timestamped Inputs

Timestamp is a functionality that records a change in input data with a relative time of when that change occurred. With digital input modules a user can configure a timestamp for changes of data. They can use the CST timestamp to compare the relative time between data samples.

This allows the programmer to achieve unparalleled accuracy in linking input signals to time references for applications such as those commonly used in motion control, without putting a huge burden on the communication and logic processing systems and related application code.

Logix Scheduled Outputs

With digital output modules a user can configure the modules to set the outputs at a scheduled time. This allows the programmer to achieve unparalleled accuracy in linking outputs to time references for applications such as axis positions in motion control or process control functions, without putting a huge burden on the communication and logic processing systems and related application code.

No Temporary Variables/Files in Logix

GX Developer has a category of variables called temporary variables. Their scope is the program block in which they are defined and their lifetime is the execution of the program block in which they are defined.

Logix does not have an equivalent to the temporary variable. All variables are static–they retain their values until changed. To achieve the functionality typically targeted in GX Developer applications use one of the following approaches:

- Program-scope tags
- Local Tags (part of the Add-On Instruction data) if programming an Add-On Instruction

GX Developer Standardized Programs

Programs can be standardized by using label programming or macros to create sequence programs. The creation and monitoring operations of ladders are identical to those of actual programs.

Label Programming

By using label programming to create sequence programs, a user can create standard programs with labels without being conscious of device numbers.

- The programs created by label programming can be compiled for use as an actual program.
- Label programming increases design efficiency.
- Creating a general program by label programming permits device-assignment changes according to equipment make up which can easily be diverted to other programs.
- If a user does not know equipment make up, labels can be used for programming.
- When equipment make up has been determined, relating labels and actual devices enables generation of an actual program.
- The program can be monitored/debugged without the label names being changed. This helps with efficient debugging.

Restrictions on Label Programming:

- Label programming is compatible with ladders and lists, but not with SFC and MELSAP-L.
- Device comments displayed are those set on the local/global variable setting screen.
- If the same label has been set to global and local variables, local variables have higher priority than global variables when they are displayed on the edit screen.
- To modify a program that has been written to the PLC CPU, first modify its label program stored in the personal computer and then write it to the PLC.
- Devices specified by label can be monitored via registration monitor. Devices cannot be monitored in batch.
- Restrictions when performing a write during RUN include the following:
  1. Edit within a range where global/local variables are set.
  2. Write during RUN cannot be performed if the program/parameters in the personal computer do not match those in the PLC CPU. When compiling is executed, be sure to write the compiling result to the PLC.

Macros

By naming ladder patterns using macro names and registering them to a file (macro registration), a user can enter simple instructions allowing the registered ladder patterns to be read and the devices to be changed for data diversion.

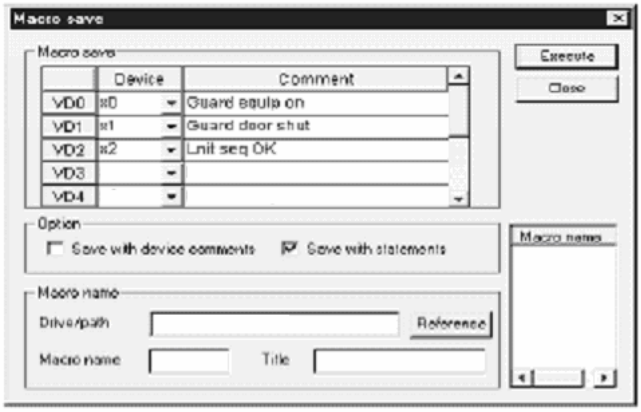
Macro Registration:

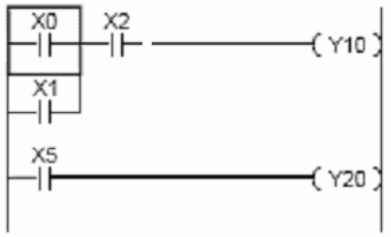
- Choose the program area to be registered as a macro.
- Set the devices to be registered.
- Set the filename of the macro to be registered.

Macro Utilization:

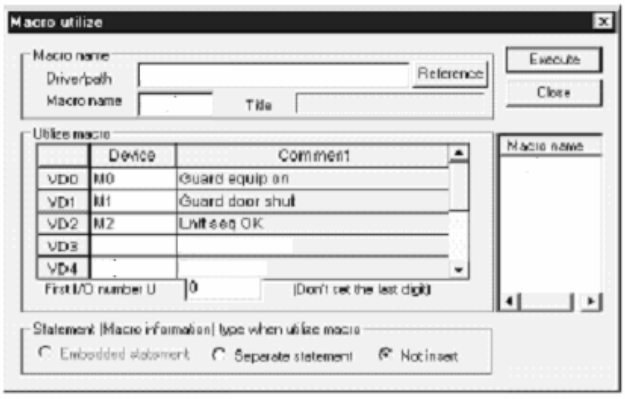
- Specify the filename registered as a macro.
- Set the devices to be changed.

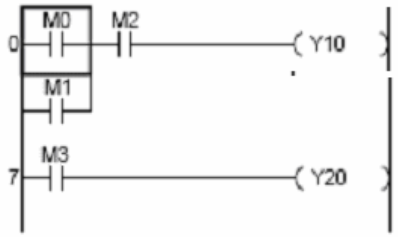
Macro Registration





Macro Utilization





Following devices may be Registered to Macros

Bit Device- digit designation
Word Device- indirect and bit designation
Extended Designation (J, U, BL)- bit, word, digit, bit and indirect designation
Designation with digit
Index qualification
Constant- integer, real, and character string
Index
Pointer
Alias

The macro registered as a ladder cannot be used in an SFC. Also, the macro registered as an SFC cannot be used in a ladder.

Chapter 3: Conversion of System Software and Standard Functions

Introduction

This chapter lists the more commonly used GX Developer system functions, explains how the equivalent is done in Logix, and provides several specific examples.

The purpose of this chapter is to make the user aware of the dedicated instructions available in Logix, so time is not wasted developing solutions that already exist.

GX Developer:

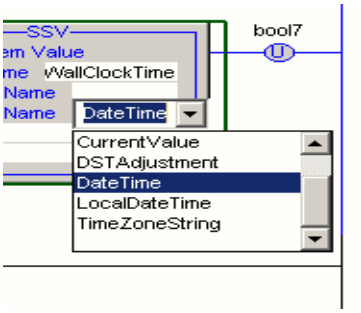
Instructions for System Clock

	Instruction
Reading	DATERD DATERDP
Writing	DATEWR DATEWRP
Adding	DATE+ DATE+P
Subtracting	DATE- DATE-P
Changing format from hh:mm:ss to seconds	SECOND SECONDP
Changing format from seconds to hh:mm:ss	HOUR HOURP

Logix System Functions

In Logix, the equivalent of most MELSOFT system functions will be the GSV (Get System Value) and the SSV (Set System Value) instructions. These instructions access a hierarchy of objects (classes, instances, and attributes) built into Logix controllers. When programming in GSV and SSV the pull-down menus guide a user through parameter selections.

SSV Instruction



Once the basics of GSV and SSV have been learned, the new Logix user may find that access to the operating system is easier than with GX Developer.

Logix Date, Time Setting and Reading

The controllers of both systems have a real-time clock which can be read or set.

Logix

SSV (Set System Value)	SSV Class – WallClockTime SSV Attribute – DateTime SSV source-specify element [0] of DINT [7].
GSV (Get System Value)	GSV Class – WallClockTime GSV Attribute – DateTime GSV dest- element [0] of DINT[7]

Read System Time

The controllers of both of these systems have a system clock which starts at the time the controller starts. In Logix it is microseconds.

Logix

GSV (Get System Value)	Returns system time in range 0…2.63 µs GSV Class – CST GSV Attribute – Current value GSV dest - specify element[0] of DINT[2] DINT[0] – lower 32 bits DINT[1] – upper 32 bits

Logix Copy

Used for copying complex data structure, such as arrays of instances of user data.

Logix Types

COP (instruction): If COP is used to copy between arrays; the start of the block (source or destination) may include an array index to address the element whose value is evaluated at run time.
CPS (instruction): Uninterruptible version – to ensure that source data cannot change during copy
CPS (ControlNet and Not required for DeviceNet Ethernet /IP)

Handling of Interrupts

Interrupts can be enabled and disabled by the user program by making calls to system functions.

GX Developer

Interrupt Program Execution Control Instructions

Interrupt Instructions
Interrupt disabled: DI
Interrupt enabled: EI
Bit pattern of execution conditions of interrupt programs :IMASK
Return from an interrupt program to the main program: IRET

Logix

SSV Inhibits specified task.	SSV Class – Task SSV Instance - Task name SSV Attribute – Inhibit Task SSV Source – DINT variable set to 1
SSV Enables specified task.	SSV Class – Task SSV Instance - Task name SSV Attribute – Inhibit Task SSV Source – DINT variable set to 0
UID	Disables interruption of the current task by a higher priority task
UIE	Enables interrupts of the current task.

Errors

The system calls return bit fields in the case of GX Developer, or an integer in the case of Logix, representing error codes.

GX Developer

If an error occurs, the self diagnostic features of the CPU return the error and store the error information in special relays (M), diagnostic relays (SM), special register (D9008), or diagnostic registers (SD).

Logix

GSV (Use SSV to reset counters or faults)	GSV Class – Fault Log GSV Attribute Major Events – No of major events Minor Events- No of Minor events Major Fault Bits – current major fault Minor Fault Bits – current minor fault GSV Target – INT or DINT to receive Data

Status–Controller

The GSV call (Logix) will return data on the controller.

Important: SFC51 requires some learning before it can be used. GSV in this case is more accessible.

Logix

GSV	Modules with a direct connection: Examine 'Fault' or 'Channel Fault' member if present. Modules with a rack optimized connection: Examine the 'SlotStatusBits' member of the adapter input data or the Output parameters are a pointer to 'Fault' member of the card as above. For a list with the returned information all other cards: Execute GSV: Class – Module Instance- Module name Attribute- Entry Status

Status–Module

The GSV call (Logix) will return data on the installed modules.

Logix

GSV	Class – Module GSV Attribute Entry Status (relationship of the Module object to the module) Fault Code Fault Info Force Status LED Status Mode (SSV also) GSV Target- depends on attribute chosen

A user can monitor fault information in the Logix tags that are created when the module is inserted into the I/O configuration.

Status–OBs and Tasks

Logix

GSV/SSV	GSV Class-Task GSV Instance-Task name GSV Attribue:Disable Update Outputs(at the end of the task) Enable Time Out Inhibit Task Instance Last Scan Time(microsecond) MaxInterval(between successive executions of task) Overlap Count(Triggered while executing) Priority Rate(period in microseconds) Start Time(value of wall clock time when last task was executed) Status(3 status Bits) Watchdog (microseconds) GSV Source/Target – depends on attribute chosen

Timers

Logix

TON (LD) TONR ST & FBD)	On-delay timer
RTO (LD) RTOR (LD & ST)	Retentive on delay timer
TOF(LD) TOFR(ST & FBD)	Off Delay Timer

Conversion Routines

GX Developer Conversion Routine Instructions

Instructions	Conversion
BCD, BCDP, DBCB, DBCDP	BIN to BCD
BIN,BINP, DBIN, DBINP	BCD to BIN
FLT, FLTP, DFLT, DFLTP	BIN to Floating Point
INT, INTP, DINT, DINP	Floating Point to BIN
DBL, DBLP	BIN16 to BIN32
WORD, WORDP	BIN32 to BIN16
GRY, GRYP, DGRY, DGRYP	BIN to Gray Code
GBIN, GBINP, DGBIN, DGBINP	Gray Code to BIN
ENEG, ENEGP	Signal Reversal for Floating Point
BKBCD, BKBCDP	BIN Block to BCD Block
BKBIN, BKBINP	BCD Block to BIN Block

Logix Conversion Instructions

Instructions	Conversion
DTOS	INT can be used as a source tag instead of DINT
DTOS	DINT to string
RTOS	Real to String
STOD	String to DINT
STOR	String to real

String Handling Routines

GX Developer
List of String Handling Instructions

BIN 32-bit Data Comparison Instructions	BIN 16-bit Data Comparison Instructions	Floating Point Data Comparison Instructions	Character String Data Comparison Instructions
LDD= ANDD= ORD=	LD= AND= OR=	LDE= ANDE= ORE=	LD$= AND$= OR$=
LDD<> ANDD<> ORD<>	LD<> AND<> OR<>	LDE<> ANDE<> ORE<>	LD$<> AND$<> OR$<>
LDD> AND> ORD>	LD> AND> OR>	LDE> ANDE> ORE>	LD$> AND$> OR$>
LDD<= AND<= ORD<=	LD<= AND<= OR<=	LDE<= ANDE<= ORE<=	LD$<= AND$<= OR$<=
LDD< AND< ORD<	LD< AND< OR<	LDE< ANDE< ORE<	LD$< AND$< OR$<
LDD>= AND>= ORD>=	LD>= AND>= OR>=	LDE>= ANDE>= ORE>=	LD$>= AND$>= OR$>=

Logix
List of String Handling Instructions

EQU	Compares strings for equality
GEQ (LD) >= (ST)	Compares String for >=
GRT(LD)	Compares String for >
LEQ(LD) <=(ST)	Compares String for <=
LES(LD) <(ST)	Compares String for <
NEQ(LD) <>(ST)	Compares String for <>
.LEN	Property of any string instance
MID	Returns a middle section of string
CONCAT	Concatenates two strings
DELETE	Deletes a section of string
INSERT	Insert source string in target string
FIND	Find a string stored in another string

There is no equivalent in GX Developer to Logix ASCII serial port instructions, neither in the instruction set nor in the function library. These would have to be programmed in STL if required by using a conversion from BIN to ASCII with the ASC(P) instruction.

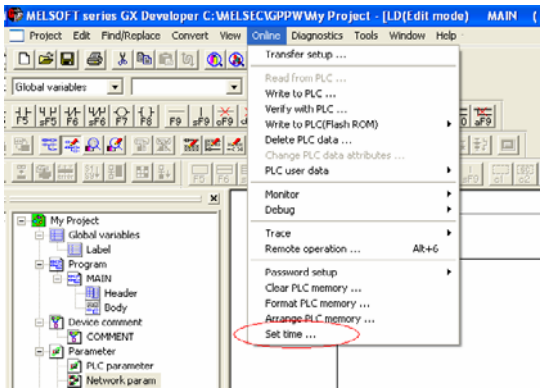
Examples of System Function Calls

These examples are intended primarily to illustrate the use of the GSV/SSV Instructions.

Setting the Clock

GX Developer

This function sets the time on the connected PLC or on all PLCs in a network.



Connection target Information

Displays the actual interface connection.

Clock Set Up

Set date, time, and day.

Specify execution target

Select currently specified stations, all stations, or group in connection, with the group number as the target for setting the internal clocks.

Specify Execution Unit

Set the board number.

Setup

Set up the time set information.

Close

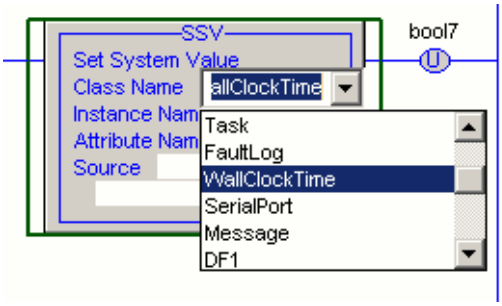
Close the dialog box.

Logix

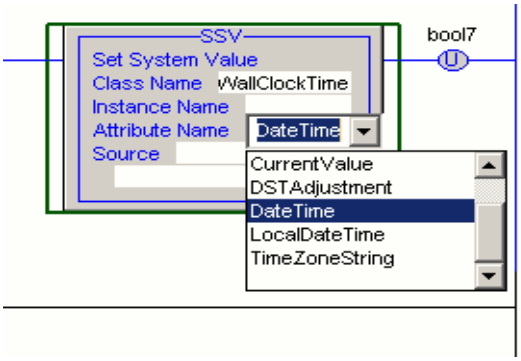
The date and time values are stored in the seven DINTs following **#date_time.**

0 - year
1 – Month
2 – Day
3 – Hour
4– Minute
5 – Second
6 - Microsecond

The Logix window shows the data structure associated with GSV and SSV. Choose the class from a pull-down menu as follows.

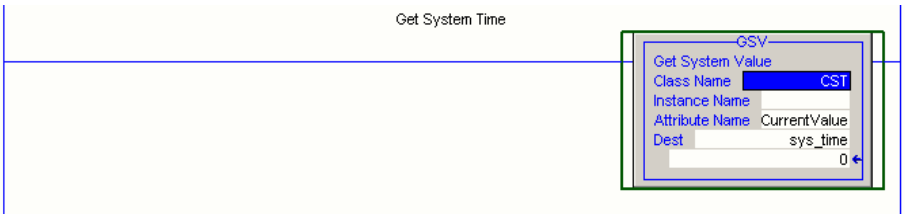


Choose the attribute from the pull-down menu as follows.



.

Finally, choose the tag that will be the source (SSV) or destination (GSV) of the data.



Disabling Interrupts

GX Developer

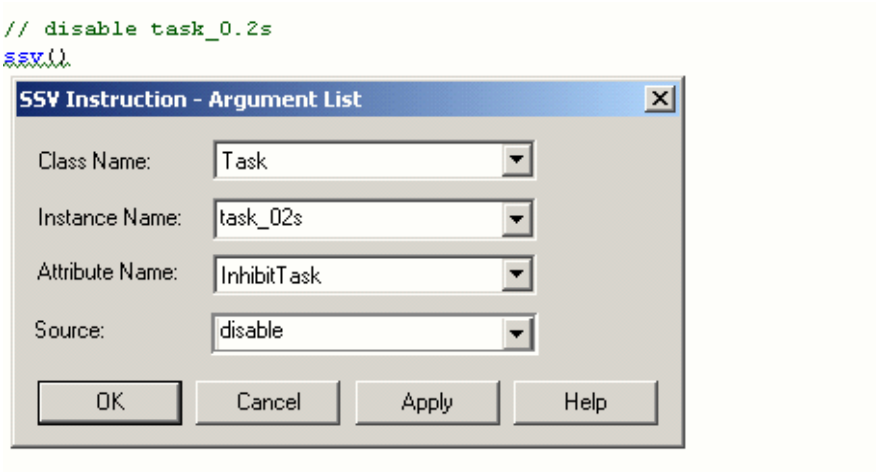
The DI instruction disables the execution of an interrupt program until an EI instruction is executed. The DI status after switching on or resetting the CPU device is active.

Logix

This example shows SSV in Structured Text.

If a user types “gsv” then “alt-A” the following dialog box appears.

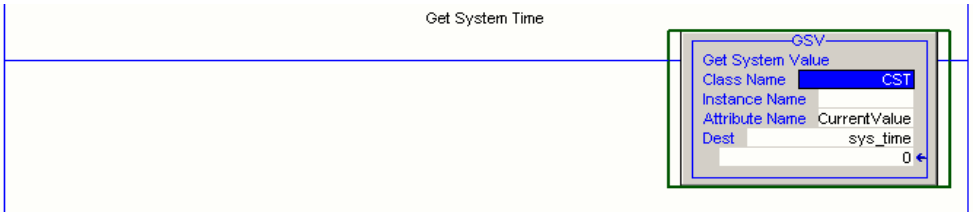
Once the parameters are entered, click OK and the actual parameters will be completed.




```
// disable task_0.2s
ssv(Task,task_02s,InhibitTask,disable);
```

Read System Time

Logix



Get Faults

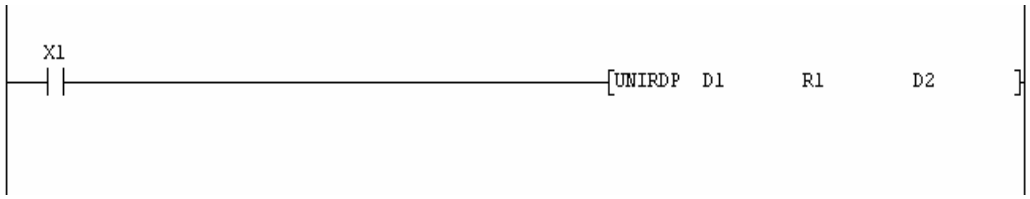
GX Developer

Debugging and failure diagnosis information: TRACE, TRACER.

Logix

Module Information

GX Developer

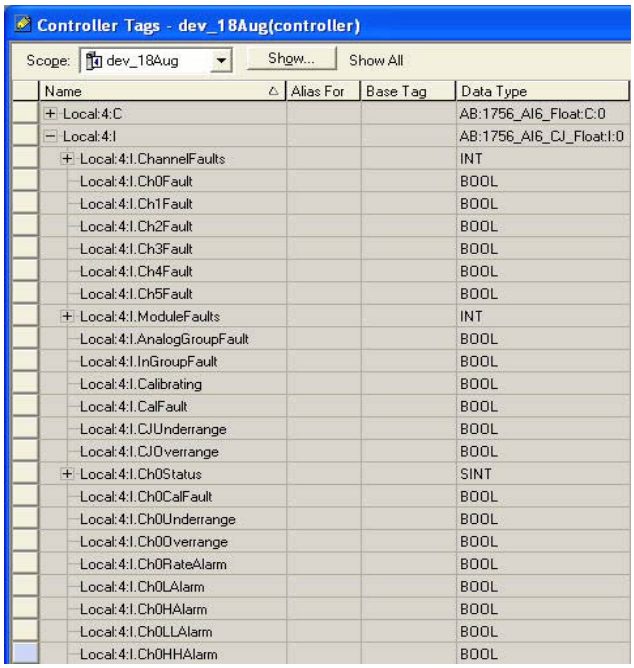


Reading module information: UNIRD, UNIRDP

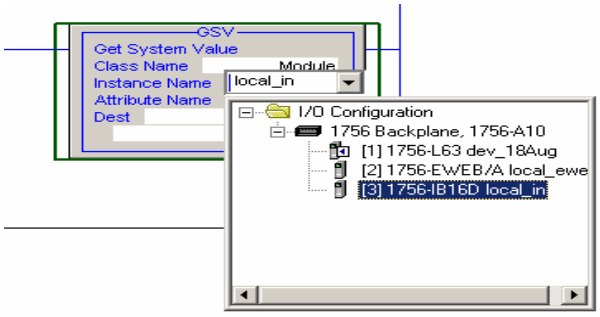
Logix

This is the easiest way to inspect the module device-profile tags which contain fault/diagnostic information.

1756-IT6I2 Thermocouple Analog-input Card Tag



Another way is to use the GSV instruction to read module objects. The window below shows how to use GSV to obtain information regarding the 1756-IB16D digital input module.



Get Scan Time

GX Developer

The Scan command in the Debug menu provides a statistical display of the scan time performance of the program in the PLC CPU.

The program-execution cycle periods can vary depending on status conditions, for example, whether inputs are polled during the cycle. The PLC CPU logs the following scan times:

- Actual Scan Time: The current program cycle time.
- Minimal Scan Time: The last minimum scan time saved.
- Maximum Scan time: The last maximum scan time saved.

Logix

The execution time can be retrieved for each Logix task.

Chapter 4: Conversion of Typical Program Structures

Introduction

The objective of this section is to demonstrate how some typical programming tasks in GX Developer can be performed in RSLogix 5000 software. The discussion is based mainly on code fragments, but there are also some complete examples.

These examples show conversion code in GX Developer. There is a limited display of rungs and network lines that must be drawn manually. Forcing of values takes several steps compared to Logix.

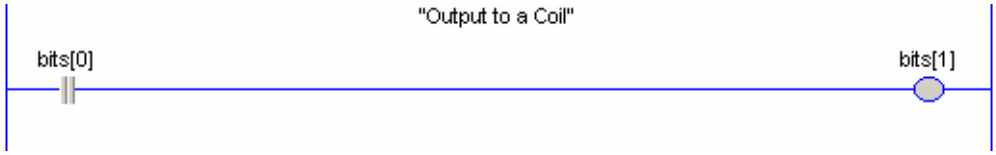
Ladder Logic Translation

This section describes a few examples of comparison between GX Developer LD and Logix LD.

Writing to a Coil

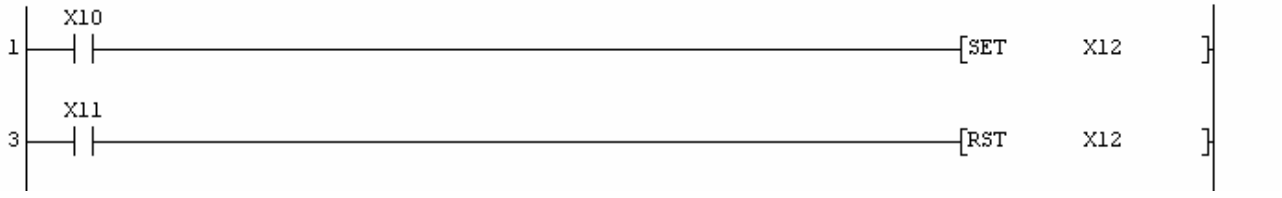
GX Developer

Logix

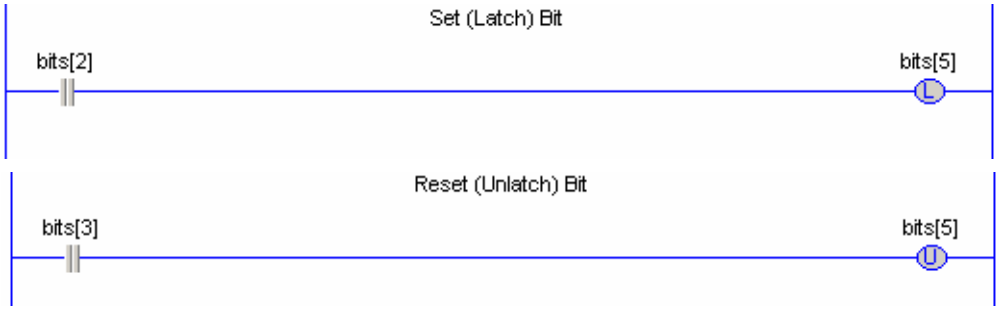


Set and Reset

GX Developer



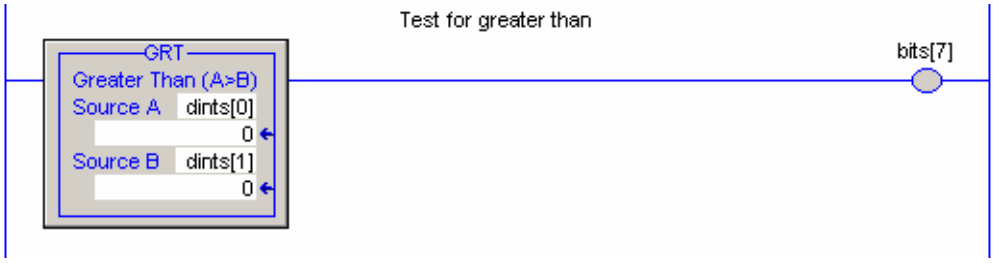
Logix



Test for Greater Than

GX Developer

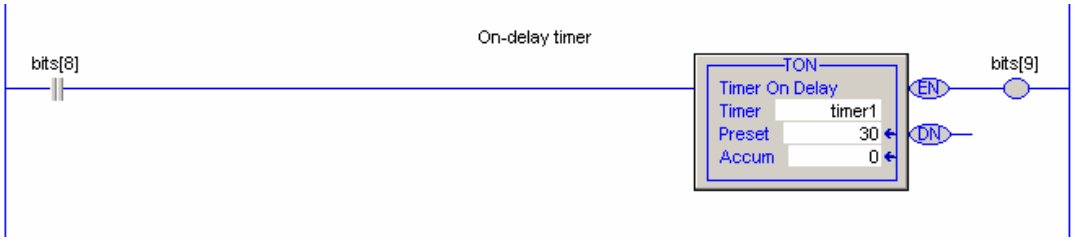
Logix



On Timer Delay

GX Developer

Logix



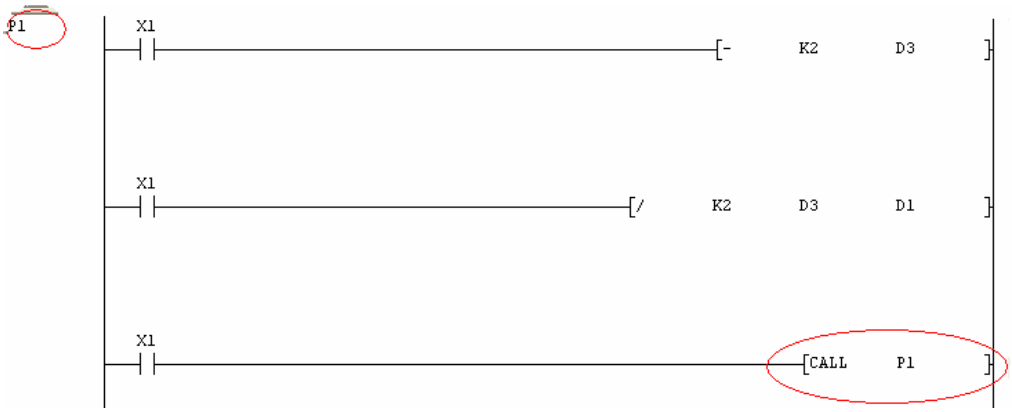
Call to User Function

GX Developer

Call to USER Function Instructions

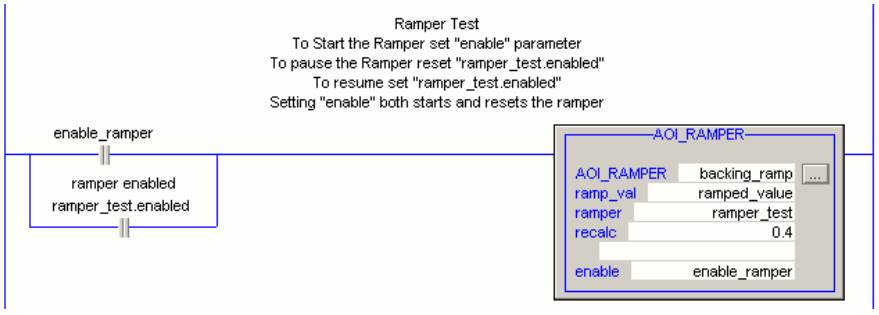
Instruction	Repetition Instructions	Subroutine Calls	Subroutine Calls between program files	Program Switching	Microcomputer Call	Index Qualification	Designation of qualification values in index qualification
MELSCEC Editor	For, Next, Break, BreakP	CALLI, CALLP, RET, FCALL, FCALLP	ECALL, ECALLP, EFCALL, EFCALLP	CHG	SUB, SUBP	IX, IXEND	IXDEV, IXSET
IEC Editor	FOR_M, NEXT_M, BREAK_MD, BREAK_P_MD	CALL_M, CALLP_M, RET_M< FCALL_MD, FCALL_P_MD	ECALL_M, ECALLP_M, EFCALL_M, EFCALLP_M	CHG_M	SUB_M, SUBP_M	IX_MD, IXEND_MD	IXDEV_M, IXSET_M

GX Developer



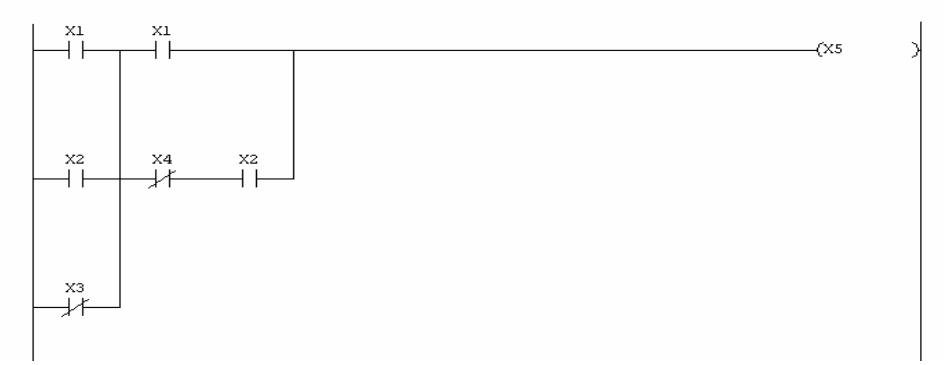
Logix

An AOI (user defined instruction) can be added simply to any routine in Logix.

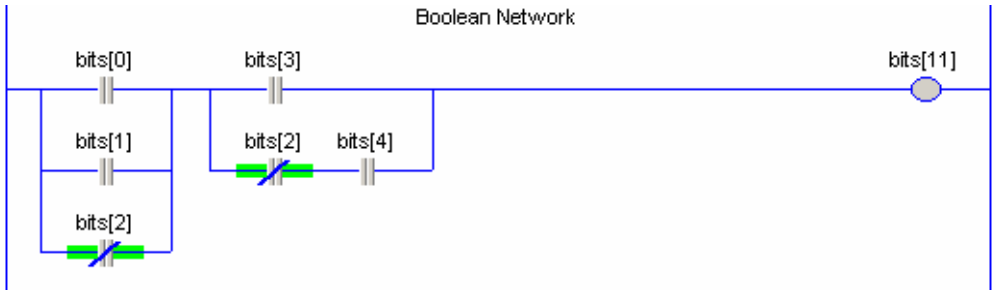


Boolean Network

GX Developer

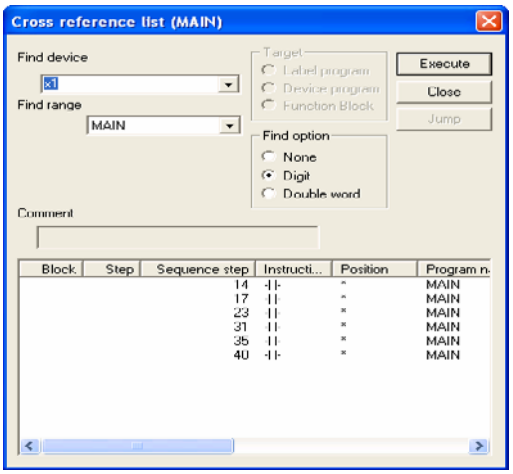


Logix

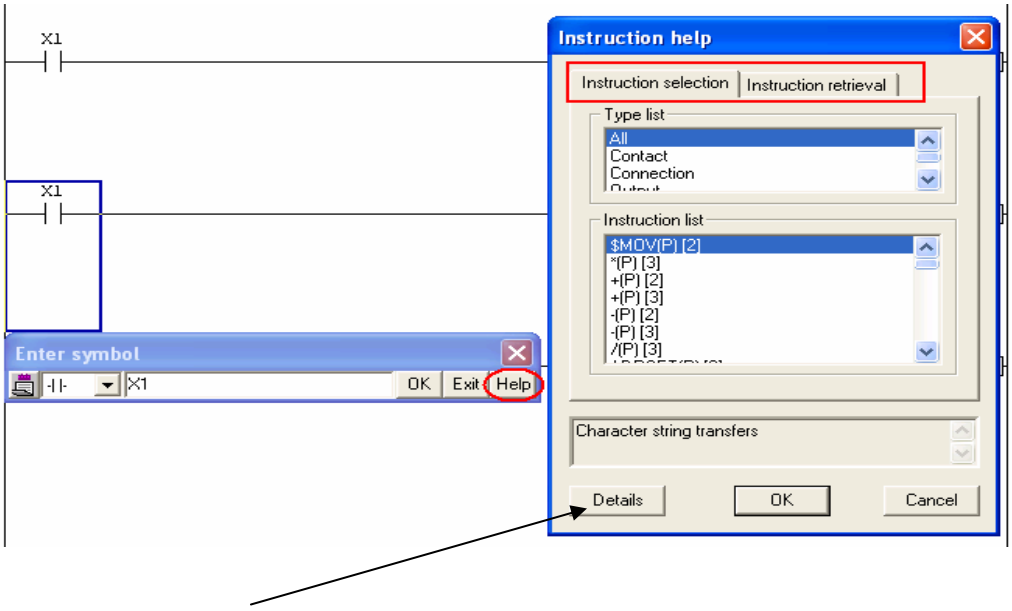


GX Developer Editor

While configuring the instructions, pull-down menus help to select the data area.



Double-click the instruction. This opens the “Instruction help” that helps in configuring the instruction.



When the Details button is clicked, the instruction help opens as shown.

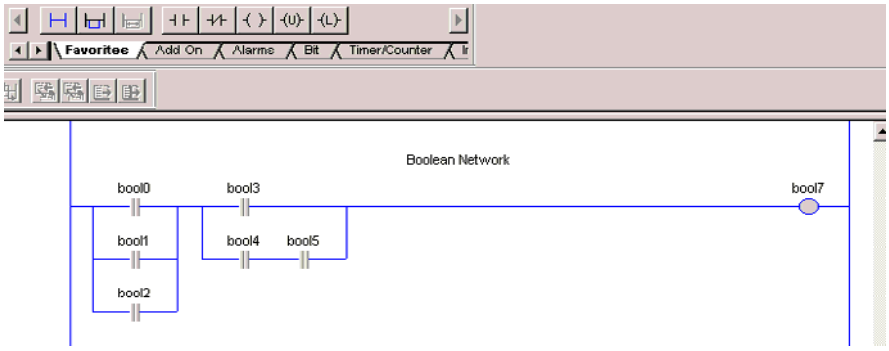
It is very challenging to create programs with GX Developer. Documentation, editing, and changing values are all much more difficult in GX Developer compared to Logix.

Commenting in GX developer is limited. It allows a maximum of 32 characters and is not automatically formatted.

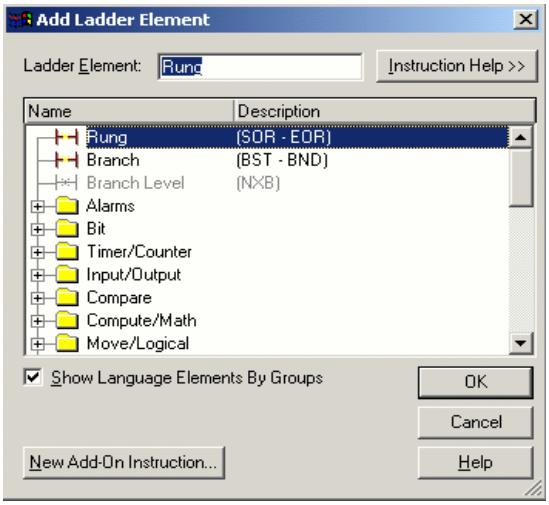
Logix LD Editor

There are seven ways to select instructions in the ladder diagram programming. Two methods that are fairly similar to the way it is done in MELSOFT are described below.

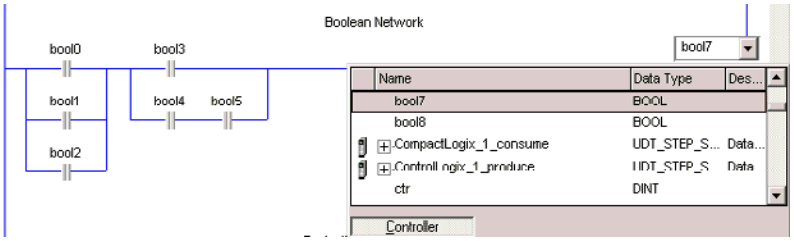
Select from a palette above the LD worksheet.



When Alt+Insert is typed, the following dialog box appears.



There are many similarities between GX Developer and Logix LD to make translation at the level of routines fairly straightforward. When configuring instructions, pull-down menus are available to allow a user to select the tag to be entered.



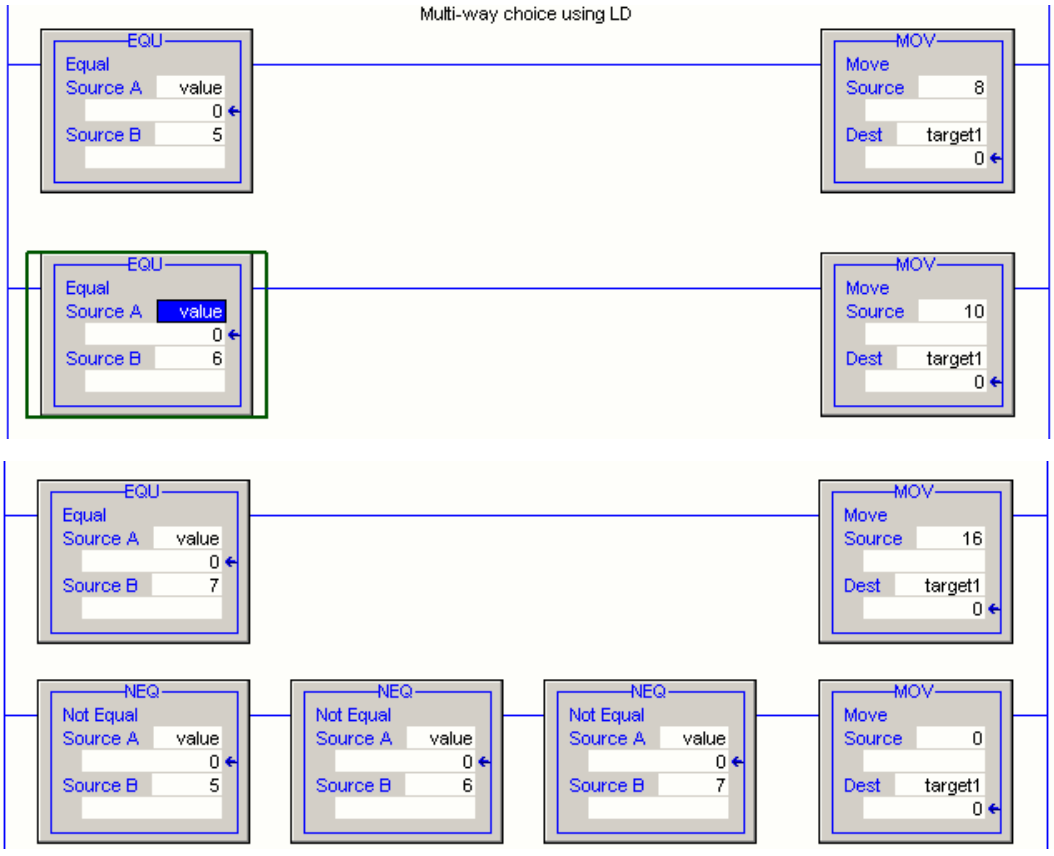
Jumps and Decision Making

GX Developer Jumps/Program Branching

IEC Editor	
JMP (Modifiers: N, C)	The jump is always executed unconditionally.
JMPC (Jump Conditional)	The jump is only executed when the value in the accumulator is 1.
JMPCN (Jump Conditional Not)	The jump is only executed when the value in the accumulator is 0.
MELSEC Editor	
JMP	Unconditional jump in program
CJ	Conditional jump in program
SCJ	Conditional jump in program, executed in the next program cycle

Logix

The following diagram shows Multi-way choice using LD.



Calling Subprograms and Function Blocks

IEC Editor	
CAL (Modifiers: N, C)	Call to a Function Block.
RET (Jump Conditional)	Return from a Function Block
MELSEC Editor	
CALL	Call a subprogram
SRET	End/return from a subprogram
RET	Terminate step ladder

GX Developer Structured Program Instructions

Structured Program Instruction calls programs, parts of programs, or switches between them. In addition, instructions for index qualification and program repetitions (loops) are supplied.

The following datatable gives an overview of structured program instructions.

Instruction	**Repetition Instructions**	**Subroutine Calls**	**Subroutine Calls between program files**	**Program Switching**	**Microcomputer Call**	**Index Qualification**	**Designation of qualification values in index qualification**
MELSCEC Editor	For, Next, Break, BreakP	CALLI, CALLP, RET, FCALL, FCALLP	ECALL, ECALLP, EFCALL, EFCALLP	CHG	SUB, SUBP	IX, IXEND	IXDEV, IXSET
IEC Editor	FOR_M, NEXT_M, BREAK_MD, BREAK_P_MD	CALL_M, CALLP_M, RET_M< FCALL_MD, FCALL_P_MD	ECALL_M, ECALLP_M, EFCALL_M, EFCALLP_M	CHG_M	SUB_M, SUBP_M	IX_MD, IXEND_MD	IXDEV_M, IXSET_M

GX Developer

Logix Structured Text CASE statement

This is another variant in ST that does the same task.

```
// multi-way choice using Structured Text CASE

case value of
    5: target := 8;
    6: target := 10;
    7: target := 16;
else target :=0;
end_case;
```

All solutions work, but this is the preferred Logix solution.

Logix–Structured Text If…Then…Else

```
// multi-way choice using Structured Text

if (value = 5) then target := 8;
elsif (value = 6) then target := 10;
elsif (value = 7) then target := 16;
else target := 0;
end_if;
```

Brackets around the ‘if’ condition are not compulsory.

.

Arrays

An array is a collection of variables of the same data type.

GX Developer

Arrays are supported only in the IEC editors. Arrays are declared in the header of the program PoU. They can be declared as local or global variables.

MELSOFT and Logix both allow arrays of simple or complex objects to be created in memory. Logix has high-level support for accessing arrays.

Array Declaration Syntax

- GX Developer: uses the declaration syntax **VAR** as a local variable in the PoU 'P_3'. Logix uses REAL [15].
- VAR_EXTERNAL: is defined as global variable and referenced in the header PoU as 'P_3'.

This example shows the definition of a one-dimensional array called ArraySingle (with four elements type INT) and a two-dimensional array called ArrayDouble (with four elements of type INT in the first dimension and five elements of type INT in the second dimension).

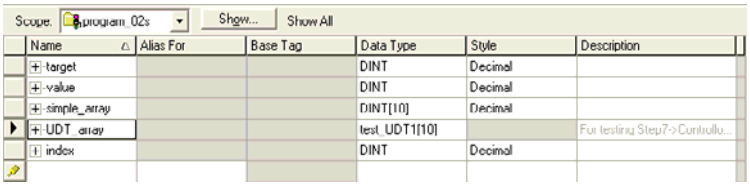
P_12 [PRG] Header					
	Class	Identifier	Type	Initial	Comment
0	VAR	ArraySingle	ARRAY[0..3] OF INT	3(0)	One_dimensional
1	VAR_EXTERNAL	ArrayDouble	ARRAY[0..3,0..4]	15(0)	Two_dimensional

GX Developer

Calling of arrays

One Dimensional arrays:	Two-dimensional arrays:
LD Actual Parameter	LD Actual Parameter
ST 'Array Name ('Element')	ST 'Array Name ('Element of 1st dimension', 'Element of 2nd dimension)

Logix-Array Creation



Logix–Array Operations in Structured Text

The following ST fragment performs the tasks described in the preceding two sections.

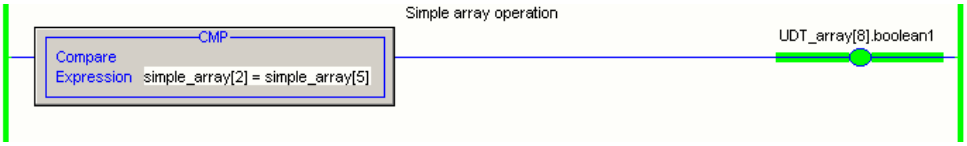
```
// array access in ST
if (simple_array[2] = simple_array[5]) then
     UDT_array[8].booleanl := 1;
else
     UDT_array[8].booleanl := 0;
end_if;

// clearing array elements
if (simple_array[0] = 5) then
     index := 0;
     while (index <= 9) do
          UDT_array[index].reall := 0.0;
          index := index + 1;
     end_while;
end_if;
```

Alternative Boolean equation:

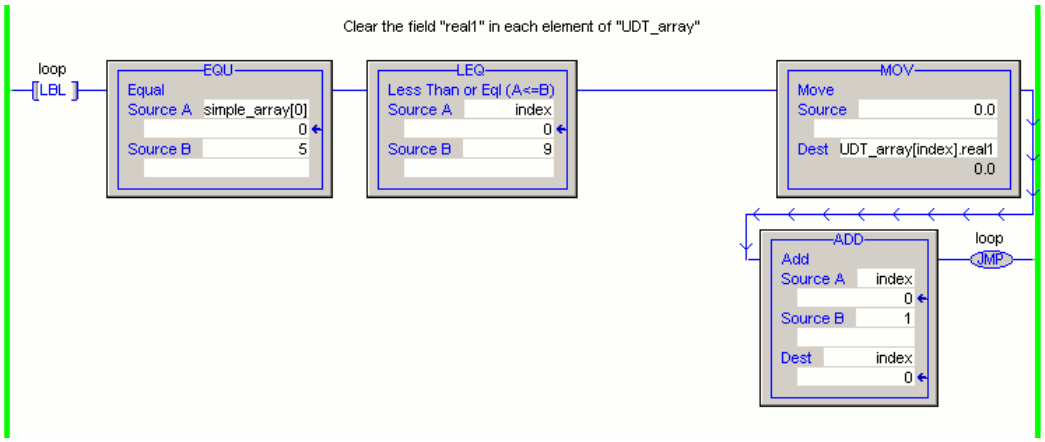
```
// array access in ST
UDT_array[8].booleanl := simple_array[2] = simple_array[5];
```

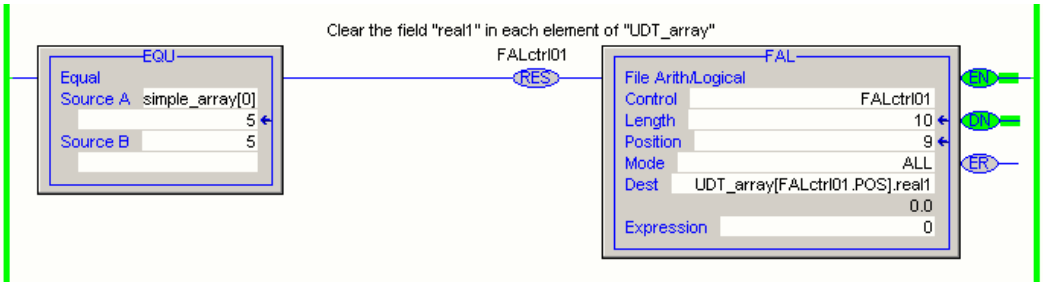
Operations in Ladder Diagram

The examples of the previous section can be written in LD by using the CMP (Compare) instruction.



Clearing the real field in the array of UDTs can be done one of the following two ways.





The first, uses the advanced FAL instruction for array operations. The second approach to clearing the array elements is a translation from Loop of the ST code.

User Data Types

Data Unit types in GX Developer

Data Unit Type (DUT) is structured, derived data types containing a collection of variables which can be of different data types. Data Unit Type can be declared as global or as local variables.

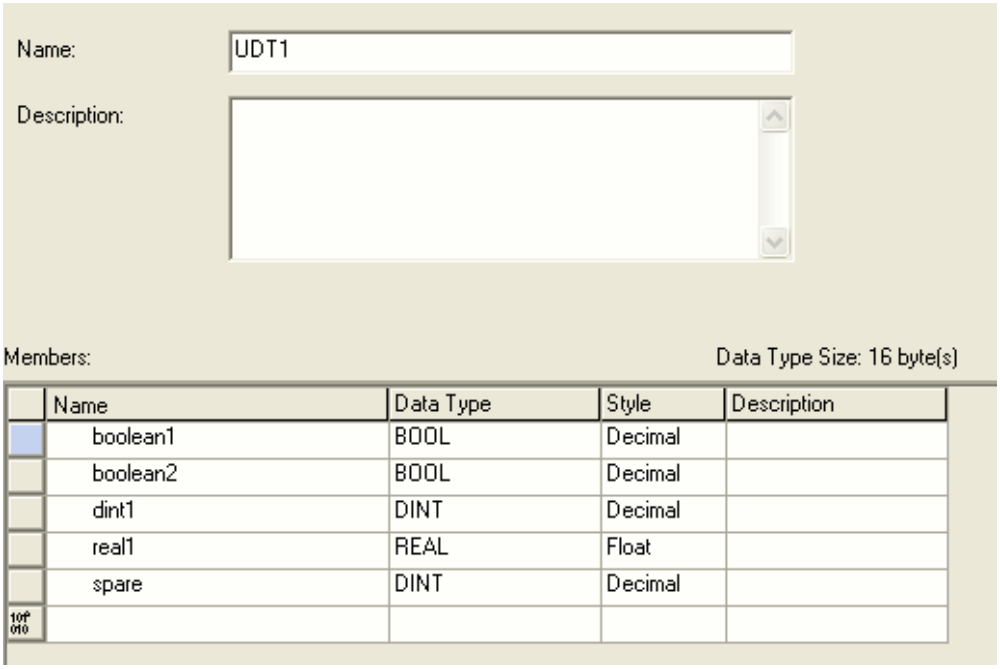
NOTE: data Unit type can only be used in the IEC editors. DUT components or arrays cannot be used in the MELSEC IL

Declaration: Data Unit types must be defined in the DUT_Pool. All the variables in the DUT structure must be declared in a special declaration table.

User Data Types in Logix

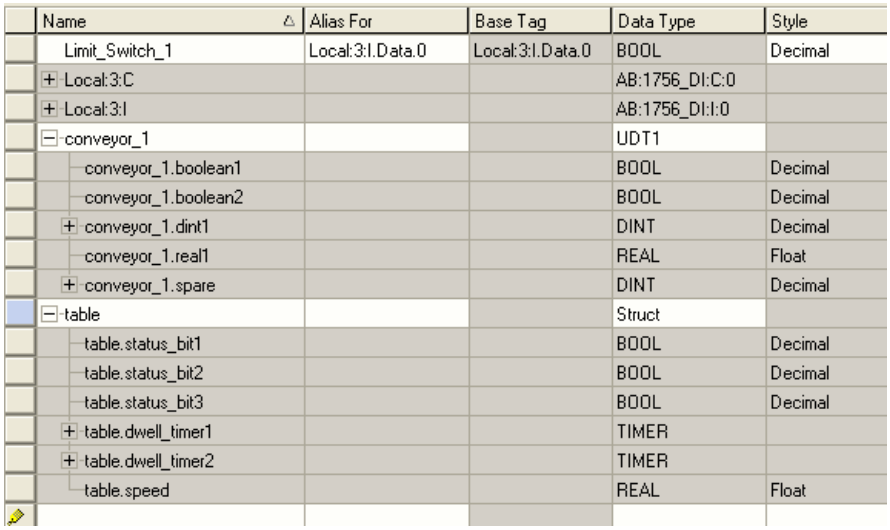
User-defined data types (structures) or UDTs let you organize your data to match your machine or process. The user can create a structure with multiple elements or varying data types as required for a particular application. The development of UDTs can both speed the software development process and make the program easier to understand.

Below is an example of a UDT:



	Name	Data Type	Style	Description
	boolean1	BOOL	Decimal	
	boolean2	BOOL	Decimal	
	dint1	DINT	Decimal	
	real1	REAL	Float	
	spare	DINT	Decimal	
10f 010				

Below, a structure is made for a conveyor and a table. Notice how the UDT contains various types of data within the structure. The conveyor contains Booleans, DINTs, REALs and others.



Name	Alias For	Base Tag	Data Type	Style
Limit_Switch_1	Local:3:I.Data.0	Local:3:I.Data.0	BOOL	Decimal
⊞ Local:3:C			AB:1756_DI:C:0	
⊞ Local:3:I			AB:1756_DI:I:0	
⊟ conveyor_1			UDT1	
conveyor_1.boolean1			BOOL	Decimal
conveyor_1.boolean2			BOOL	Decimal
⊞ conveyor_1.dint1			DINT	Decimal
conveyor_1.real1			REAL	Float
⊞ conveyor_1.spare			DINT	Decimal
⊟ table			Struct	
table.status_bit1			BOOL	Decimal
table.status_bit2			BOOL	Decimal
table.status_bit3			BOOL	Decimal
⊞ table.dwell_timer1			TIMER	
⊞ table.dwell_timer2			TIMER	
table.speed			REAL	Float

Pointers and Indexing

GX Developer

Indexing is an indirect setting made by using an index register. When Indexing is used in a sequence program, the device to be used will become the device number designated directly plus the contents of the index register.

For example, if D2Z2 has been designated, the designated device is calculated as follows: (2+3) = D5 and the contents Z2 is 3 become the designated device. There are 16 index registers, from Z0 to Z15. Each Index register can be set between -32,768 and 32,767.

With the exception of the restrictions noted below, indexing can be used with devices used with contact coils basic instructions and application instructions.

Devices which do NOT support Indexing
32-bit constant: K, H
Floating Decimal point data: E
Character String data: $
Bit designation for Word device: [], []
Function devices: FX, FY ,FD
Pointers used as labels: P
Interrupt Pointer used as labels: I
Index Register: Z
Step Relay: S
SFC Transfer devices: TR
SFC Block devices: BL
Value set for Timer: T, ST
Value set for Counter: C

Logix

In Logix there are no pointers. Arrays perform the same function as pointers, but are simpler and safer.

In computer programming, pointers to data are used principally for these three purposes:

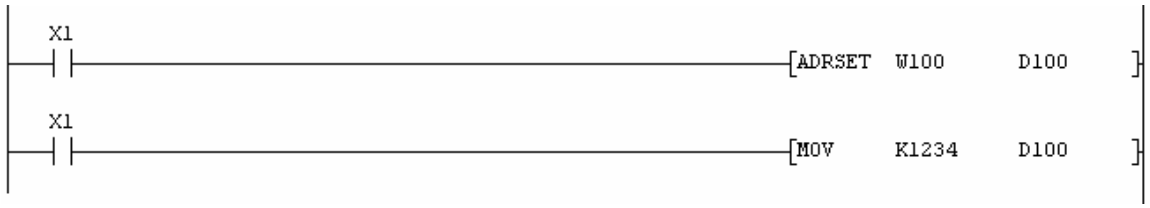
- Operations on sequentially ordered data items (arrays of objects, strings).
- Allocating, accessing, and deleting dynamically allocated objects.
- Passing references to objects as parameters in function calls.

In Logix the first purpose is covered by arrays. The second purpose is not relevant in control software because there are no dynamically allocated objects. The third is covered by 'inout' parameters in Logix Add-On Instructions.

Indirect Designation

Indirect Designation word devices (two points of word device), designate the address of the device to be used in a sequence program. This method can be used when the index register is insufficient.

For the designating devices, designate them as '@+' (word device number). For example, designating D@100 will designate the contents of D100 and D101 to the device address. The address of the device performing indirect designation can be confined with the ADRSET instruction.



Devices Capable of Indirect Designation:

Internal user devices: Word devices

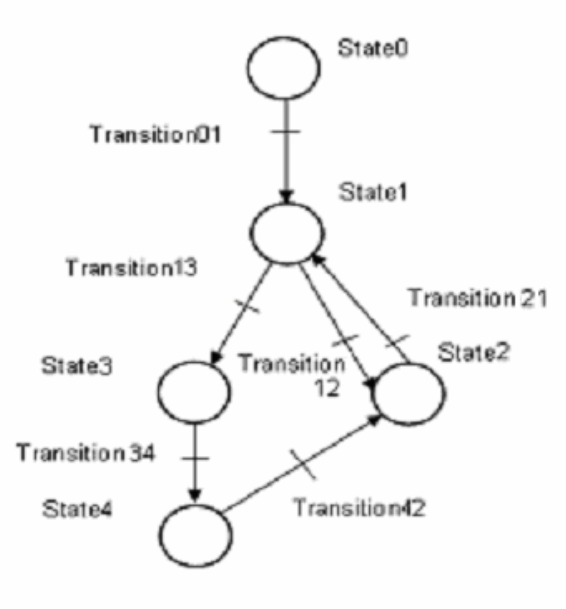
Link direct devices: Word devices

Intelligent function module devices

File register

State Machine

The State Machine is an important construct in control systems software because it greatly simplifies the task of programming sequential control.



Logix -State Machine in Structured Text

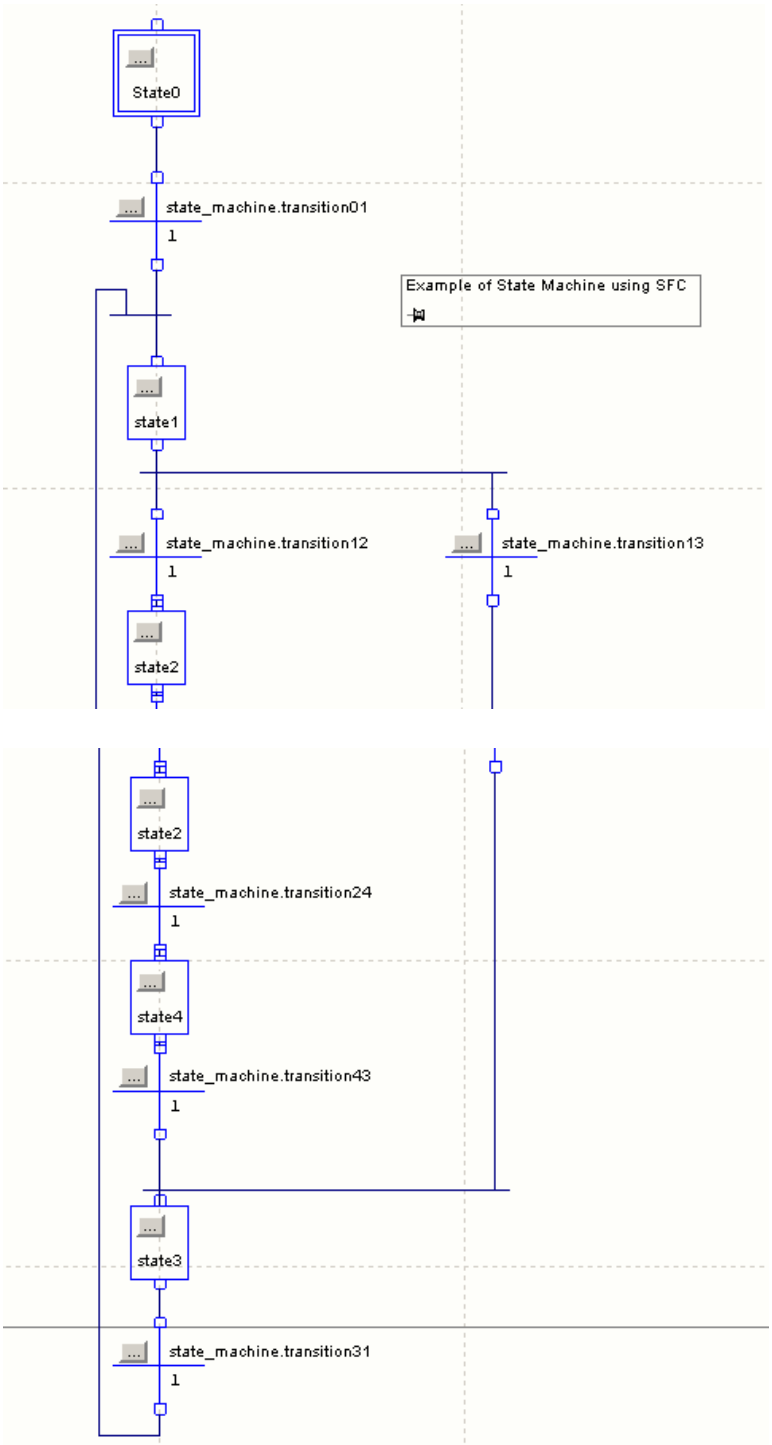
Below is the same state machine in Structured Text, using the CASE statement.

```
// implementation of State Machine using CASE in ST

case state_machine.state of
    0:  if state_machine.transition01 then
            state_machine.state := 1;
        end_if;
    1:  if state_machine.transition12 then
            state_machine.state := 2;
        elsif state_machine.transition13 then
            state_machine.state := 3;
        end_if;
    2:  if state_machine.transition24 then
            state_machine.state := 4;
        end_if;
    3:  if state_machine.transition31 then
            state_machine.state := 1;
        end_if;
    4:  if state_machine.transition43 then
            state_machine.state := 3;
        end_if;
end_case;
```

Logix -State Machine in Sequential Function Chart

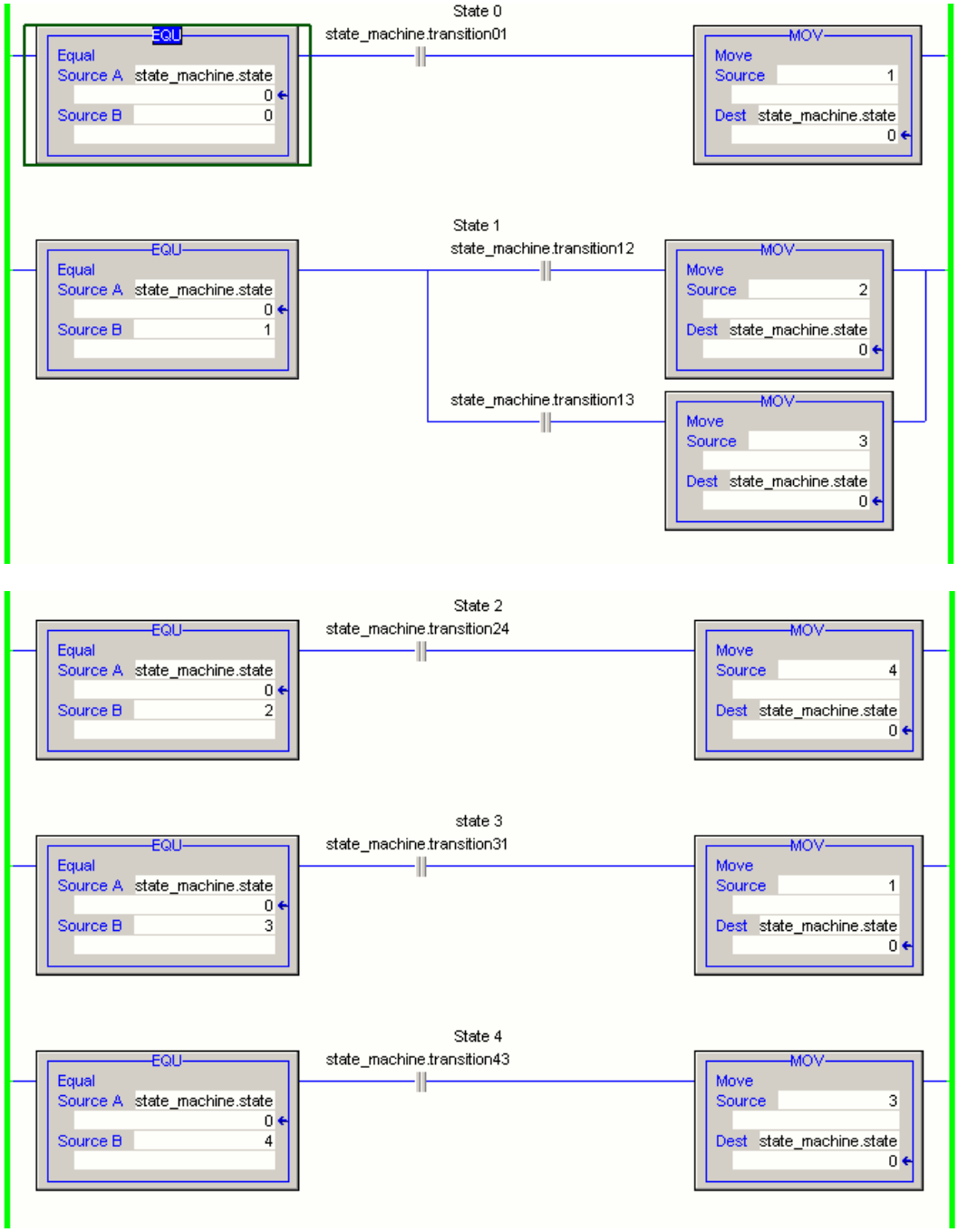
Logix provides s graphical SFC as one of its standard suite of languages. Shown below is the state machine in SFC.

Implementation of State Machine Using SFC Chart



Logix-State Machine in Ladder Diagram

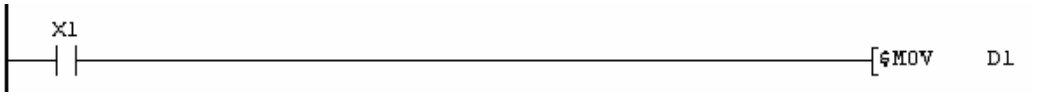
The window below shows how the state machine can be implemented in LD.



Strings

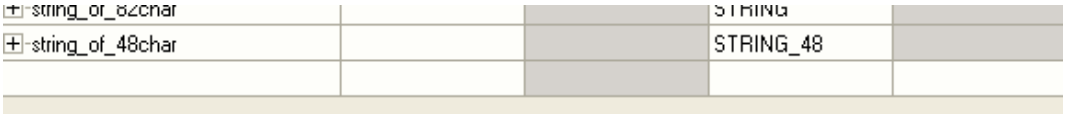
GX Developer

The data string STRING ($) possesses character strings. Character strings are all entered characters (max 50 characters).

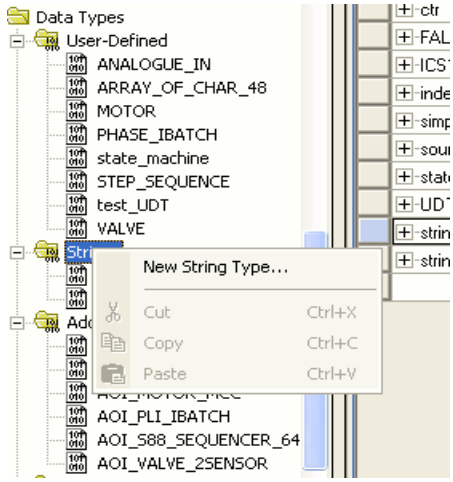


Logix

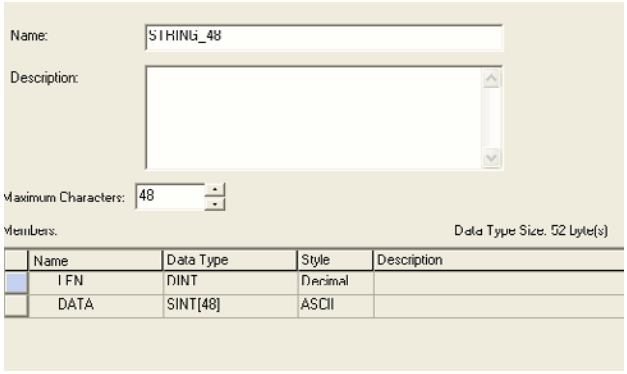
The example from the tag configuration table below shows how strings are defined in Logix.



To create a string of a length other than the 82-character default, right-click “Strings” in the project tree as shown below.



Then configure the properties as below.



Instances of the new type can be defined.

With instances of type STRING or STRING_48, there is a LEN field that automatically updates when a string constant is entered or when the string is manipulated by ASCII or STRING instructions

Temporary Files

GX Developer

Temporary files are created during compilation. These files are normally deleted when the program is finished. To keep the files from being deleted, start GX IEC Developer with the '/debug' command line option. To do this, add this option to the GX IEC developer command line in the Program item properties dialog box in the windows program manager.

: *c:\melsec\gx iec developer\sc.exe* /debug

Temporary files created during download.

- dwl_sym.asc
- upl_sym.pac
- upl_main.bin
- upl_sub.bin
- fnw_main.dis
- fnw_sub.dis
- fnw_imp.asc
- task_imp.asc

Temporary variables are used for local, temporary storage of intermediate values, and for pointers. They exist only while their block is executing, and their values are lost when the block terminates.

Logix

Logix does not have temporary variables. All storage is static that is, values are retained between code executions.

Local variables can be created for an Add-On Instruction. These variables can be used in the same way as temporary variables.

Functions

Functions are important because the development of such routines need to be done only once, and having been done, both the originator of the function and other programmers can do the same thing in a fraction of the time. This section discusses how functions can be implemented in Logix.

Logix

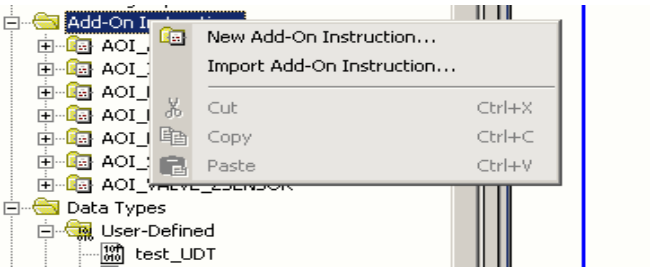
Function as Add-On Instruction

The Add-On Instruction has the same type of parameters as the FB (Input, Output and InOut) and it has its own data area. Once coded and tested, an Add-On Instruction can be used from anywhere in a program, and is sufficiently self-contained to be exported to other projects or placed in a code library.

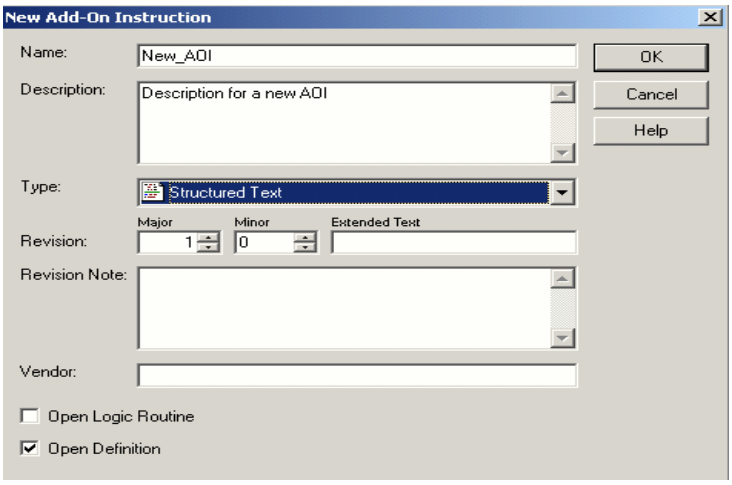
Example–a Ramp Function

This example takes a real variable and ramps it linearly from its current value to a new value at a specified rate.

Go to the Add-On Instructions branch of your project tree and right-click Add-On Instruction.

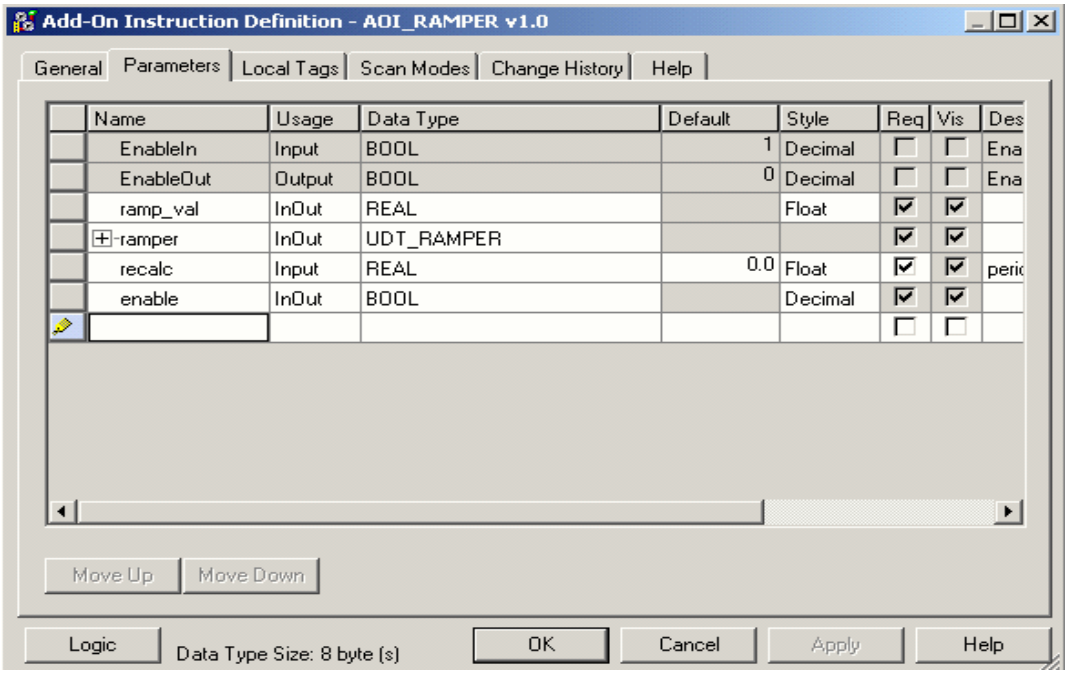


This form appears.



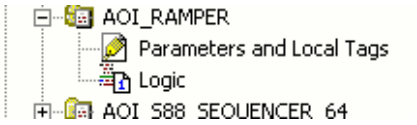
Enter the name of the Add-On Instruction and specify the language its code section will be written.

Click the Parameters tab.



Input parameters are values from the program to the Add-On Instruction. Output parameters are values from the Add-On Instruction to the Program. InOut parameters are for variables that will be modified by the Add-On Instruction. If there are any data structures, choose the InOut parameter because it is passed by reference and thus is more efficient.

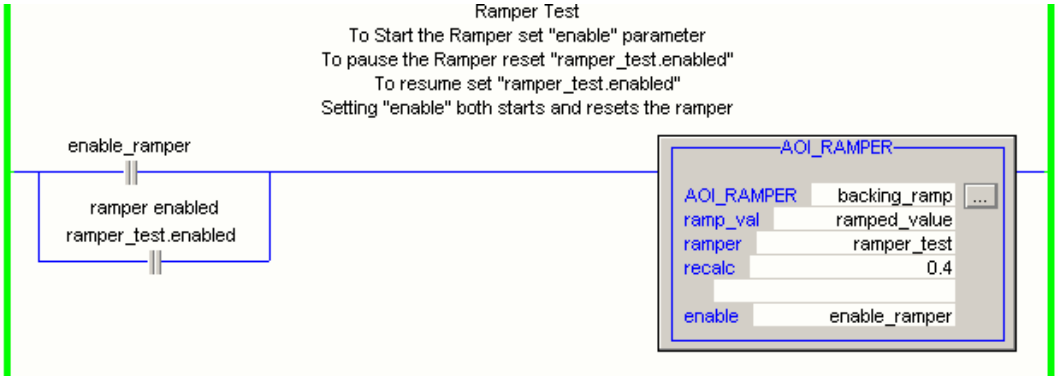
In the project tree for AOI_RAMPER, there is a logic section.



Open it to see the code for this Add-On Instruction.

```
// Ramps a real variable from its current valuse to a new value at a
// specified rate.

// Parameters:
//      ramp_val    - variable to be ramped
//      ramper      - instance of UDT UDT_RAMPER
//      recalc      - code recalculation period (s)
//      enable      - start signal

// To use - set the target value in ramper.RAMP_TARGET_ABS
//        - set the ramp rate in ramper.RAMP_RATE_ABS
//        - to Start the Ramper set "enable" parameter
//        - to pause the Ramper reset "ramper.enabled"
//        - to resume set "ramper.enabled"
//        - setting "enable" both starts and resets the ramper

// on completion, the UDT field "complete" is set and the UDT field
// "enabled" is reset

// when enable is set, initialise
if (enable & (enable xor ramper._enable)) then
    ramper.initial_output := ramp_val;
    ramper.change := ramper.RAMP_TARGET - ramp_val;
    ramper.increment := ramper.change / abs(ramper.change)
                            * ramper.RAMP_RATE_ABS * recalc;
    ramper.counter := 0;
    ramper.complete := 0;
    enable := 0;
    ramper.enabled := 1;
end_if;

ramper._enable := enable;
```

The Add-On Instruction can be called from any routine.



Note that with Add-On Instructions, a tag of type Add-On Instruction needs to be created in a data area that is visible to the routine. This is called a Backing tag.

Before writing an Add-On Instruction, check through the Instruction Help in RSLogix 5000 software. There might be an existing instruction that will do the job. The following section will illustrate this.

Block Copy, COP and CPS

GX Developer -Data Transfer Instructions

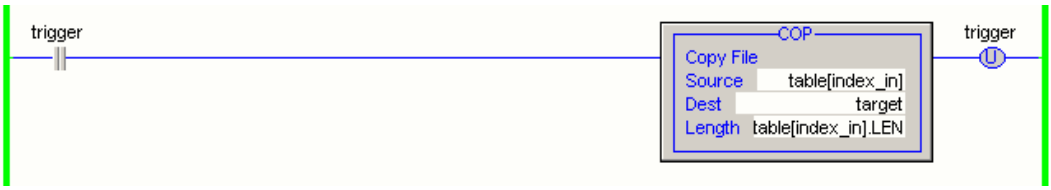
These instructions transfer, invert, or exchange data. In total 24 different instructions are supplied.

Instructions	Bin Data Transfer	Transfer of Floating Point Data	Transfer of Character String Data	Inverted BIN Data Transfer	Block Data Transfer	Block Transfer of Identical Data	BIN Data Exchange	BIN Data Exchange (16 bit blocks)	Byte Exchange
MELSEC Editor	MOV, MOVP, DMOV, DMOVP	EMOV, EMOVP	$MOV $MOVP	CML, CMLP DCML DCMLP	BMOV BMOVP	FMOV FMOVP	XCH XCHP DXCH DXCHP	BXCH BXCHP	SWAP SWAPP
IEC Editor	MOV_M MOVP_M DMOV_M DMOVP_M	EMOV_M EMOVP_M	STRING_MOV_M STRING_MOVP_M	CML_M CMLP_M DCML_M DCMLP_M	BMOV_M BMOVP_M	FMOV_M FMOVP_M	XCH_M XCH_M DXCH_M DXCHP_M	BXCH_M BXCHP_M	SWAP_MD SWAP_P_MD

Logix

In Logix, the COP built-in instruction will save all of the work.

In this case, the copy is between two arrays and the indexes are defined by indexSource and indexDest.

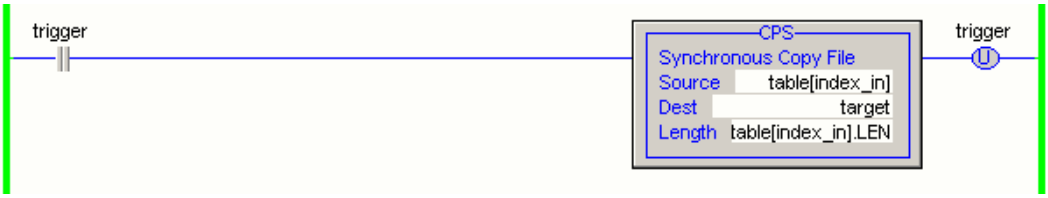


Because the source and destination specifications can include variable array indexes, COP will do the job. It is the equivalent of 'INDEXED_COPY'.

The CPS instruction is the same as COP but with one difference. The instruction cannot be interrupted. Therefore, the source and destination data will remain constant throughout its execution. Use CPS to move data that can change.

Examples include:

- Copying input data to a buffer from where the program will operate on the data.
- Copying consumed tags to a buffer from where the program will operate on the data.


trigger
CPS
Synchronous Copy File
Source table[index_in]
Dest target
Length table[index_in].LEN
trigger
U

Mathematical Expressions

GX Developer-Special Function Instructions

The following instructions are available for performing mathematical computations.

Instructions for Mathematical Computations	
SIN operation on floating decimal point data	SIN (P)
COS operation on floating decimal point data	COS(P)
TAN operation on floating decimal point data	TAN(P)
SIN -1 operation on a floating decimal point	ASIN(P)
COS-1 operation on a floating decimal point	ACOS(P)
TAN-1 operation on a floating decimal point	ATAN(P)
Conversion from floating decimal point angle to radian	RAD(P)
Conversion from floating decimal point radian to angle	DEG(P)
Square root operations for floating decimal point data	SQR(P)
Exponent operations on floating decimal point data	EXP(P)
Natural logarithmic operations on floating decimal point data	LOG(P)
Random number generation and series updates	RND(P), SRND(P)
BCD 4-digit and 8-digit square roots	BSQR(P), BDSQR(P)
BCD type SIN operation	BSIN(P)
BCD type COS operation	BCOS(P)
BCD type TAN operation	BTAN(P)
BCD type SIN-1 operation	BASIN(P)
BCD type COS-1 operation	BACOS(P)
BCD type TAN-1 operation	BATAN(P)

Example in GX Developer

This section will describe how the GX Developer programmer can perform mathematical computations in Logix. The following example will be used - the expression 'v(cos(x)^2 + sin(x)^2)'. The result of this expression is always exactly 1.

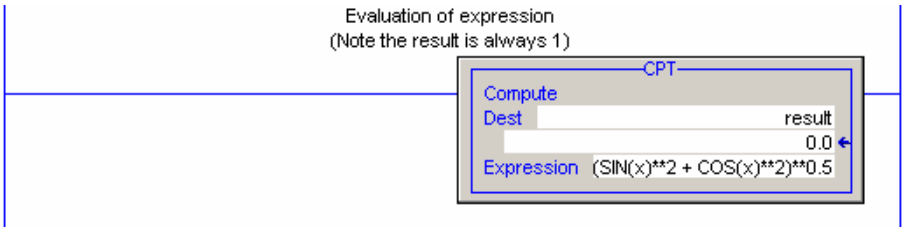
Logix - ST

The expression is entered in the same way as with any other high-level language.

```
// evaluation of mathematical expression in Structured Text

result := (SIN(x)**2 + COS(x)**2)**0.5;
```

Logix - LD

The CPT instruction enables the expression to be entered in a high-level manner.



Type Checking

With both GX Developer and Logix, parameters to Functions, Function Blocks, Instructions, and Add-On Instructions are strictly type-checked by their compilers.

In GX Developer the type of arithmetic operations must be specified. There are for example, '*I' (multiply two 16 bit integers), '*D' (multiply two 32 bit integers), and '*R' (multiply two reals). It is up to the programmer to be sure that the two numbers that are the operands of an '*R' instruction are reals. If they are not, the compiler will accept it but the result will be incorrect.

There are differences with mathematical expressions. Logix distinguishes between numeric and Boolean values. The compiler will reject expressions that illogically mix numeric and Boolean values. When it encounters expressions of mixed numeric type, it will make conversions to produce a result of the type of the declared result variable. Hence, it will interpret '*' as integer multiplication if the result is to be an integer and as real multiplication if the result is to be real.

Conclusion

The Logix methods of programming mathematical expressions are clearer, and by separating math code from other logic it will simplify testing and validation.

Other Topics Related to Programming

This is an area where Logix differs considerably from GX Developer.

Rules for GX Developer

- Temporary variables are invisible outside the block in which they are declared.
- Global static variables are visible throughout the program.
- Static variables that are declared as instance data to a function block have a special status in the FB, but they can be accessed from other parts of the program.

Rules for Logix

- Execution in Logix is divided into tasks. Each task may have several programs and each program may have several routines. Each program may have its own tag section.
- Controller scope tags are visible throughout all routines in all programs.
- Program scope tags are visible only in the routines in the program in which they are defined. This means that if a routine in one program is to share data with a routine in another program, it must use controller scope data.
- Add-On Instruction local tags are visible only to that Add-On Instruction logic.

GX Developer PoUs, Tasks, and Scheduling

Program organization unit, tasks, and scheduling are described in Chapter 2.

Larger Example–Control Module

This example will assemble some of the different topics illustrated in the previous sections. The term “Control Module” (CM) comes from the influential S88 Batch Control standard. S88 has encouraged controller software design to be more “object oriented”. This control module is for a binary valve. The Add-On Instruction is suitable for this type of programming.

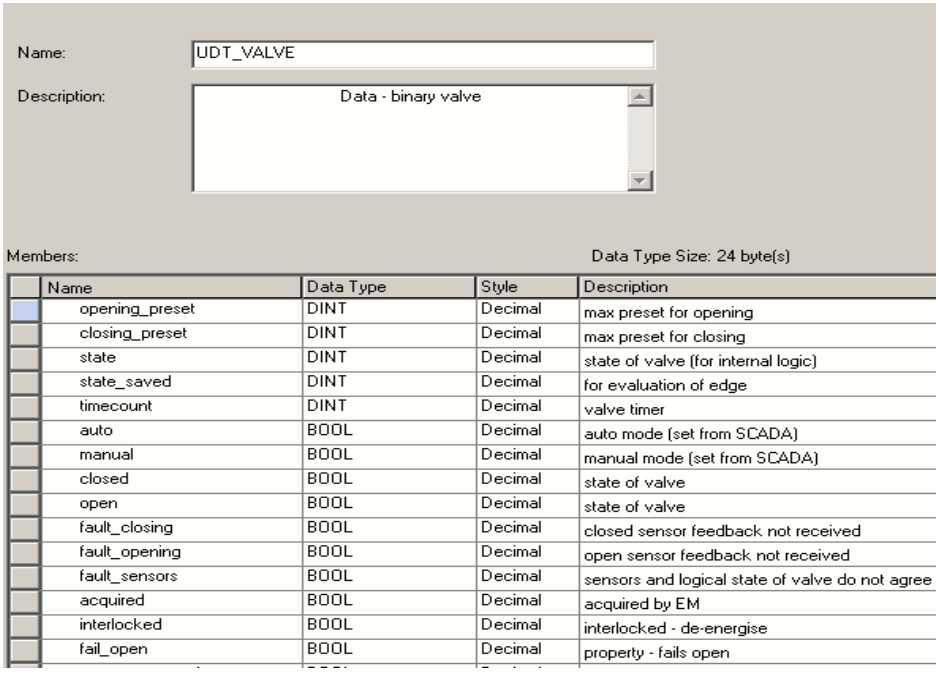
Components of the CM

These are:

- A UDT called UDT_VALVE.
- An Add-On Instruction called AOI_VALVE_2SENSOR
- A new Program under ‘task_02s’ called ‘valves_callup’, which contains program tags, sections, and a routine.

Logix-User Data Type (UDT)

The UDT is shown below.



Name: UDT_VALVE

Description: Data - binary valve

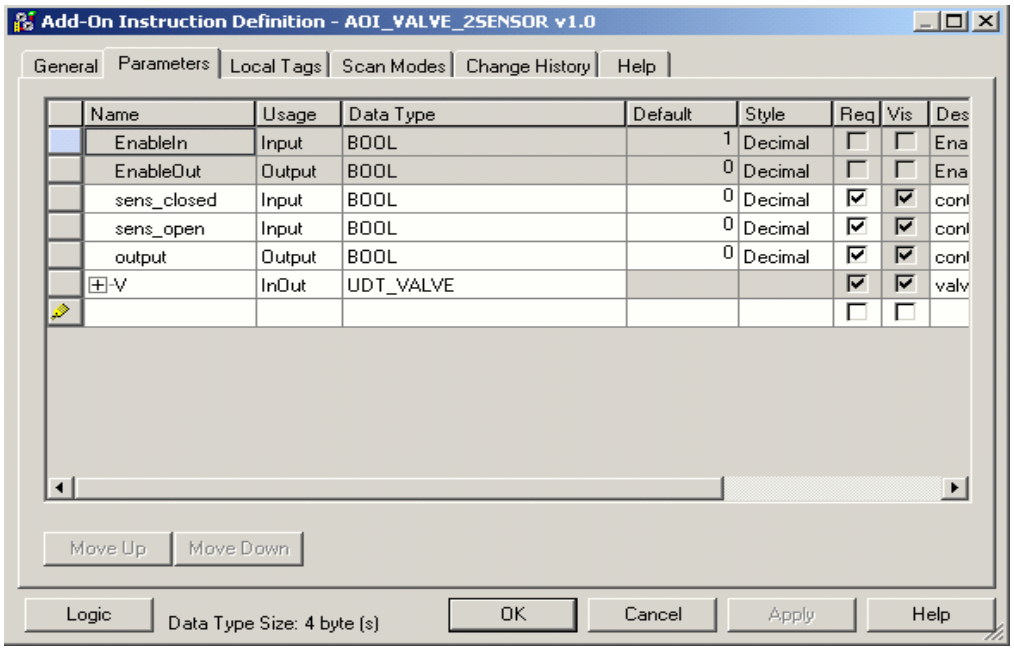
Members: Data Type Size: 24 byte(s)

Name	Data Type	Style	Description
opening_preset	DINT	Decimal	max preset for opening
closing_preset	DINT	Decimal	max preset for closing
state	DINT	Decimal	state of valve (for internal logic)
state_saved	DINT	Decimal	for evaluation of edge
timecount	DINT	Decimal	valve timer
auto	BOOL	Decimal	auto mode (set from SCADA)
manual	BOOL	Decimal	manual mode (set from SCADA)
closed	BOOL	Decimal	state of valve
open	BOOL	Decimal	state of valve
fault_closing	BOOL	Decimal	closed sensor feedback not received
fault_opening	BOOL	Decimal	open sensor feedback not received
fault_sensors	BOOL	Decimal	sensors and logical state of valve do not agree
acquired	BOOL	Decimal	acquired by EM
interlocked	BOOL	Decimal	interlocked - de-energise
fail_open	BOOL	Decimal	property - fails open

Building the UDT should be the first step - it includes all the data that is necessary to model the valve.

Logix-Add-On Instruction Parameters

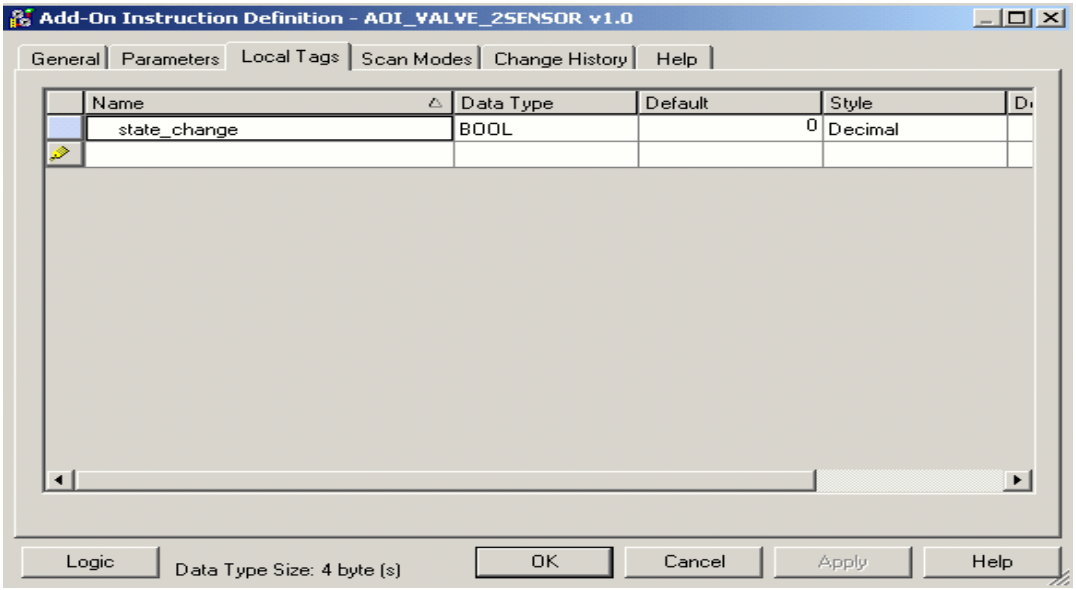
The dialog example below shows the parameter configuration screen.



The parameters that have been added are the I/O for the valve, and an object of type "UDT_VALVE". "V" must be an InOut parameter.

Logix-Add-On Instruction Local Data

The dialog example below shows the configuration page for the Add-On Instruction local data.



Logix-Add-On Instruction Logic

The code example below shows the logic for this Add-On Instruction.

```
// Control Module Valve_2sensor
// ----------------------------
// Implements logic for a valve with an open and a closed sensor and one output

// See UDT Valve for data structure.

// Note the open/close command V.open_command must be set or reset externally
// and then left until the next activation is required. Do not continuously
// hold the flag set or reset.

// increment timer counter
V.timecount := V.timecount + 1;

// evaluate change of state (state machine)
state_change := V.state <> V.state_saved;
V.state_saved := V.state;

// set output
output := (V.fail_open xor V.open_command) and not
                                        (V.interlocked or V.faulted);

// valve is faulted
V.faulted := V.fault_opening or V.fault_closing or V.fault_sensors;

// action on fault or interlock

else;
end_case;
// end state machine

// reset timer if change of state
if (state_change) then V.timecount := 0;
end_if;

// external fault reset
if (V.clear_faults) then
    V.fault_opening := 0;
    V.fault_closing := 0;
    V.fault_sensors := 0;
    V.clear_faults := 0;
end_if;
```

```
    // possible close command while waiting to open
    elsif not V.open_command then
        V.state := 0;
    // fault opening
    else
        V.fault_opening := (V.timecount >  V.opening_preset);
    end_if;
// state 3 - open - wait for close command
3:  V.closed := 0;
    V.open := 1;
    if (not V.open_command) then
        V.state := 4;
    // fault sensors
    else
        V.fault_sensors := (sens_closed) or (not sens_open);
    end_if;
// state 4 -
4:  V.state := 5;
// state 5 - wait for closed sensor
5:  if (sens_closed & not sens_open) then
        V.state := 0;
    // possible open command while waiting to close
    elsif V.open_command then
        V.state := 3;
    // fault closing
    else
        V.fault_closing := (V.timecount >  V.closing_preset);
    end_if;
```

```
if V.faulted or V.interlocked then
    if V.fail_open then
        V.state := 3;
        V.open_command := 1;
    else
        V.state := 0;
        V.open_command := 0;
    end_if;
end_if;
```

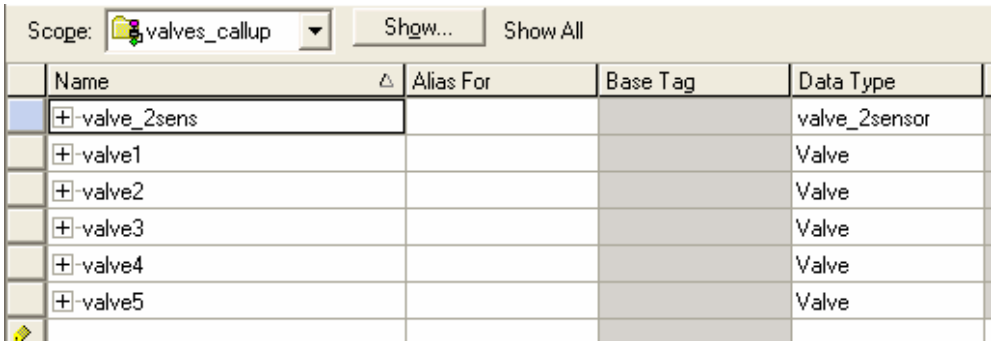
```
// state machine:
// the state machine does not set outputs - it monitors inputs
// to set status and faults.
case V.state of
    // state 0 - valve is closed - wait for open command
    0:  V.closed := 1;
        V.open := 0;
        if (V.open_Command) then
            V.state := 1;
        // fault sensors
        else
            V.fault_sensors := (not sens_closed) or (sens_open);
        end_if;
    // state 1 -
    1:  V.state := 2;
    // state 2 - waiting for open sensor
    2:  if (sens_open & not sens_closed) then
            V.state := 3;
```

The tags referred to in this logic are all parameters or local tags. This means that the Add-On Instruction could be used in any program (provided the UDT Valve is also present).

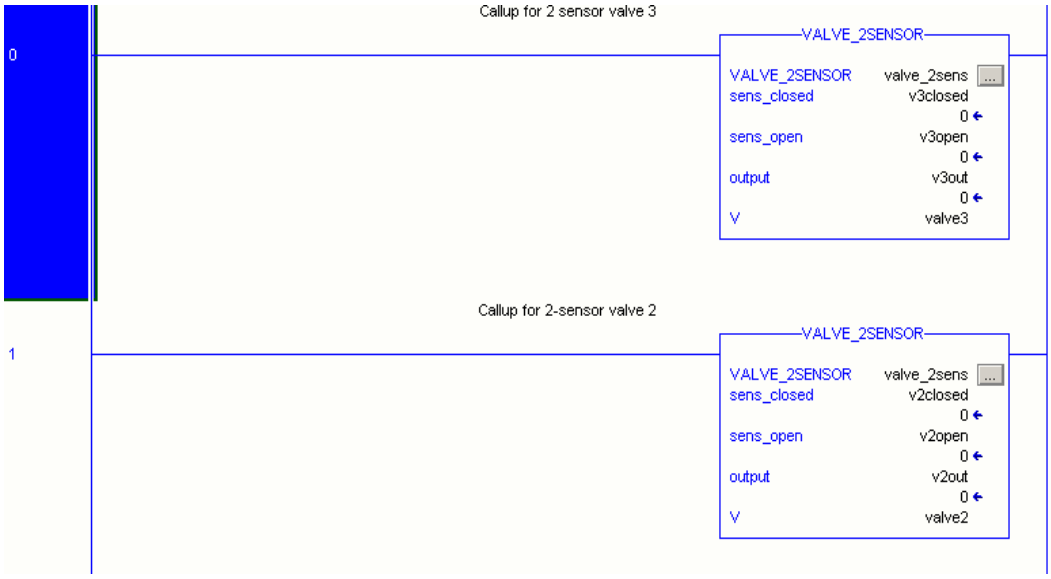
Logix-Call-up

Both the call-up code and the instances of UDT Valve are located in the program “valves_callup”, which runs under task_02s. The frequency with which the call-up code is executed depends on the application and the size of the valve.

The example below shows the data instances.



Add an instance of type Valve for each physical valve. The first tag is the required ‘backing tag’ for the Add-On Instruction.



Call the Add-On Instruction once for each valve. The actual parameters are the actual I/O tags for the valve's sensors and solenoid, and the instance of UDT “valve”.

The I/O tags will appear only in the call to the Add-On Instruction. They will not be used anywhere else in the program. Apart from being tidier from the software structure point of view, this cancels any risk of problems arising from asynchronous updating of I/O.

Remember that with Logix controllers the I/O are scanned asynchronously.

Chapter 5: Common Mistakes When Converting to Logix

Introduction

The objective of this section is to point out some of the design and programming mistakes that GX Developer users often make when converting applications to Logix. These mistakes have been identified by examination of Logix programs that have been converted from GX Developer.

Programming mistakes fall into these two categories:

- Programming that leads to reduced controller efficiency.
- Programming that leads to a control system that is difficult to understand, maintain, and develop.

In most cases, coding for efficiency will also improve the readability and modularity of the program. Conversely, improving the program's structure should also make it more efficient.

Not Selecting Appropriate Hardware

This chapter is concerned mainly with software. Remember that the correct selection of hardware is a requirement for satisfactory operation. It is possible that the number of controllers and racks may not be the same as for an equivalent GX Developer system.

Underestimating Impact of Task Scheduling

In the area of scheduling and interrupts, there is not much difference in the capability of the two systems. However, in the Logix scheduling is more actively encouraged.

It is quite common for GX Developer programmers to neglect scheduling when working with Logix controllers. See Chapter 2 for a more detailed account of scheduling in Logix.

Performing Translation Instead of Conversion

It is a common mistake to translate a GX Developer program line-by-line to Logix. Instead, a more thorough process is needed, which is described as conversion. This will cover choice of languages, scheduling, and choice of code routines.

By converting rather than translating GX Developer programs, a user will make better use of the capability of your Logix system.

Not Using the Most Appropriate Logix Languages

Programmers often neglect Logix languages other than ladder logic.

Read Chapter 2 for a discussion of how to choose a Logix language and Chapter 4 for examples of GX Developer code translated into Logix.

Implementation of Incorrect Data Types–DINT versus INT

It is commonly advised to use DINT rather than INT. The example below shows an addition of two DINTs versus the addition of two INTs.

Add DINTs

```
// add two DINTs and assign to a third DINT

for index := 0 to 999 do
    result_DINT := operandA_DINT + operandB_DINT;
end_for;
```

Add INTs

```
// add two INTs and assign to a third INT

for index := 0 to 999 do
    result_INT := operandA_INT + operandB_INT;
end_for;
```

Timing Results

The table below shows relative times (smaller numbers are faster). The numbers here are for comparison only with other numbers in the table. They should not be compared with entries in other tables.

Method	Relative Times
Add DINTs with ST For Loop	53
Add INTs with ST For Loop	100

A user should use DINT for all integer work in Logix. They should only use INT or SINT only if they are interfacing to an external system that requires the use of INTs or SINTs.

User Code Emulating Existing Instructions

Programmers often write user code when an existing instruction will do the job. As an example, compare copying an array with user code and with the COP instruction.

User Code

```
for index := 0 to 99 do
    target_DINT[index] := source_DINT[index];
end_for;
```

COP Instruction

```
cop(source_DINT[0], target_DINT[0], 100);
```

Below are the relative timings for the two methods. Again, the numbers here are for comparison only with other numbers in the table. They should not be compared with entries in other tables.

Method	Relative Timing
Copy array of DINTs with structured text	100
Copy array of DINTs with COP	18

To perform operations like copy arrays, GX Developer library functions that are written in IL are used. If the library function does not do what is required, a new one can be written. The functions written can be almost as efficient as the ones that GX Developer provides.

However, in Logix it is difficult for a programmer to write a copying function that is as efficient as the built-in COP. The lesson for GX Developer programmers is to check the Instruction Help in RSLogix 5000 software carefully.

Incorrect Usage of COP, MOV, and CPS

MOV copies a simple value (immediate or tag) to a simple tag type–DINT, INT, SINT, or REAL. COP can do the same as MOV (the source cannot be an **immediate** value), but its more important use is to copy complex data types. It would be a minor programming mistake to use COP to copy simple data types.

A mistake that is seen often is to use multiple MOVs to copy a data structure when one COP could be used.

If the source data could change during copying due to asynchronous I/O updates, use CPS instead. This instruction cannot be interrupted so source data will remain constant while copying.

Incorrect Usage of CPT

In Logix, the CPT instruction can be used to evaluate expressions. The expression is entered in one of the fields of the instruction. It is very convenient.

However, CPT should be used only if more than one arithmetic instruction would be required to evaluate the expression. If a single instruction is sufficient it will be faster than CPT.

Read more about CPT in Chapter 4.

Not Handling Strings in Optimal Way

If a user wants to define a new String type for example, with a different number of characters than the default 82, it would be a mistake to create a new “User **Data** Type”. Instead, create a new String data type. The advantage of doing it this way is that the “LEN” field will automatically update as the length of the string changes.

Extensive Usage of Jumps

In Logix jumps can occur only in Ladder Logic. It is recommended that the JMP instruction be used sparingly. Jumps in ladder logic often make the program difficult to read.

Not Using Aliased Tags

Remember to create aliased tags for the I/O tags that RSLogix 5000 software creates. They will make the program easier to read. See Chapter 2 for more information.

Chapter 6: GX Developer to Logix Glossary

Introduction

This chapter provides a glossary of GX Developer terms.

Hardware Terminology

Mitsubishi Term	Definition	Logix Term	Definition
Interface modules	Connects the remote I/O station and system with the PLC CPU	Bridge	
Controller	A unit, such as a programmable controller, that controls machine or process elements	Controller	
CPU	Central processing unit	Controller	
Ethernet	Network for connection of information processors, such as personal computers and workstations	EtherNet/IP, ControlNet	Both of these networks have the same functionality as the Industrial Ethernet network
Serial communication module	Modules QJ71C24 and QJ71C24-R2 enable communication with peripheral devices via a standard serial interface	Serial	DF1 or DH485 protocols
PLC	Programmable logic controller	Controller	
Profibus DP	Enables extremely fast data exchanges with a wide variety of slave devices, including remote I/O, frequency inverters, operator terminals, and other devices from third-party manufacturers	EtherNet/IP, ControlNet, DeviceNet	Ethernet/IP and ControlNet networks can perform most functions of the Profibus DP network, but the DeviceNet network has the closest correspondence
MC protocol	Used by the Q-Series PLC and external devices to read and write device data and the programs of the PLC CPU via the Q-Series C24 or E71	Common Industrial Protocol (CIP)	Unifies the ControlNet, EtherNet/IP, and DeviceNet networks

Software Terminology

GX Developer Term	Comment	Nearest Logix Term	Comment
Accumulator	A result handling facility used in the IEC instruction list	N/A	In Logix languages, there is no need to access low-level structures of the CPU
Array	Derived data types, consisting of fields of one kind of variable, definable up to three dimensions	Array	Syntax REAL[8] indexing always starts at 0
Data areas for address declaration	Addresses X,Y,D, and so forth	N/A	Use tags
Data transfer instruction	These instructions transfer, invert, or exchange data (24 instructions are supplied)	COP	Instruction (use MOV for a simple variable)
BOOL		BOOL	
Task pool is the control centre for program execution	Tasks in the task pool are polled cyclically	Continuous task	Continuously executed
Global variable Local variable	Has fixed hardware addresses (absolute addresses) that are declared globally for the entire project and can be referenced from all function blocks Accessible only within the POUs in which they are declared	Controller-scope or program-scope tag database	Global - visible within the program that the database is linked to
DINT	Double integer	DINT	Double integer
DWORD	32-bit word	DINT	
FBD	Function block diagram	FBD	Function block diagram
Function	A program organization unit (POU) with the class 'Function' - can be used in the same way as a normal programming instruction	Routine Add-On Instruction	Both of these **could** correspond to a function
Function block	A program organization unit (POU) with the class 'Function' - can be used in the same way as a normal programming instruction	Routine Add-On Instruction program	All of these **could** correspond to a function block
PLC parameter option in the project data list dialog box	Read PLC data button on the I/O assignment tab causes the module configuration information to be read back on the dialog box	I/O configuration	Branch of controller organizer
INT	Integer	INT	Use is deprecated (slower than DINT)
Event triggered tasks	Interrupt triggered execution control is possible only with	Periodic task	Periodically executing task

	input modules that support this function		
LD	Ladder Logic	LD	Ladder Logic
Library	Three types of libraries: standard, manufacturer, and user	GSV, SSV	Instructions - Get System Value and Set System Value
Task pool	Control center for the execution of the programs	Task	Program unit called by the operating system
Index register	Pointers	N/A	Use arrays
REAL	32-bit floating point number	REAL	32-bit floating point number
Project data list	Contains all of the tasks defined in the dialog box	Controller organizer	Component of RSLogix 5000 software
GX Developer	Development and monitoring software for MELSOFT controllers	RSLogix 5000 software	Development and monitoring software for Logix
Instruction list	A programming language, GX IEC developer supports two instruction list variants, IEC and MELSEC	N/A	Use Structured Text, Ladder Logic, or Sequential Function Chart
STRING	Contains characters	STRING	Sequence of SINTs (default length 82) that contains its length as property .LEN
Data areas/system variable	Name for data memory address	Tag	Tag defines the structure of the variable **and** reserves memory
Temporary variables	Scope is the program block in which they are defined and their lifetime is the execution of the program block in which they are defined	N/A	Use tags
WORD	16-bit word	INT	
DUT	Data unit types	UDT	User data type
Macros	By naming any ladder patterns (macro names) and registering them to a file (macro registration), merely entering simple instructions allows the registered ladder patterns to be read and the devices to be changed for data diversion	N/A	
Label	The word ‘label’ is automatically displayed as a dummy variable when you are programming jump instructions in the graphical editors	N/A	

MELSEC System Q and Rockwell Automation Comparison

Product		**Mitsubishi**				**Rockwell Automation**		
Cat. No.		**Q Series**				**CompactLogix L3X**	**Compact Logix L43**	**Advantage**
Models Available		**FX3U**	**Q02**	**Q02H**	**Q06H**	**L31, L32E, L32C, L35CR, L35E**	**L43**	
Memory	User program memory	64 KB steps	28 KB steps	28 KB steps	60 KB steps	1.5 MB	2 MB	More memory better for future expansion. Tag based structure makes it easy. The 1768 controller does not require a battery.
Local I/O points		256	4096	4096	4096	L32E-512, L35E-960	512	
Network I/O + Local I/O		16…384 (discrete I/O, max 256)	8192	8192	8192	10,000 and more	10,000 and more	More I/O capacity when connecting in network-DeviceNet, Ethernet, ControlNet.
Expansion racks		Yes	7	7	7	Up to 30 modules	18 modules (2 + 16)	More modules can be connected locally and many more by using DeviceNet adapter modules remotely. Orientation can be either horizontal or vertical.
Multiple CPUs		Yes (up to 4)				No	No	A CompactLogix system can have many CPUs in the network. Data can be exchanged by using Producer/Consumer model.
Communication (Ethernet)		A lot of configuration required to get connected				Built in	Extra module required	Very easy getting connected to the EtherNet/IP network.
Programming		GX Developer-Ladder and SFC only				RSLogix-Ladder, SFC/ST, FBD	RSLogix-Ladder, SFC/ST, FBD	Gives an advantage to SFC/ST and FBD in process and so forth.

Product		Mitsubishi				Rockwell Automation		
Catalog No.		Q Series				Compact Logix L3X	Compact Logix L43	Advantage
Models Available		**FX3U**	**Q02**	**Q02H**	**Q06H**	**L31, L32E, L32C, L35CR, L35E**	**L43**	
Networks supported and equivalent	EtherNet/IP (open)	Does not support I/O on EtherNet/IP network				EtherNet/IP network is built in L32E and L35E	1768-ENBT available	Supports I/O on EtherNet/IP network and in all Logix families.
	DeviceNet (open)	Yes - Their device level network is CC Link				Interfaces with as many as 30 Compact I/O modules		Supports multi-master in the DeviceNet network. (More SDNs can be connected.)
	ControlNet (open)	Equivalent of ControlNet network- MELSECNET is not open				1769-L35CR available	1768-CNB coming soon	MELSECNET is a proprietary network.
Producer/Consumer model bus		No				Yes		Very important for deterministic communication.
User-defined structure		No				Yes		You can create you own data structures.
Array type		No				Yes		Pointing to memory becomes easy.
Programming software		GX Developer for sequence control and MT Developer for motion control				RSLogix 5000 sequence, process, motion, and drive control		Programming languages serve a purpose in the platform it is being used. Ladder Diagram is used for sequential control applications, such as material handling, assembly machines, and motion applications. SFC is useful for managing process batch and repetitive machine control. ST is used for a programmer familiar with high-level programming.
Remote programming		Possible with lots of difficulty				Yes		User can browse any controller on network.
Communication within N/W		Additional coding and hardware required				Seamless connectivity		Multiple physical networks appear as a single network to the user.

Mitsubishi GT15 Series HMI Cross-reference

S No	Feature	GT15-75VNBA	PVP	PVP CE
1	Color	256 color (16 bit color graphics)	324 color (18 bit color graphics)	324 color (18 bit color graphics)
2	Diagonal	10.4 in.	10.4 in.	10.4 in.
3	Size (W x H)	211 x 158	211 x 158	211 x 158
4	Type	TFT	Color active matrix, thin-film transistor (TFT) with liquid crystal display (LCD)	Color active matrix, thin-film transistor (TFT) with liquid crystal display (LCD)
5	Resolution (Pixel)	640 x 480	640 x 480	640 x 480
6	User inputs	Touchscreen	Touch, keypad, touch and keypad	Touch, keypad, touch and keypad
7	CPU	64 bit RISC-CPU	324 color (18 bit color graphics)	650 MHz Celeron processor
8	Internal memory	5 MB	Standard: 128 MB RAM/ 128 MB Flash Extended: 256 MB RAM/ 256 MB Flash, field-upgradeable to 512 MB, 20 MB required for operating system and FTView ME software	Standard: 128 MB RAM/ 128 MB Flash Extended: 256 MB RAM/ 256 MB Flash, field-upgradeable to 512 MB, 20 MB required for operating system and FTView ME software
9	Communication interface	RS232, front USB mini-B	Ethernet, RS-232, 2 USB plus	Ethernet, RS-232, 2 USB plus
10	Memory slot	CompactFlash card slot	CompactFlash card slot	CompactFlash card slot
11	Operating ambient temp - display	0…50 °C (*6)	0…55 °C (32…131 °F)	0…55 °C (32…131 °F)
12	Operating ambient temp - other than display	0…55 °C (*6)	0…55 °C (32…131 °F)	0…55 °C (32…131 °F)
13	Storage ambient temp	-20…60 °C	. -25…70 °C (-13…158 °F)	. -25…70 °C (-13…158 °F)
14	Operating ambient humidity	10…90% RH, noncondensing (*1)	5…95% without condensation	5…95% without condensation
15	Storage ambient humidity	10…90% RH, noncondensing (*1)	5…95% without condensation	5…95% without condensation
16	Shock - operating	5…9 Hz	0.012 in p-p, 10…57 Hz	0.012 in p-p, 10…57 Hz
17	Shock - nonoperating	9…150 Hz	2 g peak, 57…500 Hz	2 g peak, 57…500 Hz
18	Operating atmosphere	No corrosive gas	NEMA Type 12, 13, 4X, IP54, IP65	NEMA Type 12, 13, 4X, IP54, IP65
19	Altitude (*3)	2000 m or less	2000 m	2000 m
20	Installation location	Inside the control panel	NEMA Type 12, 13, 4X, IP54, IP65	NEMA Type 12, 13, 4X, IP54, IP65
21	Input power supply voltage	100…240V AC (10…15%)	85…264V AC	85…264V AC
22	Input frequency	50/60 Hz 5%	47…63 Hz	47…63 Hz
23	Input max apparent power	110VA (at max load)	160VA max	160VA max
24	Contrast adjustment	Not possible	Not possible	Not possible

S.No	Feature	GT15-75VNBA	PVP	PVP CE
25	Intensity	280 [cd/m2]	300 cd/m2 Nits	300 cd/m2 Nits
26	Intensity adjustment	4-level	10-level	10-level
27	Life (*2)	Approx. 41,000 hr (operating ambient temp: 25 °C)	CCFL 50,000 hr life, min.	CCFL 50,000 hours life, min.
28	Number of touch keys	1200 objects/screen (matrix structure of 30 lines and 40 columns)	Analog resistive	Analog Resistive
29	Key size	Min 16 x 16 dots		
30	Number of simultaneous touch points	Max of 2 objects	Max 1 object	Maximum 1 object
31	Life	1 million times or more (operating force 0.98N max)	1 million presses or more (operating force 10…110 g max)	1 million presses or more (operating force 10…110 g max)
32	Battery backed-up data	Lithium battery (option) clock data and maintenance time notification data	Lithium battery (in-built) is used by the Real-time clock and static RAM	Lithium battery (in-built) is used by the Real-time clock and Static RAM;
33	Battery life	Approx. 5 years (operating ambient temp of 25 °C)	4 years (operating @ 25 °C)	4 years (Operating @ 25°C)
34	RS-232	RS-232, 1 ch; transmission speed: 115.2/57.6/38.4/ 19.2/9.6/4.8 Kbps; connector shape: D-sub 9-pin (male); application: for communicating with a controller or connecting a personal computer (project data upload/download, OS installation, transparent function)	RS-232, 1ch; transmission speed: 115.2/57.6/38.4/19.2/9.6/4.8 Kbps; connector shape: D-sub 9-pin (male); application: for communicating with a controller or connecting a personal computer (project data upload/download, OS installation, printing)	RS-232, 1ch; Transmission speed : 115,200/57,600/38,400/19,200/9,600/4,800 bps; Connector shape : D-sub 9-pin (Male); Application: For communicating with a controller or connecting a personal computer (Project data upload/download, OS installation, Printing)
35	USB	USB (Full speed 12 Mbps), 1ch; connector shape: mini-B; application: personal computer communication (screen data upload/download, OS installation and FA transparent function)	USB (full speed 12 Mbps), 2 ch for keyboard, mouse connectivity, barcode scanner, and printing	USB (Full Speed 12Mbps), 2ch for Keyboard, mouse connectivity, Barcode Scanner & printing
36	CF card	CompactFlash slot, 1ch; connector shape: TYPE I; application: data transfer, data storage	CompactFlash slot, 1ch; connector shape: TYPE I; application: data transfer, data storage	Compact flash slot, 1ch; Connector shape : TYPE I; Application : Data transfer, data storage
37	Scripting functions	Allow terminals the ability to run aux codes	Not possible	Activex Support - Open Function
38	Alarm and messaging	Advanced alarm and recipe management	Advanced alarm	Highly Advanced alarm, messaging and Recipe management
39	Trending and sampling	Graphical trending and data sampling	Graphical trending & data sampling/storage in local and networked drive	Graphical trending & data sampling/storage in local & networked drive
40	Multi language	Supports unicode 2.1 true type fonts (scalable)	Supports unicode 2.1 and run time language switching possible true type fonts	Supports Unicode 2.1 & Run Time Language Switching Possible True Type fonts (scalable)

S.No	Feature	GT15-75VNBA	PVP	PVP CE
41	Data sharing	FTP, email, client/server functions	Not possible	FTP, email, client/server functions, remote terminal server connectivity and control
42	Printing features	Printing and report generation	Alarm, diagnostics, printing option	Alarm, diagnostics, and open report generation/printing option
43	Maintenance	Tracks total hours terminal has been in operation	Terminal operation, temperature of operation, battery condition monitoring, 25+ start-up test utility with scheduled maintenance to check health of the terminal	Terminal operation, temperature of operation, battery condition monitoring, 25+ start-up test utility with scheduled maintenance to check health of the terminal
44	Weight, approx.	2.3 kg	2.6…2.9 kg	2.6…2.9 kg
45	Operating system	Proprietary	Proprietary	Open - Windows CE.NET
46	Additional programs that can be run	System monitor — Direct access to devices in CPU Ladder monitor — Direct access to CPU Logic Ladder Monitor function requires additional memory expansion board List editor — Modify logic without a personal computer (supports A/AnS-Series CPU) Motion monitor — Access to Q-Motion CPU (Q172/3CPUs) parameters/monitoring data through special pre-programmed screens Motion monitor — Access to Q-Motion CPU (Q172/3CPUs) parameters/monitoring data through special pre-programmed screens Network monitor — Direct access to CPU stations on Mitsubishi's high-speed networks (MNET10/II/B)	Not possible	– File viewers for MS Office suite: Excel, Word, PowerPoint software – PDF file viewer – WordPad text editor – Web server application – FTP server – Support for the .NET compact framework (anything like system monitor, ladder monitor, list editor, motion monitor, and network monitor can be supported)

Notes:

Notes:

Rockwell Automation Support

Rockwell Automation provides technical information on the Web to assist you in using its products. At http://support.rockwellautomation.com, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration, and troubleshooting, we offer TechConnect support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://support.rockwellautomation.com.

Installation Assistance

If you experience a problem within the first 24 hours of installation, please review the information that's contained in this manual. You can also contact a special Customer Support number for initial help in getting your product up and running.

United States	1.440.646.3434 Monday – Friday, 8am – 5pm EST
Outside United States	Please contact your local Rockwell Automation representative for any technical support issues.

New Product Satisfaction Return

Rockwell Automation tests all of its products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning and needs to be returned, follow these procedures.

United States	Contact your distributor. You must provide a Customer Support case number (call the phone number above to obtain one) to your distributor in order to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for the return procedure.

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication LOGIX-AP009A-EN-P - April 2009