Allen-Bradley

SoftLogix5800 System

1789-L10, 1789-L30, 1789-L60

User Manual

Rockwell Automation

Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we use notes to make you aware of safety considerations:

ATTENTION

Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss

Attention statements help you to:

- identify a hazard
- avoid a hazard
- recognize the consequences

IMPORTANT

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

Introduction

This version of the SoftLogix5800 System User Manual corresponds to version 13 of the controller. Revision bars (shown in the left margin of this page) indicate changed information. Changes made to this manual include:

For this information about:	See:
periodic save (persistent storage)	page 1-5
controlling I/O over an EtherNet/IP network	chapter 5
EtherNet/IP module LEDs	page C-2
Windows events	chapter 10

The SoftLogix controller also supports event tasks. See the *Logix5000 Controllers Common Procedures Manua*l, 1756-PM001.

Notes:

Purpose of this Manual

This manual guides the development of projects for SoftLogix5800 controllers. It provides procedures on how to establish communications and/or programming links over these networks:

- ControlNet
- DeviceNet
- EtherNet/IP
- serial

This manual works together with the *Logix5000 Controllers Common Procedures Programming Manual*, publication 1756-PM001, which covers the following tasks:

- Manage project files
- Organize your logic
- Organize tags
- Program routines
- Test a project
- Handle faults

How to Use This Manual

As much as possible, we organized this manual to explain, in a task-by-task manner, how to install, configure, program, operate and troubleshoot a SoftLogix control system.

Related Documentation

These core documents address the Logix5000 family of controllers:

If you are:	Use this publication:
a new user of a Logix5000 controller This quick start provides a visual, step-by-step overview of the basic steps you need to complete to get your controller configured and running.	Logix5000 Controllers Quick Start publication 1756-QS001
an experienced user of Logix5000 controllers This system reference provides a high-level listing of configuration information, controller features, and instructions (ladder relay, function block diagram, and structured text).	Logix5000 Controllers System Reference publication 1756-QR107
any user of a Logix5000 controller This common procedures manual explains the common features and functions of all Logix5000 controllers.	Logix5000 Controllers Common Procedures publication 1756-PM001

SoftLogix-specific information is also available:

For	Read this document	Document number
Information on installing a SoftLogix5800 controller	SoftLogix5800 Controller Installation Instructions	1789-IN001
Information on the Logix5000 controllers general instruction set	Logix5000 Controllers General Instruction Set Reference Manual	1756-RM003
Information on Logix5000 controller process control and drives instruction set	Logix5000 Controllers Process Control/Drives Instruction Set Reference Manual	1756-RM006
Information on Logix5000 controller motion instruction set	Logix5000 Controllers Motion Instruction Set Reference Manual	1756-RM007
Execution times and memory use for instructions	Logix5000 Controllers Execution Time and Memory Use Reference Manual	1756-RM087
Information on grounding and wiring Allen-Bradley programmable controllers.	Allen-Bradley Programmable Controller Grounding and Wiring Guidelines	1770-4.1

If you would like a manual, you can:

- download a free electronic version from the internet at **www.theautomationbookstore.com**
- purchase a printed manual by:
  - contacting your local distributor or Rockwell Automation representative
  - visiting **www.theautomationbookstore.com** and placing your order
  - calling 1.800.963.9548 (USA/Canada) or 001.330.725.1574 (Outside USA/Canada)

Chapter 1

What Is SoftLogix?

Chapter 2

Controlling Motion Devices

Chapter 3

Communicating with Devices on a DeviceNet Link

Chapter 4

Communicating with Devices on a ControlNet Link

Chapter 5

Communicating with Device on an Ethernet Link

Chapter 6

Communicating with Devices on a Serial Link

Chapter 7

Configuring and Using Simulated I/O

Chapter 8

Using External Routines

Chapter 9

Developing External Routines

Chapter 10

Programming Windows Events to Monitor and Change Controller Execution

Appendix A

Windows Considerations

Appendix B

System Performance Tuning Guidelines

Appendix C

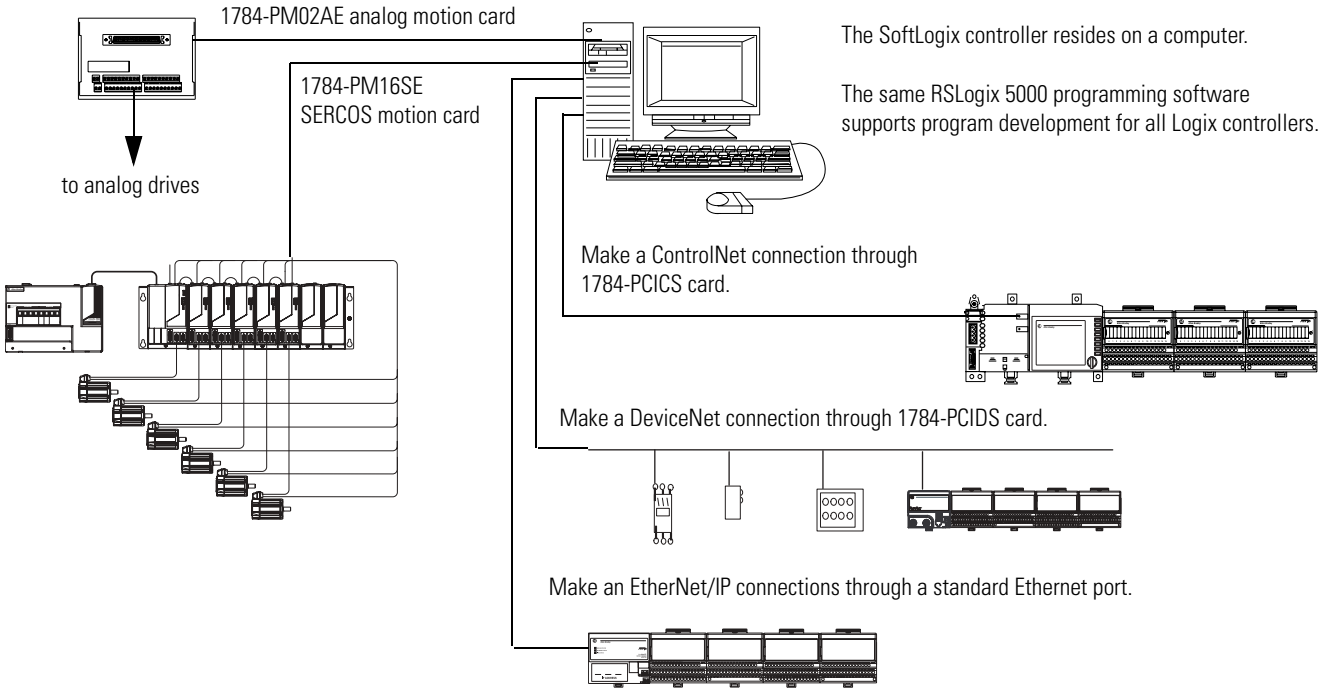
Monitoring Controller LEDs

What Is SoftLogix?

Using This Chapter

The SoftLogix controller is part of the Logix environment. The SoftLogix controller is a software-based controller that supports the Logix instructions, including the motion instructions. A SoftLogix system can consist of these components:

- RSLogix 5000 programming software that supports every Logix controller. Program (on-line or off-line) in ladder logic, function block diagram, structured text, and sequential function chart.
- 1784-PM16SE and 1784-PM02AE motion cards provide integrated motion control
- 1784-PCICS communication card that provides communication and I/O control over a ControlNet network or a 1784-PCIC communication card that provides **only** communication over a ControlNet network
- 1784-PCIDS communication card that provides communication and I/O control over a DeviceNet network
- a commercially-available Ethernet port for messaging and controlling I/O over an EtherNet/IP network.



Select the SoftLogix5800 product that best fits your application:

If you need (maximum):	Use this controller:	Available slots:
1 SoftLogix5800 controller memory size limit of 2 Mbytes per controller[1] 2 PCI network interface cards, which can be a mix of: • one 1784-PCICS • one 1784-PCIC • one 1784-PCIDS • one EtherNet/IP card[2] no motion support 1 1784-SIM module no third party virtual backplane module support	1789-L10	3-slot virtual chassis[3]
2 SoftLogix5800 controllers memory size limit of 64 Mbytes per controller 5 PCI network interface cards[4] 2 1784-PM02AE analog motion cards 1 1784-PM16SE SERCOS motion card 5 1784-SIM modules EtherNet/IP support third party virtual backplane module support	1789-L30	5-slot virtual chassis
6 SoftLogix5800 controllers memory size limit of 64 Mbytes per controller 16 PCI network interface cards[4] 4 1784-PM02AE analog motion cards 4 1784-PM16SE SERCOS motion card 16 1784-SIM modules EtherNet/IP support third party virtual backplane module support	1789-L60	16-slot virtual chassis

(1) Even though the 1789-L10 controller supports two PCI network interface cards, each card must be a different network card. You cannot have two of the same cards installed in the computer.

(2) EtherNet/IP via PCI bus card or embedded EtherNet/IP port on the PC motherboard.

(3) As of firmware revision 12, the 1789-L10 controller now supports 3 slots.

(4) The number of available slots in the virtual chassis is limited by activation level. You can have as many PCI communication cards as you have PCI slots or as you have slots in the virtual chassis, whichever number is smallest.

Most SoftLogix applications run additional software on the same PC as the controller. Make sure the computer meets these requirements:

- IBM-compatible Pentium 4 1.6 GHz
- 256 KBytes of RAM
- 50 Mbytes hard disk space

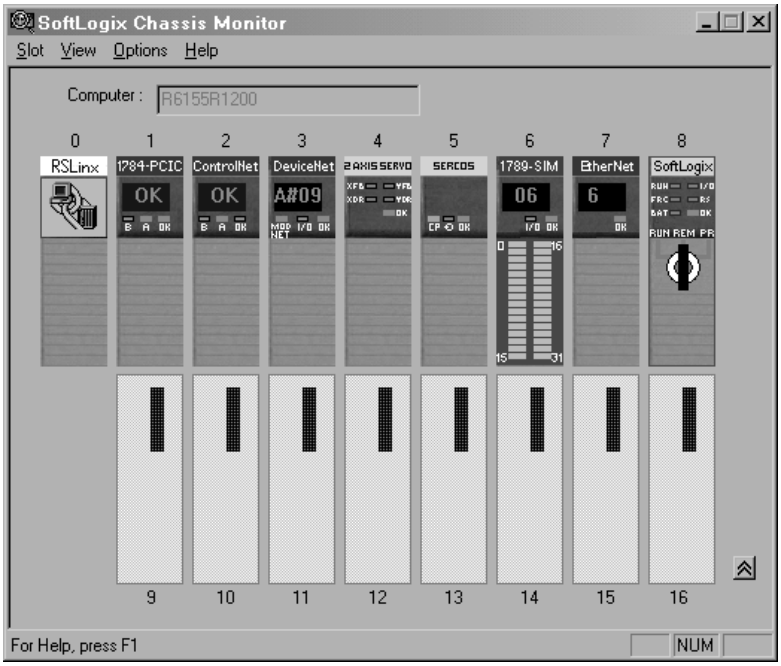
Demanding applications including sequential, motion, and other local applications running on the PC may require a dual CPU to achieve performance requirements.

IMPORTANT The same SoftLogix5800 controller is supplied in all of the above products. Regardless of the product you have, select 1789-L60/A in RSLogix 5000 software when you specify a controller type.

For information about:	See page
using the chassis monitor	1-3
developing programs	1-7
how the SoftLogix system uses connections	1-11
selecting a system overhead percentage	1-17

Using the Chassis Monitor

The Chassis Monitor is your window into the SoftLogix system so you can monitor the system components. The Chassis Monitor models a chassis. You install virtual devices in the virtual chassis to represent the controller and cards in your system.

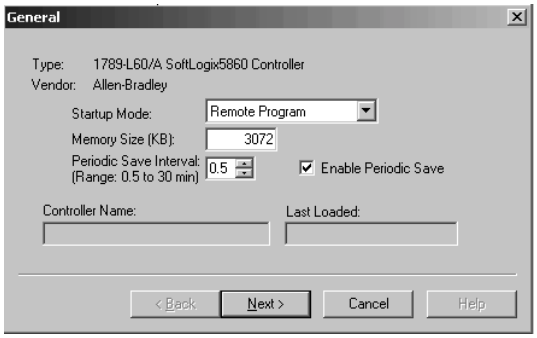
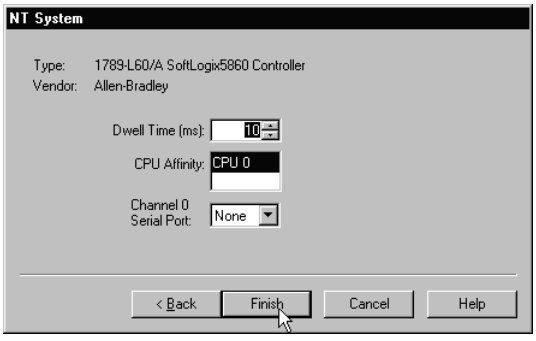


IMPORTANT Treat the computer running a SoftLogix controller like an industrial controller and not a PC. A PC can perform many operations that are incompatible with the real-time operations required by a SoftLogix controller.

The Chassis Monitor is your interface to the SoftLogix controller. Use the monitor to:

- add and configure controllers
- add and configure communication cards
- add and configure motion cards
- change processor mode
- monitor controller and associated module status
- monitor motion performance

When you install a controller, the Chassis Monitor lets you configure specific characteristics about the controller:

On this configuration dialog box:	Specify these characteristics:	
	Startup Mode	Specify how the controller should behave when its service is started. Select Remote Program (default) or Last Controller State
	Memory Size	Specify the memory size (Kbytes) to allow for the controller. The limit is the amount of RAM in your computer. The default is 3072 Kbytes. See the information on page 1-5 about determining an appropriate memory size.
	Periodic Save Interval	Specify whether you want to save the current controller information (tag data values and configuration information) periodically, and if so, specify how often (minutes). Specify an interval between 0.5 and 30 minutes. Online edits to the program are saved instantly, regardless of Periodic Save interval. The default is enabled for 10 minutes. See the information on page 1-5 about this setting's impact on overall system performance.
	Continuous Task Dwell Time (ms)	Specify the dwell time (0-1000 ms) made available for all other Windows applications. The default is 10 ms. The dwell time is the time between the end of the continuous task and the start of the next execution of the continuous task. This setting has an impact on overall system performance, see Appendix B.
	CPU Affinity	If your computer has multiple Pentium CPUs, select which CPU to use for this controller. The default is CPU 0.
	Channel 0 Serial Port	Select which COM port to use for serial communications. Select COM1, COM2, COM3, or COM4. The default is none.

Determining a memory size

IMPORTANT The memory size you specify is the amount of RAM in your computer that you want to allocate to the SoftLogix controller. This allocated RAM is not available to Windows NT or any other application.

The following equations provide an estimate of the memory needed for a controller. Each of these numbers includes a rough estimate of the associated user programming. Depending on the complexity of your application, you might need additional memory.

Controller tasks	_____	* 4000 =	_____	bytes (minimum 1 needed)
Discrete I/O points	_____	* 400 =	_____	bytes
Analog I/O points	_____	* 2600 =	_____	bytes
Communication modules	_____	* 2000 =	_____	bytes
Motion axis	_____	* 8000 =	_____	bytes
		Total =	_____	bytes

If you want to change the amount of memory you specified for a controller, you must first remove the controller from the SoftLogix chassis monitor. Then re-install the controller and specify the new memory size.

Specifying a periodic save interval

The periodic save task executes at a priority of “user-mode high”. This means that the control process running within the SoftLogix5800 controller **WILL NOT** be impacted by a periodic save, but other user applications **WILL** be impacted if they run at a priority lower than “user-mode high”. Most HMI applications run at a “user-mode normal” priority. If these applications run on the same computer as the SoftLogix5800 controller, these applications will be starved of CPU cycles while the periodic save is in progress. If you run an HMI application remotely and gather data from a SoftLogix5800 controller via OPC, the performance of the HMI may also be impacted during a periodic save. The controller handles both the periodic save “tag value upload” and HMI OPC requests through the same communications mechanism.

When the periodic save task executes, it performs these actions:

1. For every tag defined within the controller, the current tag value is read from the controller.

   The larger the amount of data, the longer the periodic save takes and the greater the impact on HMI responsiveness.

2. The current tag values read in step 1, along with the current program file are saved to the computer disk drive.

   The larger the archive file, the longer the periodic saves takes and the greater the impact on HMI responsiveness. However, tag data size has more of an impact than archive file size.

To maintain better HMI responsiveness, you can:

- Turn off the periodic save interval

  Even with the periodic save interval disabled, a periodic save occurs if a remote terminal performs an upload. This ensures that the most current tag data values and archive file are uploaded.

  If you disable the periodic save, you can still initiate a save manually by using the Save menu item on the controller from the chassis monitor or programmatically from an external routine or application (see page 10-9).

- Increase the periodic save interval so that it occurs less frequently.
- Use a dual CPU computer.

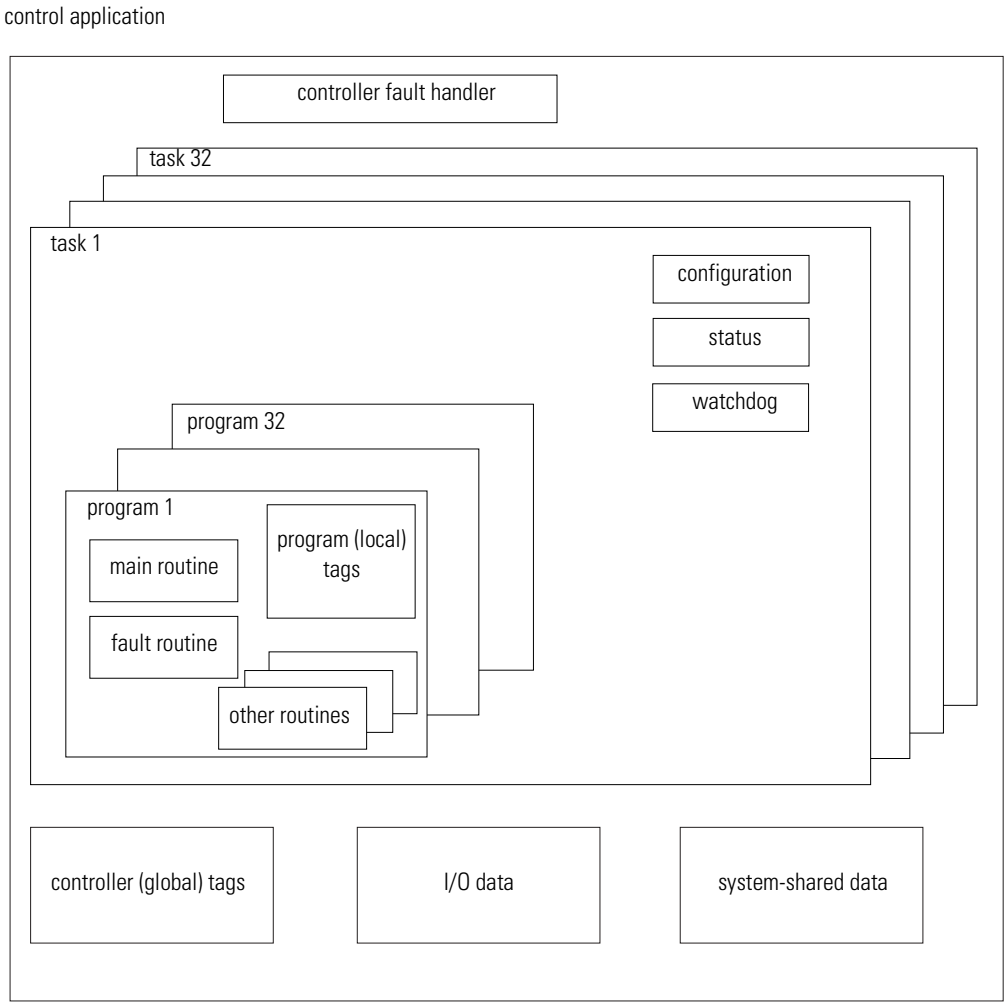
  On a dual CPU computer, the Windows operating system automatically balances the periodic save and HMI applications across the CPUs.

For more information on system tuning and the periodic save interval, see Appendix B.

Developing Programs

The controller's execution model is a preemptive multitasking system that is IEC 1131-3 compliant. This environment provides:

- tasks to configure controller execution
- programs to group data and logic
- routines to encapsulate executable code written in a single programming language



Defining tasks

A task provides scheduling and priority information for a set of one or more programs. You can configure tasks as either continuous or periodic. The SoftLogix controller supports as many as 32 tasks, only one of which can be continuous.

A task can have as many as 32 separate programs, each with its own executable routines and program-scoped tags. Once a task is triggered (activated), all the programs assigned to the task execute in the order in which they are grouped. Programs can only appear once in the Controller Organizer and cannot be shared by multiple tasks.

Specifying task priorities

Each task in the controller has a priority level. The controller uses the priority level to determine which task to execute when multiple tasks are triggered. There are 3 configurable priority levels for periodic tasks that range from 1-3, with 1 being the highest priority and 3 being the lowest priority. A higher priority task will interrupt any lower priority task. The continuous task has the lowest priority and is always interrupted by any periodic task.

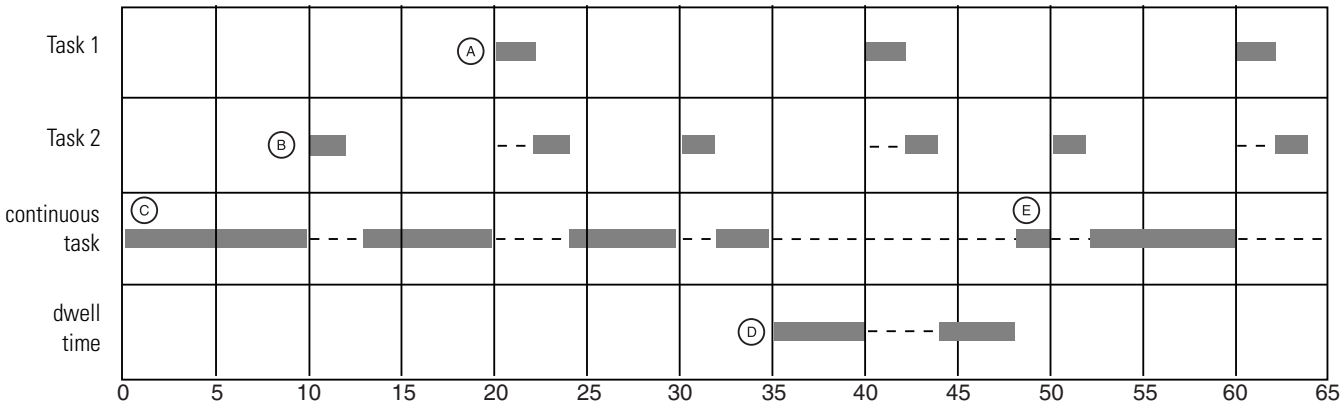
The continuous task dwell time determines how much time to allow for other Windows programs, running at a normal priority, to execute. The dwell time is the time between the end of the continuous task and the start of the next execution of the continuous task. The dwell time does not affect periodic tasks. Periodic tasks execute as scheduled regardless of the dwell time. By default, the dwell time is 10ms. This setting has an impact on overall system performance, see Appendix B.

Configuring tasks based on other events

The SoftLogix controller supports an additional Windows event trigger. This trigger lets you monitor Windows events in Windows 2000 or Windows XP operating systems so that applications outside of the SoftLogix controller can cause a task within the SoftLogix controller to execute. For more information, see Chapter 10.

The following example shows the task execution order for an application with periodic tasks and a continuous task.

Task:	Priority Level:	Task Type:	Actual Execution Time:	Worst Case Execution Time:
1	1	20ms periodic task	2ms	2ms
2	2	10ms periodic task	4ms	6ms
na	none (lowest)	continuous task	25ms	35ms
na	none	dwell time	10ms	14ms



Notes:

A. The highest priority task interrupts all lower priority tasks.

B. A lower priority task can be interrupted multiple times by a higher priority task.

C. The continuous task runs at the lowest priority and is interrupted by all other tasks.

D. When the continuous task completes, the dwell time starts. The dwell time does not affect periodic tasks. Periodic tasks execute as scheduled regardless of the dwell time.

E. When the dwell time completes, the continuous tasks restarts, unless a higher priority task is running.

Defining programs

Each program contains program tags, a main executable routine, other routines, and an optional fault routine. Each task can schedule as many as 32 programs.

The scheduled programs within a task execute to completion from first to last. Programs that aren't attached to any task show up as unscheduled programs. You must specify (schedule) a program within a task before the controller can scan the program.

Defining routines

A routine is a set of logic instructions in a single programming language, such as ladder logic. Routines provide the executable code for the project in a controller. A routine is similar to a program file or subroutine in a PLC or SLC processor.

Each program has a main routine. This is the first routine to execute when the controller triggers the associated task and calls the associated program. Use logic, such as the JSR instruction, to call other routines.

You can also specify an optional program fault routine. The controller executes this routine if it encounters an instruction-execution fault within any of the routines in the associated program.

The SoftLogix controller supports routines developed with the relay ladder and function block editors of RSLogix 5000 programming software. You can edit relay ladder and function block routines either offline or online. You can also develop C/C++ routines and incorporate them into your project. See chapter 8 for information on adding external routines to a project; see chapter 9 for information on developing external routines.

Instruction execution

When performing math operation, the SoftLogix controller handles INT to REAL conversions slightly different than hardware-based Logix controllers. The SoftLogix controller completes the math operation using the INT data and then converts the result to REAL data, which is more consistent with how math operations occur on personal computers. The hardware-based Logix controllers first convert INT data to REAL data and then perform the math operation.

How the SoftLogix System Uses Connections

The SoftLogix system uses a connection to establish a communication link between two devices. Connections can be:

- controller to local I/O modules or local communication modules
- controller to remote I/O or remote communication modules
- controller to remote I/O (rack optimized) modules
- produced and consumed tags
- messages

You indirectly determine the number of connections the controller uses by configuring the controller to communicate with other devices in the system. Connections are allocations of resources that provide more reliable communications between devices than unconnected messages. The SoftLogix system supports both scheduled and unscheduled connections and unconnected messages.

Method:	Description:
scheduled connection • highest level of determinism • unique to ControlNet	A scheduled connection is unique to ControlNet communications. A scheduled connection lets you send and receive data repeatedly at a predetermined rate, which is the requested packet interval (RPI). For example, a connection to an I/O module is a scheduled connection because you repeatedly receive data from the module at a specified rate. Other scheduled connections include connections to: • communication devices • produced/consumed tags On a ControlNet network, you must use RSNetWorx for ControlNet to enable all scheduled connections and establish a network update time (NUT).
unscheduled connection • deterministic • used by both ControlNet and EtherNet/IP	An unscheduled connection is a message transfer between controllers that is triggered by the requested packet interval (RPI) or the program (such as a MSG instruction). Unscheduled messaging lets you send and receive data when needed. All EtherNet/IP connections are unscheduled.
unconnected message • least deterministic	An unconnected message is a message that does not require connection resources. An unconnected message is sent as a single request/response.

Each 1784-PCICS ControlNet communication card supports 128 total connections, 127 of which can be scheduled connections. How you configure these connections depends on how many cards are in the controller. The controller supports a total of 250 connections.

Determining Connections for Produced and Consumed Tags

The SoftLogix controller supports the ability to produce (multicast) and consume (receive) system-shared tags. System-shared data is accessible by multiple controllers over a ControlNet or EtherNet/IP network. Produced and consumed tags each require scheduled connections.

This type of tag:	Requires these connections:
produced	By default, a produced tag allows two other controllers to consume the tag, which means that as many as two controllers can simultaneously receive the tag data. The local controller (producing) must have one connection for the produced tag and the first consumer and one more connection for each additional consumer (heartbeat). The default produced tag requires two connections. As you increase the number of controllers that can consume a produced tag, you also reduce the number of connections the controller has available for other operations, like communications and I/O.
consumed	Each consumed tag requires one connection for the controller that is consuming the tag.

The maximum number of produced/consumed tags you can configure depends on the connection limits of the communication device that transfers the produced/consumed data

For two controllers to share produced or consumed tags, both controllers must be attached to the same network. You cannot bridge produced and consumed tags between two networks.

Determining Connections for Messages

Messages transfer data to other devices, such as other controllers or operator interfaces. Some messages use unscheduled connections to send or receive data. These connected messages can leave the connection open (cache) or close the connection when the message is done transmitting. The following table shows which messages use a connection and whether or not you can cache the connection:

This type of message:	Using this communication method:	Uses a connection:
CIP data table read or write	CIP	✓
PLC2, PLC3, PLC5, or SLC (all types)	CIP	
	CIP with Source ID	
	DH+	✓
CIP generic	N/A	✓

Connected messages are unscheduled connections on both ControlNet and EtherNet/IP networks.

If a message executes repeatedly, cache the connection. This keeps the connection open and optimizes execution time. Opening a connection each time the message executes increases execution time.

If a message executes infrequently, do not cache the connection. This closes the connection upon completion of the message, which frees up that connection for other uses.

Each message uses one connection, regardless of how many devices are in the message path. To conserve connections, you can configure one message to read from or write to multiple devices.

You can cache as many as 16 messages (a combination of any type, not including block-transfer) at one time. If you try to cache more than 16, the controller determines the 16 most-currently used messages and caches those. If there are 16 messages cached, and a message is triggered that is currently not cached, the controller drops the connection of the oldest-cached message to make room for the new message.

In addition to 16 cached messages, you can also cache as many as 16 block-transfer messages. The same conditions apply to caching block-transfer messages as described above for caching other types of messages.

Determining Connections for I/O Modules

The SoftLogix system uses connections to transmit I/O data. These connections can either be direct connections or rack-optimized connections.

Connection:	Description:
direct	A direct connection is a real-time, data transfer link between the controller and an I/O module. The controller maintains and monitors the connection between the controller and the I/O module. Any break in the connection, such as a module fault or the removal of a module while under power, causes the controller to set fault status bits in the data area associated with the module.
rack-optimized	For digital I/O modules, you can select rack optimized communication. A rack optimized connection consolidates connection usage between the controller and all the digital I/O modules on a rack (or DIN rail). Rather than having individual, direct connections for each I/O module, there is one connection for the entire rack (or DIN rail).

Depending on the type of I/O modules, both direct connections and rack-optimized connections can be used. The following table lists several of the I/O systems and the available connections types.

I/O System:	Supported Connection Type(s):
digital ControlNet I/O	direct connection or rack-optimized connection(1)
analog ControlNet I/O	direct connection
digital EtherNet/IP I/O	direct connection or rack-optimized connection(1)
analog EtherNet/IP I/O	direct connection
DeviceNet I/O	rack-optimized connection

(1) Rack-optimized connections for diagnostic and E-fuse modules do not send diagnostic or fuse data to controller.

To conserve the number of connections that are available, place digital I/O modules together in the same location and use a rack-optimized connection. To select a rack-optimized connection, select a “rack-optimized” option for the communication format when you add the communication device and I/O modules to the controller project in RSLogix 5000 programming software.

If you have analog I/O modules, or want a direct connection to specific I/O modules, you do not have to create the rack-optimized connection to the communication device. To use direct connections to I/O modules, select “none” for the communication format of the communication device.

Determining Total Connection Requirements

The SoftLogix controller supports 250 connections. Each 1784-PCICS ControlNet communication card supports 128 total connections, 127 of which can be scheduled. Do not configure more connections than the controller can support. Use the following table to tally ControlNet connections:

Connection Type:	Device Quantity:	Connections per Device:	Total Connections:
1784-PCICS communication card		0	0
remote ControlNet communication device (such as a 1794-ACN15, -ACNR15 or 1756-CNB module) configured as a direct (none) connection configured as a rack-optimized connection		 0 **or** 1	
remote I/O device over ControlNet (direct connection)		1	
remote EtherNet/IP communication device (such as a 1794-AENT or 1756-ENBT module) configured as a direct (none) connection configured as a rack-optimized connection		 0 **or** 1	
remote I/O device over EtherNet/IP (direct connection)		1	
produced and consumed tag produced tag and one consumer each additional consumer		 1	
		1	
consumed tag		1	
cached message		1	
		total	

The SoftLogix controller also uses connections for DeviceNet devices and motion devices. Use this table to tally other connections:

Connection Type:	Device Quantity:	Connections per Device:	Total Connections:
1784-PCIDS communication card		2	
remote I/O device over DeviceNet (accounted for in rack-optimized connection for communication card)		0	0
1784-PM16SE SERCOS motion card		3	
1784-PM02AE analog motion card		3	
		total:	

Restarting the Controller

You restart the controller by:

- rebooting the computer

  or

- removing and re-inserting the controller in the virtual chassis

After restarting the controller, you must upload or download from RSLogix 5000 software before you can go online with the controller.

This is because the RSLogix 5000 project file (.ACD) contains explicit knowledge of the physical memory addresses used by the controller. When you restart the controller, all the physical addresses for the controller are regenerated. Note that as long as the controller is not restarted, you can go online and offline as many times as required.

Going online with the controller

You must save the RSLogix 5000 project after a download completes or you will not be able to go online with the controller. After downloading, the physical address information has changed. RSLogix 5000 software prompts you to save and indicates that a change has occurred even though you might not have made changes to the project. Saving the project stores the physical address information into the .ACD file.

An upload recovers all the information that was downloaded to the controller, including documentation. This is because of the persistent storage feature that you enable by specifying a periodic save interval (see page 1-4). On a download, the persistent storage copies the entire project file to the controller. The controller opens and goes online with the project file so that any edits made by RSLogix 5000 workstation(s) are saved into the persistent image (the controller's copy of the project file). Online edits are saved to the persistent image immediately; tag data values are saved to the persistent image at every periodic save interval (10 minute default). If the periodic save is disabled, tag data values are not saved, but online edits are still saved to the persistent image.

The SoftLogix controller maintains a change log that holds 999 entries. This means that as you edit an RSLogix 5000 project file, you must save the project file before you make 999 changes. If you make more than 999 changes to a project, you will not be able to go back online without performing an upload or a download.

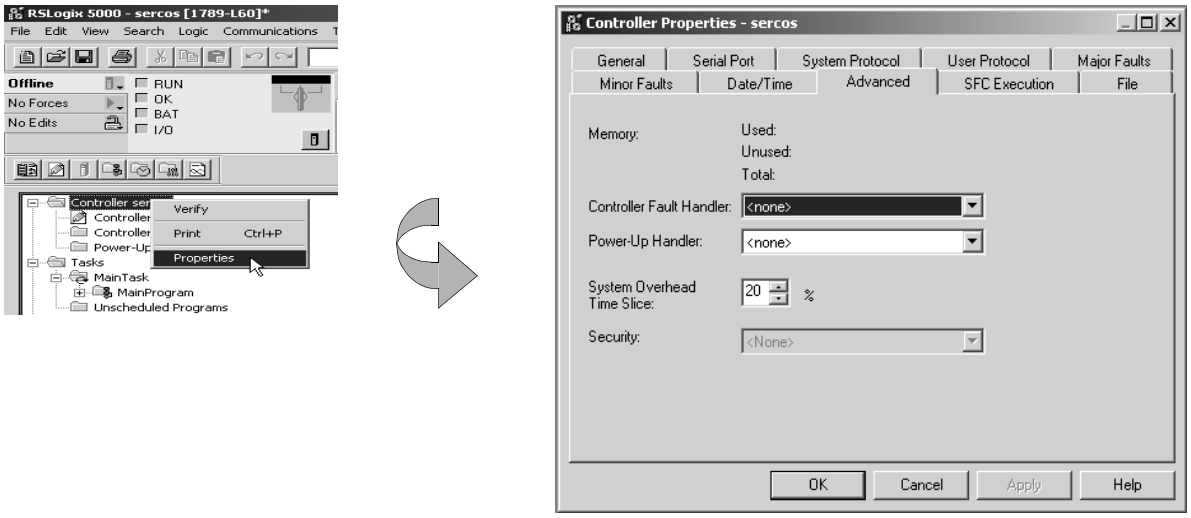
Uploading to the controller

If your project has edits and you want to upload the project to the controller, RSLogix 5000 software prompts you to save the project before uploading. Whether you select yes or no to save the project, the edits are saved before the upload occurs. This happens because the edits are already stored in the controller as you make the edits.

Selecting a System Overhead Percentage

The Controller Properties lets you specify a percentage for system overhead. This percentage specifies the percentage of controller time (excluding the time for periodic tasks) that is devoted to communication and background functions.

1. View properties for the controller and select the Advanced tab.



The system overhead function interrupts the continuous task. The percentage you specify determines the amount of the continuous task to allocate to system overhead functions, which include:

- communicating with programming and HMI devices (such as RSLogix 5000 software)
- responding to messages
- sending messages, including block-transfers
- re-establishing and monitoring I/O connections (such as RIUP conditions); this *does not* include normal I/O communications that occur during program execution
- bridging communications from a one communication device to another communication device across the virtual chassis

This function lets the controller take care of communication requests that occur from other controllers or from queued requests from within the controller's application program. If communications are not completing fast enough, increase the system overhead percentage.

Due to the fact that SoftLogix controller runs natively on your computer's Pentium CPU, the default setting of 10% yields satisfactory performance for most applications.

Notes:

Controlling Motion Devices

Using This Chapter

Integrating motion

The component of motion that most directly effects performance in the SoftLogix controller is the coarse update setting. The coarse rate is the periodic rate (at which the motion task executes) to compute the servo commanded position, velocity, and acceleration values to be sent to the output modules when executing motion instructions.

In a ControlLogix system, to improve motion performance, you would typically use a dedicated ControlLogix CPU for each motion module. In a SoftLogix system, adding more controllers actually decreases system performance. If you use multiple controllers, upgrade your computer hardware with a faster CPU or increase your coarse update period.

Configuring Your System for a Motion Card

For the SoftLogix controller to control motion applications, you need:

1784-PM16SE SERCOS motion card:	1784-PM02A analog motion:
You need: • a 1784-PM16SE motion card (4 per computer maximum) • RSLogix 5000 programming software to configure the motion card and its associated axes (16 per card) **1784-PM16SE requirements:** • maximum of four 1784-PM16SE cards per computer • can associate only one 1784-PM16SE card with one controller	You need: • a 1784-PM02AE motion card (4 per computer maximum) • a 1784-PMCSY4 synchronization cable If you have multiple 1784-PM02AE motion cards, you must link the cards with a 1784-PMCSY4 synchronization cable. • a 1784-PM02AE-TP01 or 1784-PM02AE-TP03 termination panel You make all field terminations to the motion card using the 1784-PM02AE-TP01 or 1784-PM02AE-TP03 termination panel and associated cable. • RSLogix 5000 programming software to configure the 1784-PM02AE motion card and its associated axes (2 per card) **1784-PM02AE requirements:** • maximum of four 1784-PM02AE cards per computer • maximum of four 1784-PM02AE cards can be associated with one controller • cannot associate a 1784-PM02AE motion card with the same controller as a 1784-PM16SE card

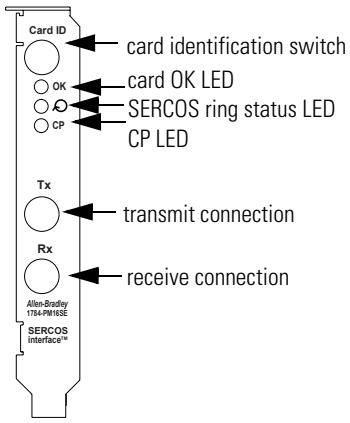
IMPORTANT **Do not** mix 1784-PM16SE and 1784-PM02AE cards in the same motion group. RSLogix 5000 software does not prevent you from mixing the cards in the same group. If you mix 1784-PM16SE and 1784-PM02AE cards in the same group, the motion group will never synchronize and error 19 “Group Not Synchronized” occurs when you try to execute a MAH instruction.

The latest drivers for various items like video and networking devices may be required for satisfactory system operation

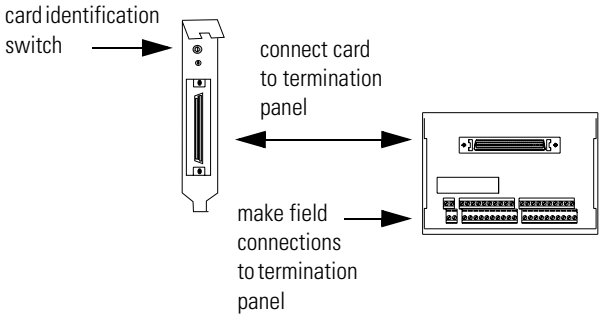
Step 1: Install the hardware

Make sure the motion card is properly installed in a 32-bit, primary PCI slot in the computer.

1784-PM16SE SERCOS motion card:



1784-PM02A analog motion:



If you have more than one 1784-PM02AE card, use the 1784-PMCSY4 synchronization cable to connect the motion cards within the computer.

- Use the card identification switch to identify each motion card in your computer. The card identification switch is a slotted, rotary switch with 16 positions (0-9 and A-F). Use a slotted screwdriver to select a setting.
- The switch setting uniquely identifies the motion card from any other similar-type motion cards in your computer. (A 1784-PM16SE card can have the same switch setting as a 1784-PM02AE card without creating a problem.) The switch setting and the PCI slot where you install the card **do not** correspond to the backplane slot in the SoftLogix chassis. You use the SoftLogix chassis monitor to place the communication card in a specific backplane slot (see the next page).
- Make a label to place on the mounting bracket of the card, or use a pen to write on the mounting bracket of the card. The label should include the card identification switch setting and a name you can use to identify the card from any others you might install in the computer.

For more information about installing a 1784-PM16SE motion card, see the *16-Axis Servo Card Installation Instructions*, publication 1784-IN041.

For more information about installing a 1784-PM02AE motion card, see the *2-Axis Servo Card Installation Instructions*, publication 1784-IN005.

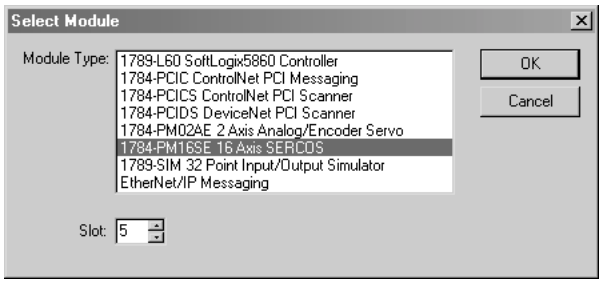
Step 2: Create the motion card in the chassis

Before you can operate the motion card, you must create the motion card as part of the SoftLogix chassis. This example shows creating a 1784-PM16SE SERCOS motion card.

1. From the SoftLogix chassis monitor, select Slot → Create Module or right click the appropriate slot and select Create. Select the motion card.

Specify the backplane slot number.

Click OK

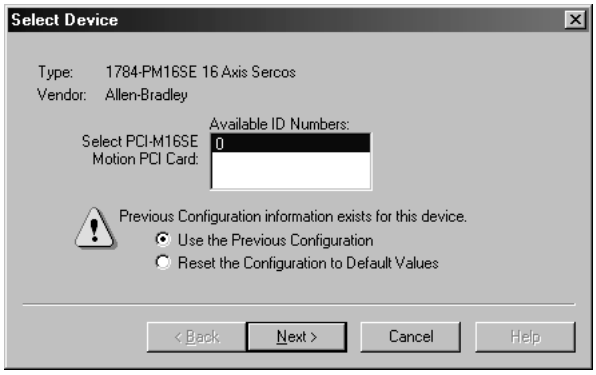


2. Specify which motion card to use by selecting an available ID number, which corresponds to the setting on the card identification switch.

Select the ID number.

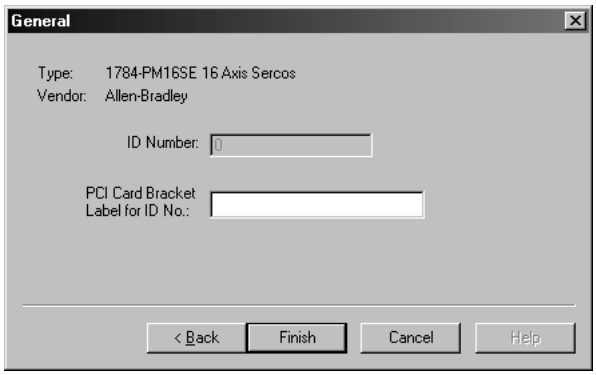
If you previously configured the 1784-PM16SE card that you selected by ID number, the chassis monitor remembers the configuration from the last time you used the card (whether in the same or different slot).

Click Next



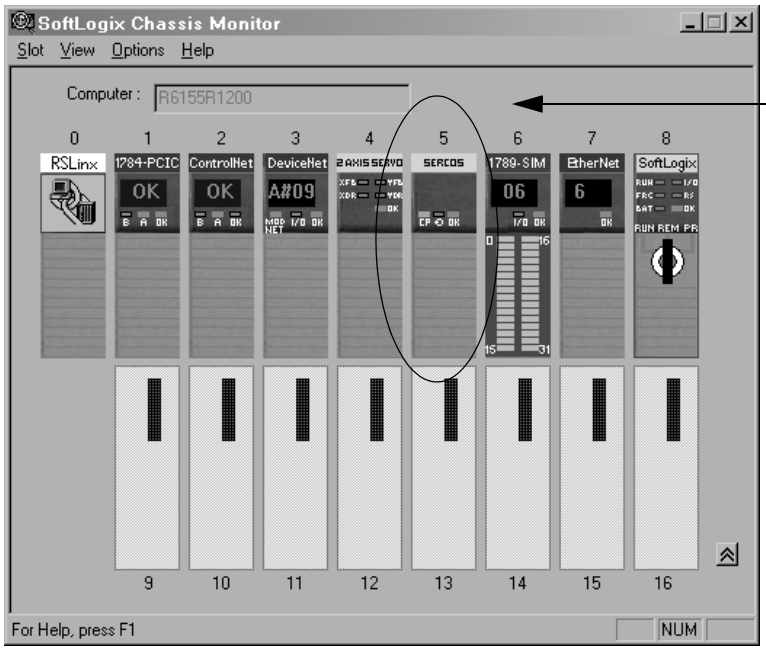
3. Enter the label name for the card (this is the name you wrote on the label of the card to help you identify the card from others in the same computer).

Click Finish



You can specify any slot number greater than 0 for the motion card. RSLinx software resides in slot 0.

The chassis monitor shows the 1784-PM16SE card as a virtual module in the SoftLogix chassis. The LEDs on the virtual monitor emulate the LEDs on the front of the similar ControlLogix motion module.



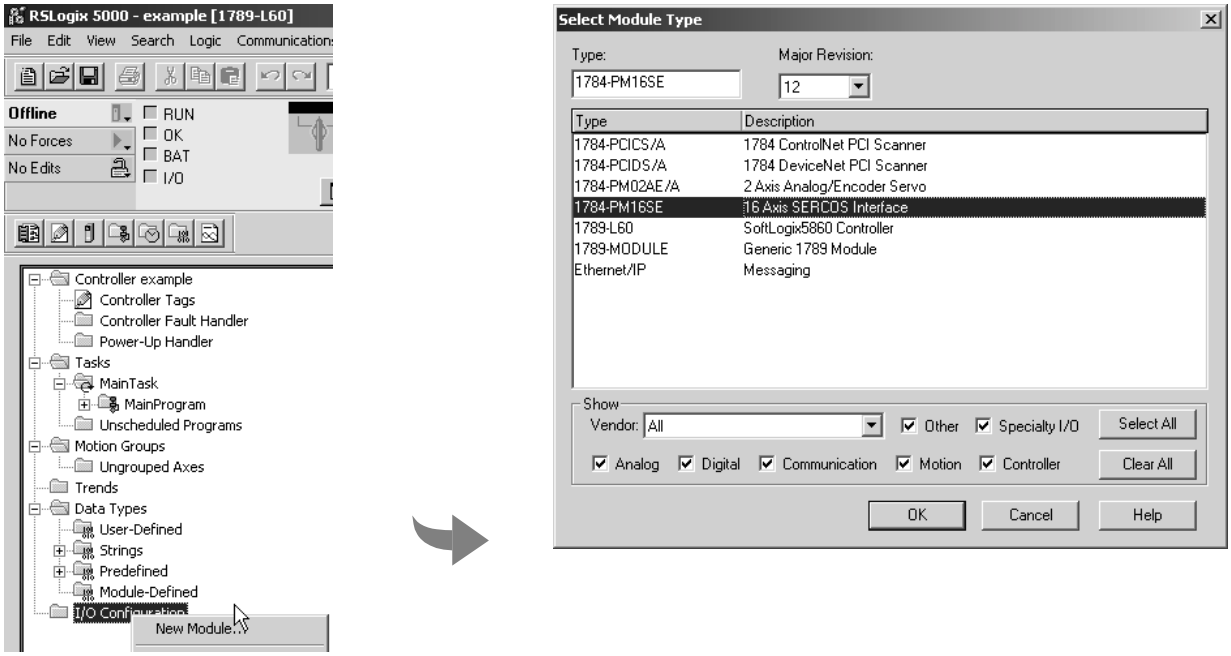
This chassis monitor has a 1784-PM16SE card installed in slot 5.

Step 3: Configure the motion card as part of the project

Use RSLogix 5000 programming software to map the motion card as part of the SoftLogix project. In the Controller Organizer, add the card to the I/O Configuration folder. This example shows configuring a 1784-PM16SE SERCOS motion card.

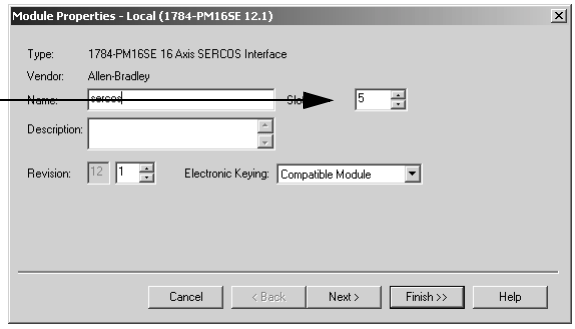
1. In RSLogix 5000 programming software, select the I/O Configuration folder.

2 Right-click to select New Module and add a 1784-PM16SE motion card.



3. Specify the appropriate motion card settings.

This must be the same slot number you specified on the SoftLogix chassis monitor.



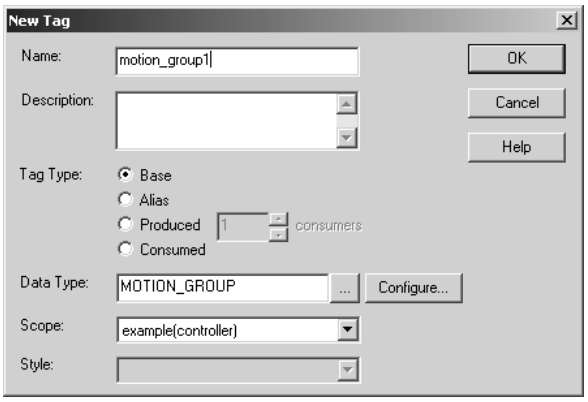
For information about motion groups and axes for:	See page
SERCOS motion card	2-7
analog motion card	2-14

Creating a SERCOS Motion Group and Axis

To configure axes for a SERCOS motion card, you must first create the motion group.

1. In the controller organizer, right-click **Motion Groups** and select **New Motion Group**.

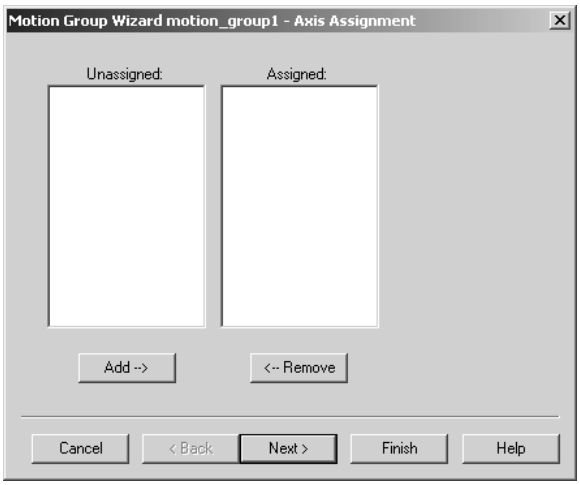
2. Define the new motion group.



In this field:	Type:
Name	The name of the group.
Description	A description of the group (optional).

3. Click **Configure** to specify the axes for the motion group.

   If you have already created unassigned axes, assign them to the motion group. Otherwise, you can assign axes later by modifying the properties of the motion group.

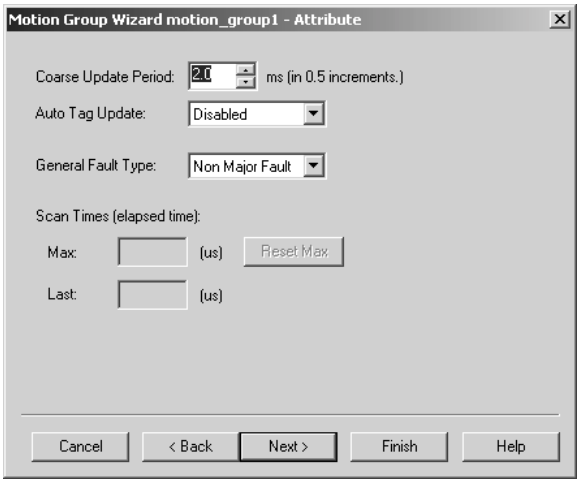


This field:	Lists:
Unassigned	The axes that are not assigned to any group in the controller.
Assigned	The axes that are assigned to this motion group.

   Highlight an axis in the Unassigned window and click **Add** to move it into the Assigned window.

4. Click **Next**.

5. Define the attributes of the motion group.

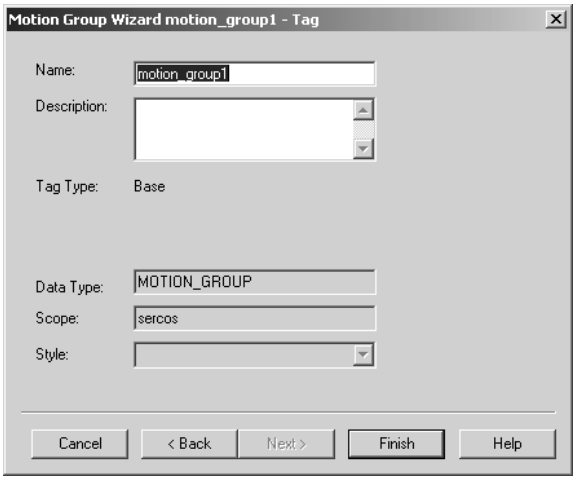


In this field:	Type:
Coarse Update Period	Select the periodic rate at which the motion task executes to compute the servo commanded position, velocity, and accelerations to be sent to the motion card when executing motion instructions. If the coarse update period is too small, the controller may not have time to execute non-motion related ladder logic. The configuration sets the lower limit on the coarse update period based on the number of axes in the group.
Auto Tag Update	Determines whether axis parameter values will be automatically updated during operation. Choose from: • Enabled – turns On automatic tag updating • Disabled – turns Off automatic tag updating
General Fault Type	Select the general fault type for the motion group: • Non Major Fault – Any faults detected by the motion group will not cause the processor to fault. The application programmer needs to handle the fault in the program. • Major Fault – Any faults detected by the motion group will cause the processor OK light to go blinking red and the fault routine to be invoked. If the fault routine handles the fault and clears it, then the OK light turns green. If the fault routine does not clear the fault, then the OK light becomes solid red and the processor stops executing the program.

6. Click **Next**.

7. Define the tag for the motion group.

   The tag name defaults to the group name.



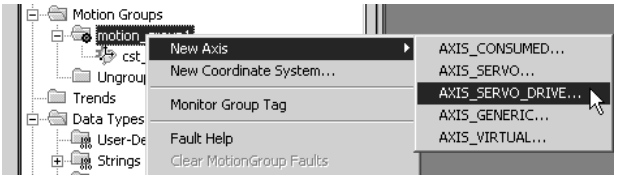
In this field:	Type:
Name	The name of the group.
Description	A description of the group (optional).

8. Click **Finish**.

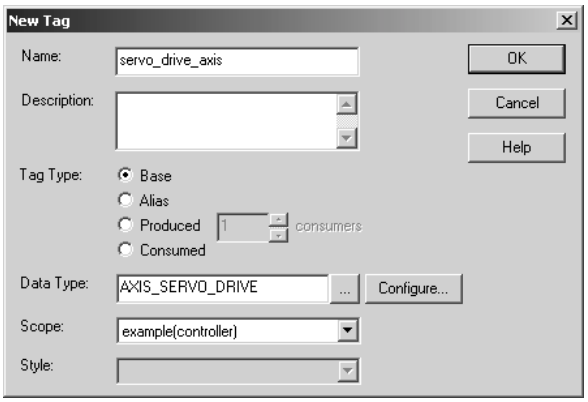
Creating an axis

You can create an axis directly assigned to a motion group or you can create an axis that is unassigned and later you can assign it to a motion group. To create an axis:

1. In the controller organizer, right-click the motion group (or you can right-click the Ungrouped Axes folder) and select **New Axis**. Select the type of axis you want to create.



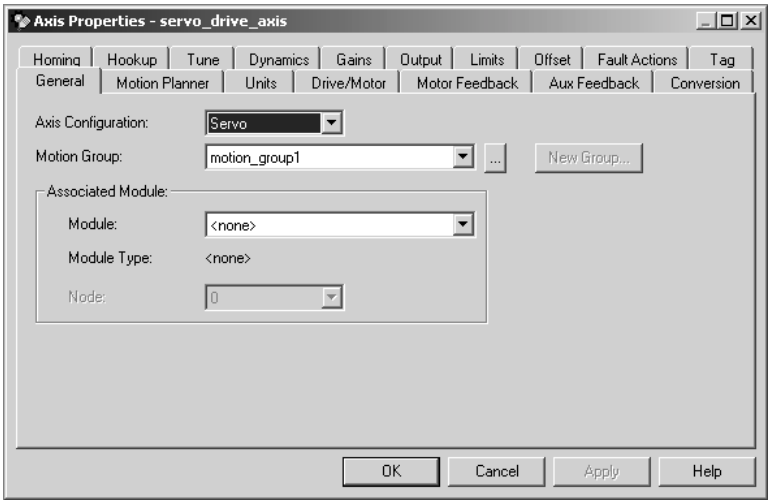
2. Define the axis.



In this field:	Type:
Name	The name of the axis.
Description	A description of the axis (optional).

3. Click **Configure** to display the Axis Wizard. Click **Finish** when done.

   You can also configure the axis by right-clicking on the axis and selecting **Properties**

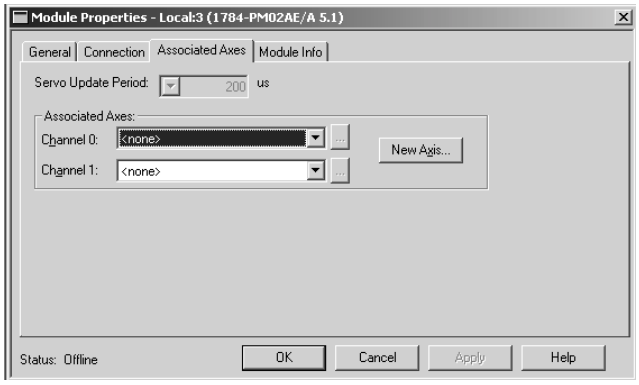


Use this tab:	To:
General	Do the following for an axis: • configure the axis • assign the axis, or terminate the assignment of an axis, to a Motion Group. • associate the axis with a motion card RSLogix 5000 software supports only one Motion Group tag per controller.
Motion Planner	Determine how many output cam execution nodes (instances) are created for an axis. The Execution Target parameter for the MAOC/MDOC instructions specify which of the configured execution nodes the instruction affects.
Units	Determine the units you will use to define your motion axis.
Drive/Motor	Configure the servo loop for an axis and open the Custom Drive Scaling Attributes dialog box.
Motor Feedback	Configure a motor feedback device, if any, for the axis.
Aux Feedback	Configure an auxiliary feedback deice, if any, for the axis.
Conversion	View the pos.itioning mode (if applicable) and configure the feedback resolution for an axis
Homing	Configure the attributes related to homing an axis.
Hookup	Configure and initiate axis hookup and marker test sequences for an axis.
Tune	Configure and initiate the axis tuning sequence for an axis.
Dynamics	View or edit the dynamics related parameters for an axis.
Gains	Perform the following offline functions: • adjust, or "tweak" gain values that have been automatically set by the tuning process (in the Tune tab of this dialog) • manually configure gains for the velocity and position loops for an axis.

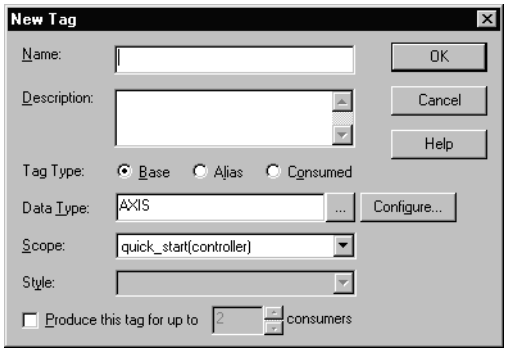
Use this tab:	To:
Output	Make the following offline configurations: • set the torque scaling value, which is used to generate gains • enable and configure the Notch Filter • enable and configure servo's low-pass digital output filter for an axis
Limits	Make the following offline configurations: • enable and set maximum positive and negative software travel limits • configure both Position Error Tolerance and Position Lock Tolerance for an axis
Offset	Make offline adjustments to the following Servo Output values: • friction compensation, • velocity offset • torque offset
Fault Action	Specify the actions that will be taken in response to these faults: • drive thermal fault • motor thermal fault • feedback noise fault • feedback fault • position error fault • hard overtravel fault • soft overtravel fault
Tag	Modify the name and description of the axis.

Creating an Analog Motion Axis and Group

To create an axis, click **New Axis** on the Associated Axes tab in the module properties window.



Specify this information:



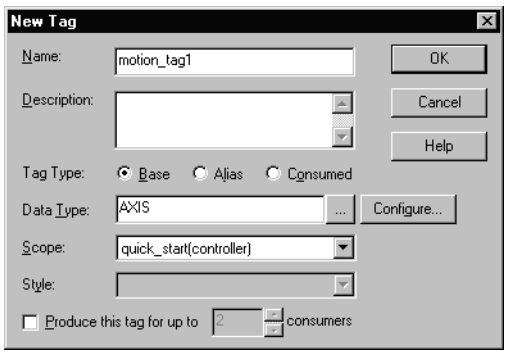
In this field:	Type:
Name	The name of the axis.
Description	A description of the axis (optional).

Configuring an analog axis

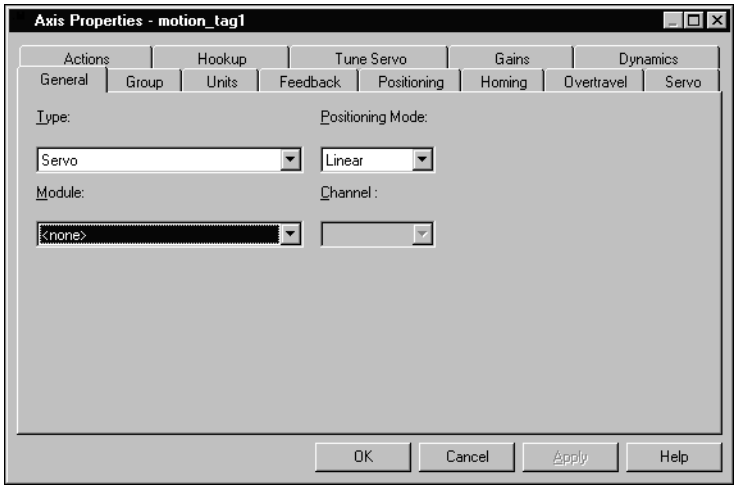
To configure the axis:

1. Click **Configure** in the new tag window.

Enter an axis name. →



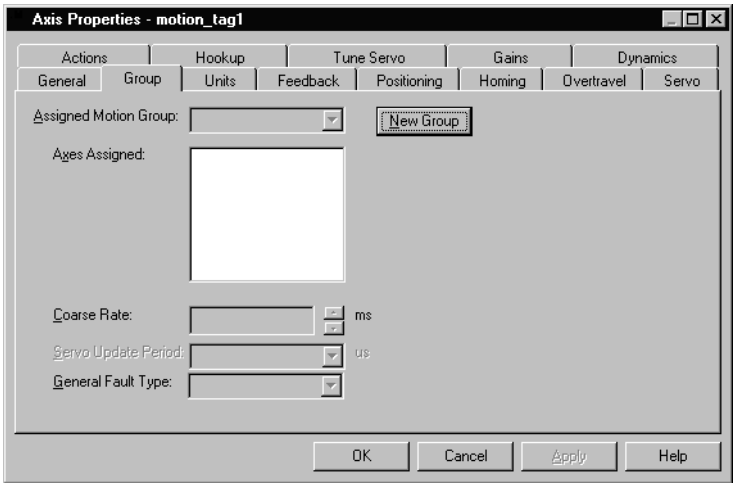
2. On the General tab, select the type of axis and positioning mode. (You assign a motion card and channel to the axis later.)



In this field:	Select the:
Type	Type of axis you want
Positioning Mode	Type of axis positioning you want to use

3. Click **OK**.

4. On the Group tab, assign a motion group.

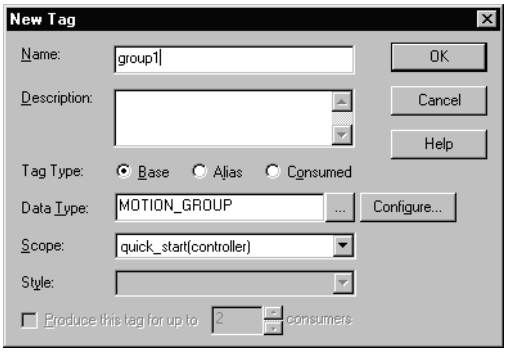


To:	Then:
create a new motion group	Click **New Group**.
use an existing motion group	Go to Step 7.

IMPORTANT During configuration, you must name and configure a motion group, which results in a MOTION_GROUP tag. After configuring the motion group, you can assign your axes to your motion group.

5. Specify this information:

Make sure you enter a group name. →

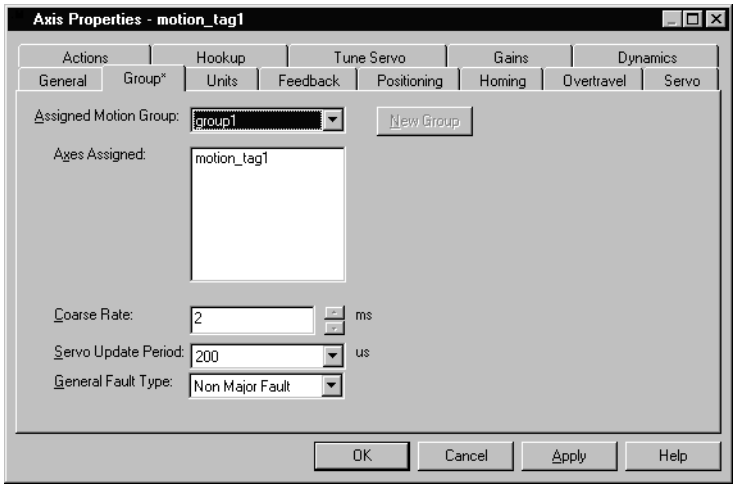


In this field:	Type:
Name	The name of the motion group.
Description	A description of the motion group (optional).

6. Click **OK**.

7. On the Group tab, assign the axis to a motion group and specify this information:

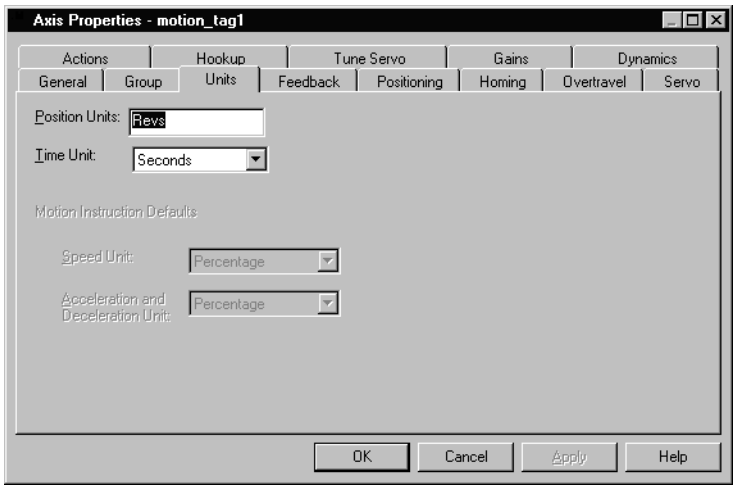
Select the motion group. →



In this field:	Select the:
Assigned Motion Group	Motion group.
Coarse Rate	Update rate for your axis
Servo Update Period	Closure time interval for your axis
General Fault Type	Fault type for your axis

8. Click **OK**.

9. On the Units tab, define the position units in which you want to program (e.g., meters, yards, feet, degrees).

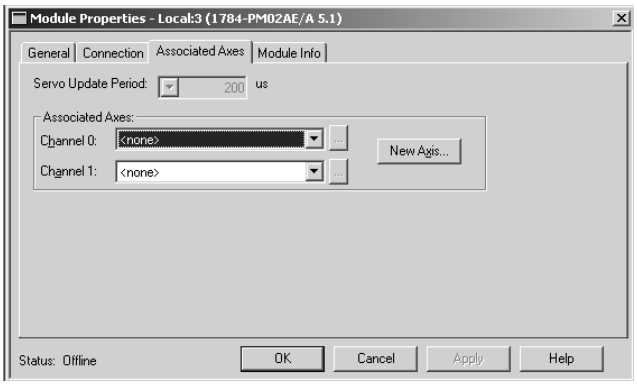


10. Click **OK**.

11. To continue configuring your axis, complete the entries on the other tabs. When finished with the entries on a tab, click **OK**.

IMPORTANT The diagnostic testing and auto tuning options are available only when the controller is online. Before going online, complete the configuration of all the motion cards and download your application program.

12. Assign the axis to a channel (the physical connection on the motion card to which the axis is wired).



To:	Then:
Assign your axis to channel 0	In the *Channel 0* field, select your axis from the drop-down menu
Assign your axis to channel 1	In the *Channel 1* field, select your axis from the drop-down menu
Add another axis	Click **New Axis**.
Complete your configuration	Select **Finish**.

IMPORTANT You can also name and configure axes and motion groups using the controller tag editor. The tag editor supports copy and paste operations, which can make axis naming and configuration easier and faster.

Running Hookup Diagnostics and Autotuning

Once you add and configure the motion cards and their axes, you can download your program. After going online, complete hookup diagnostics and auto tuning.

1. Download your project.

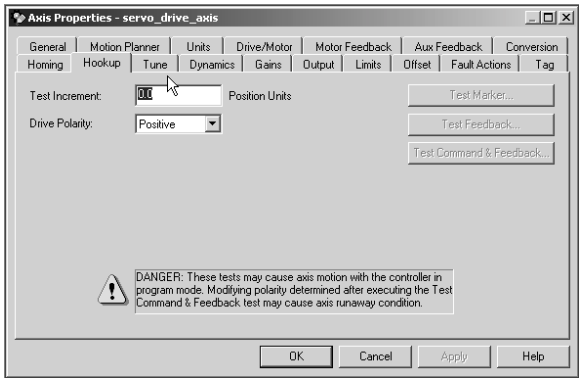
TIP

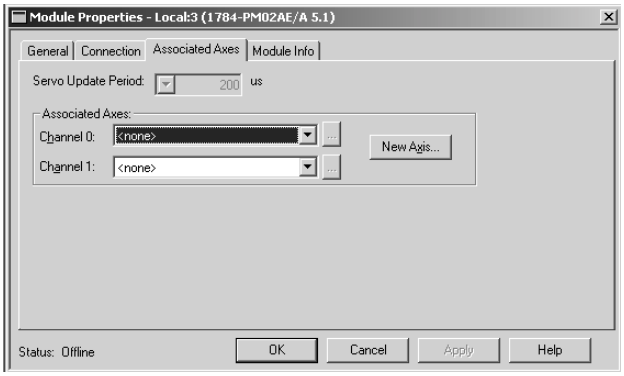
The project can be a blank program, but it must include complete configuration information for all your modules and axes.

2. Verify that a connection is established with each module in the I/O configuration of the controller.

3. Access the Hookup test tab:

1784-PM16SE SERCOS motion card:	1784-PM02A analog motion:
Right-click on the axis in the Controller Organizer and select Properties.	In the module properties window for the motion card, select the channel that you assigned to the axis.





4. Select the Hookup tab and run the three hookup diagnostics.

   When the tests are is finished, the dialog box displays "Complete."

5. Select the Tune tab and run auto tuning.

6. When diagnostic testing and auto tuning are complete, click **OK**.

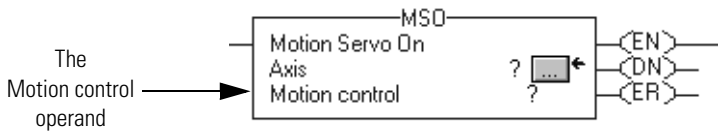
For more information about hookup diagnostics, see the *SoftLogix Servo Card Setup and Configuration User Manual*, publication 1784-UM003.

Developing Logic for Motion Control

The motion instructions operate on one or more axes. You must identify and configure axes before you can use them.

For more information on individual motion instructions, see the *Logix5000 Controllers Motion Instruction Set Reference Manual*, publication 1756-RM007.

Each motion instruction has an operand named Motion control. This field uses a MOTION_INSTRUCTION tag to store status information during the execution of motion instructions. This status information can include instruction status, errors, etc.



ATTENTION

Tags used for the motion control operand of motion instruction should only be used once. Re-use of the same motion control operand in other instructions can cause unintended operation of the control variables.

You can read motion status and configuration parameters in your logic using two methods.

Method:	Example:
Directly accessing the MOTION_GROUP and AXIS structures	• Axis faults • Motion status • Servo status
Using the GSV instruction	• Actual position • Command position • Actual velocity

In your ladder logic program, you can modify motion configuration parameters using the SSV instruction. For example, you can change position loop gain, velocity loop gain, and current limits within your program.

For more information on the SSV instruction, see the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

Handling motion faults

Two types of motion faults exist.

Type	Description	Example
Errors	• Do not impact controller operation • Should be correct to optimize execution time and ensure program accuracy	A Motion Axis Move (MAM) instruction with a parameter out of range
Minor/Major	• Caused by a problem with the servo loop • Can shutdown the controller if you do not correct the fault condition	The application exceeded the PositionErrorTolerance value.

You can configure a fault as either minor or major by using the Axis Wizard-Group window.

Understanding errors

Executing a motion instruction within an application program can generate errors. The MOTION_INSTRUCTION tag has a field that contains the error code. For more information about error codes for individual instructions, see the *Logix5000 Controllers Motion Instruction Set Reference Manual*, publication 1756-RM007.

Understanding minor/major faults

Several faults can occur that are not caused by motion instructions. For example, a loss of encoder feedback or an actual position exceeding an overtravel limit will cause faults. The motion faults are considered type 2 faults with error codes from 1 to 32. See *Logix5000 Controllers Common Procedures Programming Manual*, publication 1756-PM001.

Running a Motion Application Using Microsoft Windows XP

The new System Restore feature of Windows XP affects SoftLogix motion applications. When System Restore is enabled, random motion retries occur, which can result in irregular motion or motion glitches.

The System Restore feature provides a way to restore the system to a previously known state that would otherwise require you to reinstall an application or even the entire operating system. Setup applications that are compatible with Windows XP integrate with System Restore to create a restore point before an installation begins. By default, the feature creates a restore point every 24 hours while the system is up. It does this by creating a restore point directory, then snapshotting a set of critical system files, including parts of the registry. System Restore tracks changes to files and directories, and saves copies of files that are being changed or deleted in a restore point change log. Restore point data is maintained on a per-volume basis.

To disable System Restore:

1. From the Start Menu, right click on My Computer and select Properties. This displays System Properties.
2. Select the System Restore tab from the System Properties page.
3. Check the box labeled Turn off System Restore.
4. Click on the OK button for the change to take effect.

Communicating with Devices on a DeviceNet Link

Using This Chapter

Configuring Your System for a DeviceNet Link

For the SoftLogix controller to operate on a DeviceNet network, you need:

- a 1784-PCIDS DeviceNet communication card
- RSLinx software to install the DeviceNet communication driver
- RSLinx software to install the virtual backplane driver

  You only install the virtual backplane driver once on the computer where you run the SoftLogix controller. This chapter assumes you have already installed the driver. For an example of installing the driver, see chapter 1, “Getting Started.”

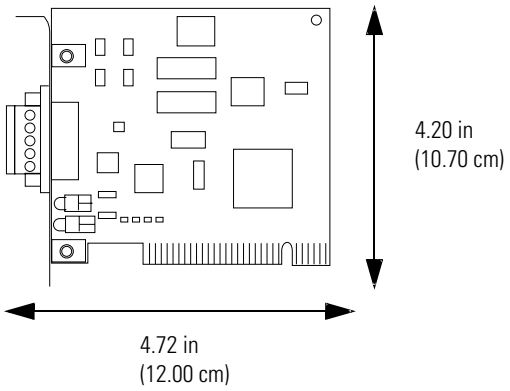
- RSLogix 5000 programming software to configure the communication card as part of the SoftLogix system
- RSNetWorx for DeviceNet software to configure the devices on the network
- IOLinx software must be installed for the SoftLogix controller to be able to read and write I/O data

Step 1: Install the hardware

Make sure the 1784-PCIDS communication card is properly installed in the computer. You need to:

- Install the card in any PCI slot within the computer.

  It does not matter which PCI slot you use for the communication card. The PCI slot in the computer **does not** correspond to the backplane slot in the SoftLogix chassis. You use the SoftLogix chassis monitor to place the communication card in a specific backplane slot (see the next page).



- Install IOLinx software so the SoftLogix controller can use the 1784-PCIDS communication card to control DeviceNet I/O.
- Make a label to place on the mounting bracket of the card, or use a pen to write on the mounting bracket of the card. The label should include the serial number of the card and a name you can use to identify the card from any others you might install in the computer

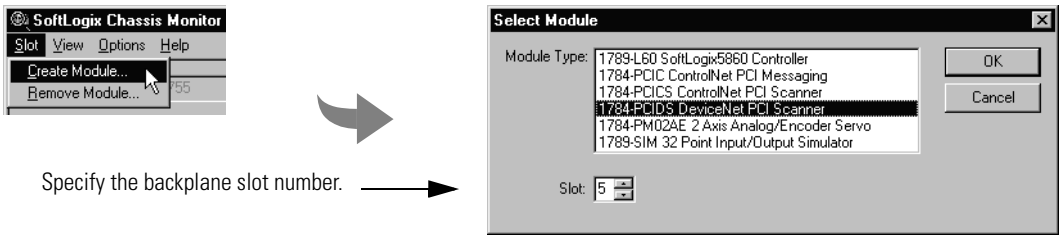
  Remember the serial number of each communication card you install. You use the serial number to identify which card you want in which slot of the SoftLogix chassis.

For more information about installing a 1784-PCIDS communication card, see the *DeviceNet PCI Interface Card Installation Instructions*, publication 1784-IN004.

Step 2: Create the communication card in the chassis

Before you can connect the SoftLogix system to the DeviceNet network, you must create the 1784-PCIDS card as part of the SoftLogix chassis.

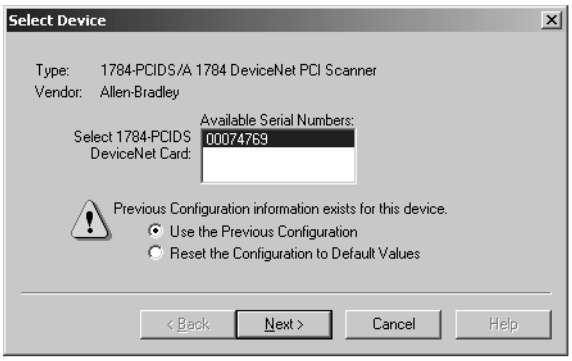
1. From the SoftLogix chassis monitor, select Slot → Create Module or right click the appropriate slot and select Create. Select the 1784-PCIDS card.



Specify the backplane slot number.

Click OK

2. Select the serial number of the 1784-PCIDS card you want.



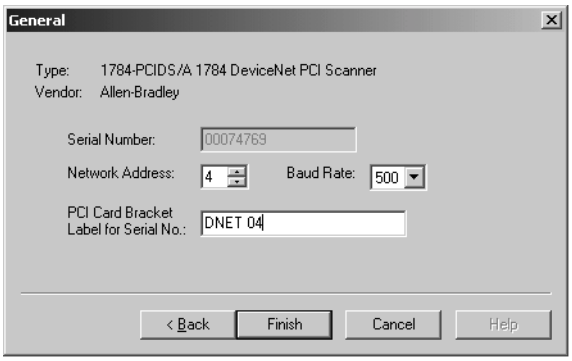
Select the serial number of the card.

If you previously configured the 1784-PCIDS card that you selected by serial number, the chassis monitor remembers the configuration from the last time you used the card (whether in the same or different slot).

Click Next

3. Specify configuration settings for the 1784-PCIDS card:

- specify the node address (MAC ID) on the DeviceNet network
- specify the data rate
- enter the label name for the card (this is the name you wrote on the label of the card to help you identify the card from others in the same computer)

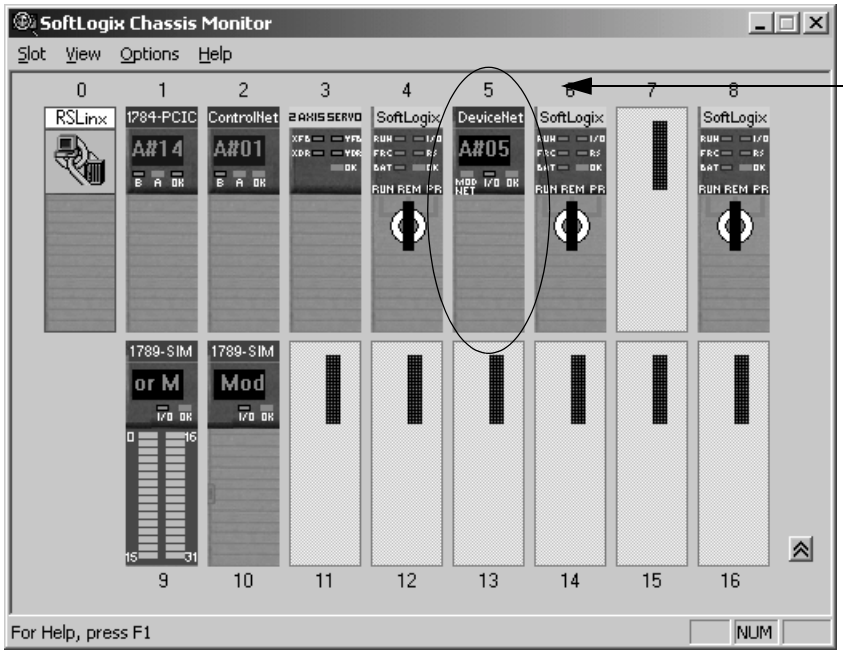


Click Finish

You can specify any slot number greater than 0 for the communication card. RSLinx software resides in slot 0.

IMPORTANT When you add a 1784-PCIDS card to the chassis monitor, the card must be connected to a valid, powered DeviceNet network. And, the baud rate you choose for card must be same as the baud rate for the DeviceNet network. Otherwise, the card will fail to insert in the chassis monitor.

The chassis monitor shows the 1784-PCIDS card as a virtual module in the SoftLogix chassis. The LEDs on the virtual monitor emulate a 1756-DNB communication module.

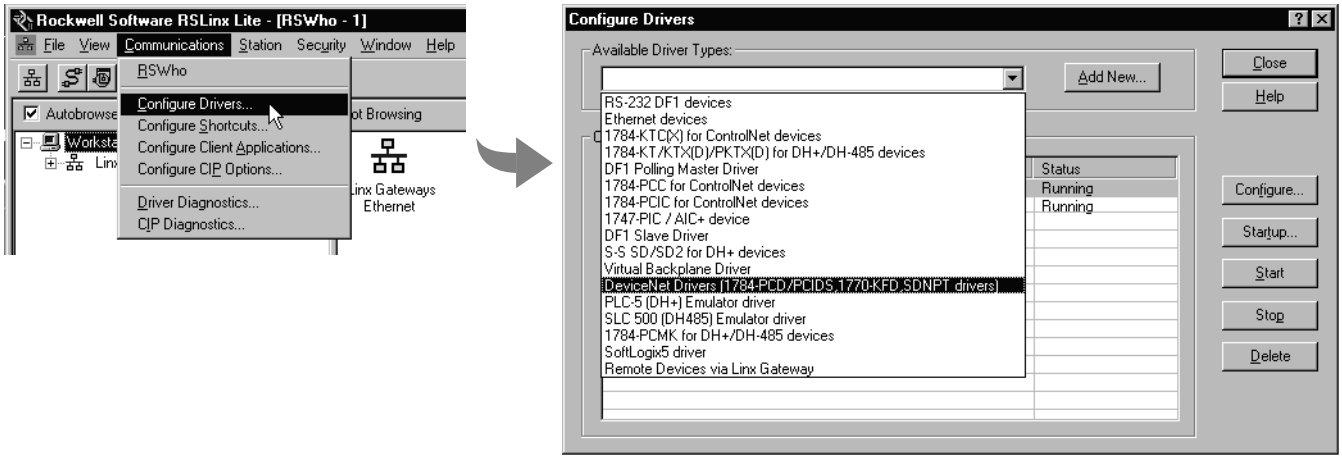


This chassis monitor has a 1784-PCIDS card installed in slot 5.

Step 3: Install the communication driver

Use RSLinx software to configure the DeviceNet communication driver for the 1784-PCIDS communication card.

1. In RSLinx software, select Configure Driver. Select the appropriate driver.



The device settings will be grayed out because you specified the baud rate and node address when you created the module in the SoftLogix chassis.

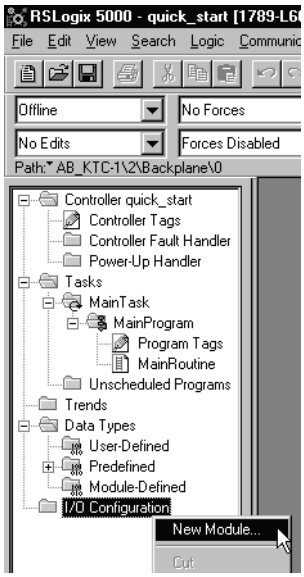
You only have to install the DeviceNet communication driver on the computer that you use to run RSNetWorx for DeviceNet. This example assumes that you are running the SoftLogix controller and RSNetWorx on the same computer.

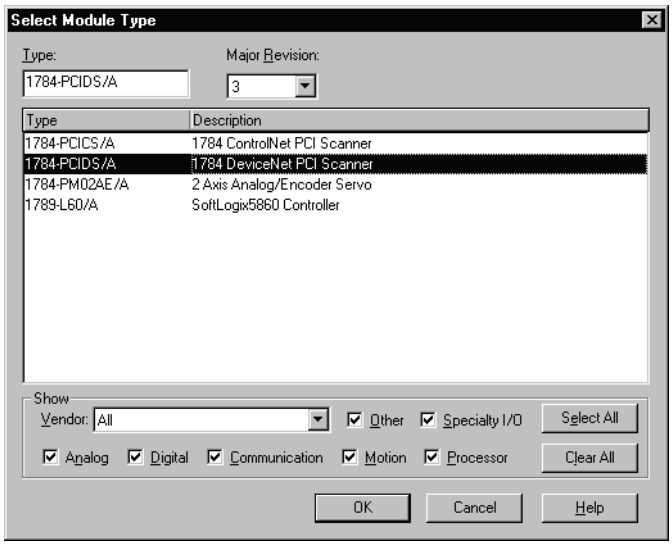
Step 4: Configure the communication card as part of the project

Use RSLogix 5000 programming software to map the 1784-PCIDS communication card as part of the SoftLogix project. In the Controller Organizer, add the communication card to the I/O Configuration folder.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.

2 Right-click to select New Module and add a 1784-PCIDS communication card.

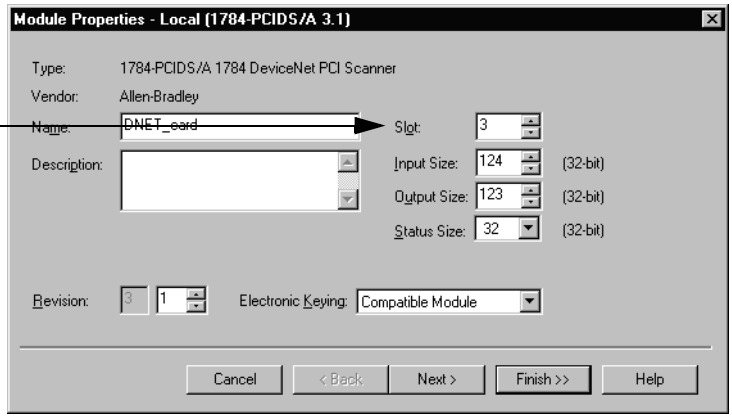




3. Specify the appropriate communication card settings.

4. This must be the same slot number you specified on the SoftLogix chassis monitor.

Make sure your selections for Input Size, Output Size, and Status Size are big enough to hold the data you expect. If the sizes are too small, data will be truncated. If the sizes are too big, the software zero pads the data blocks.

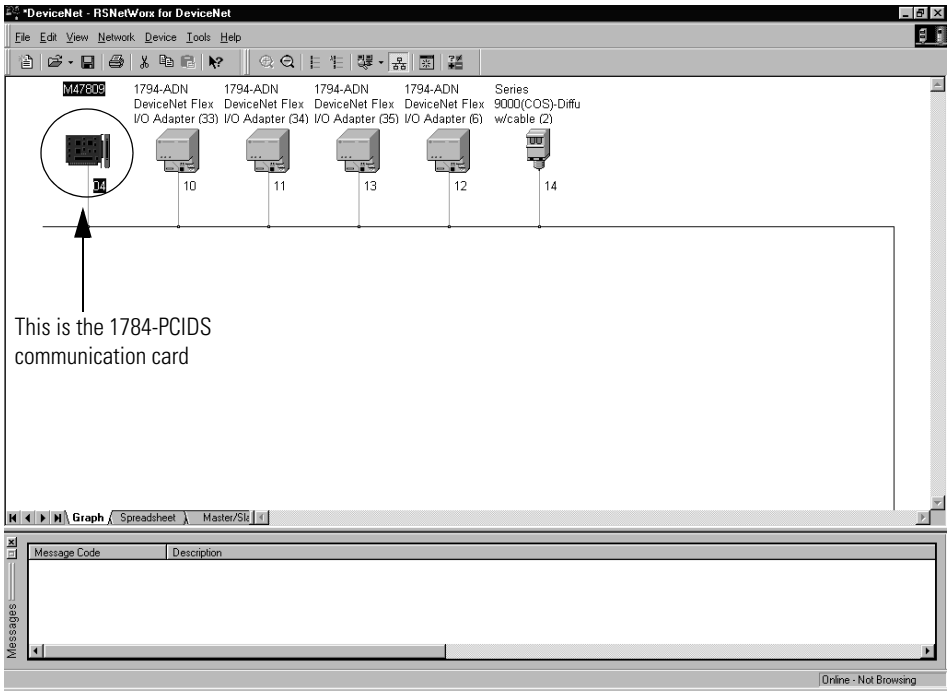


The virtual backplane driver must be installed via RSLinx software before you can download a project to the SoftLogix controller. Complete your system configuration and develop your program logic. Then download the project to the SoftLogix controller.

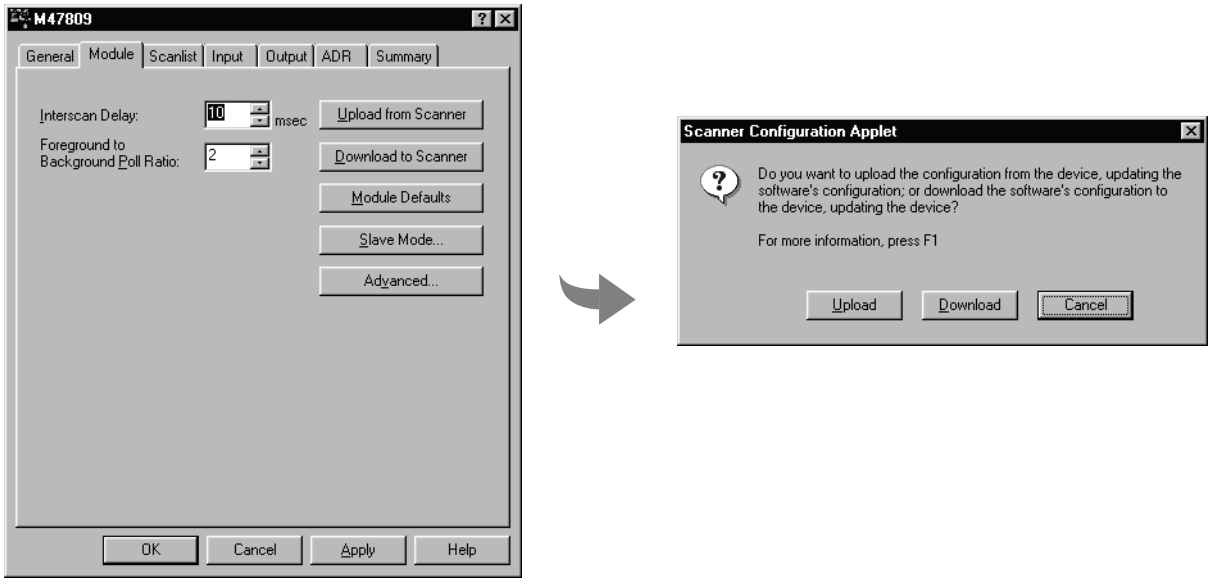
Step 5: Define the scan list

Use RSNetWorx for DeviceNet software to define the scan list. The project must already be downloaded from RSLogix 5000 programming software to the controller and the controller must be in Program or Remote Program mode.

1. In RSNetWorx software, go online, enable edits, and survey the network.



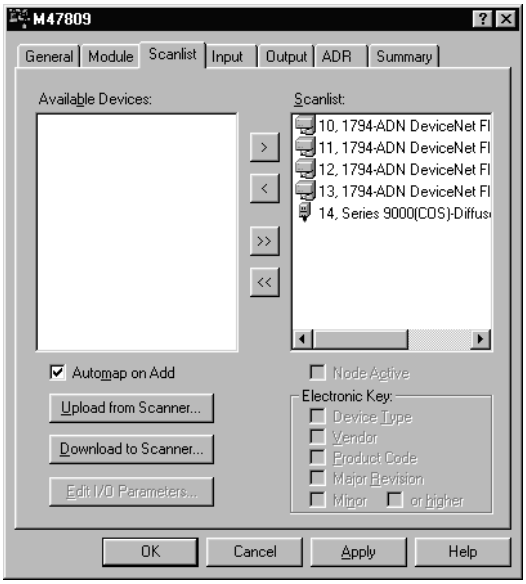
2. Double-click the 1784-PCIDS card and select the Module tab to configure the card. Upload the network information when prompted.



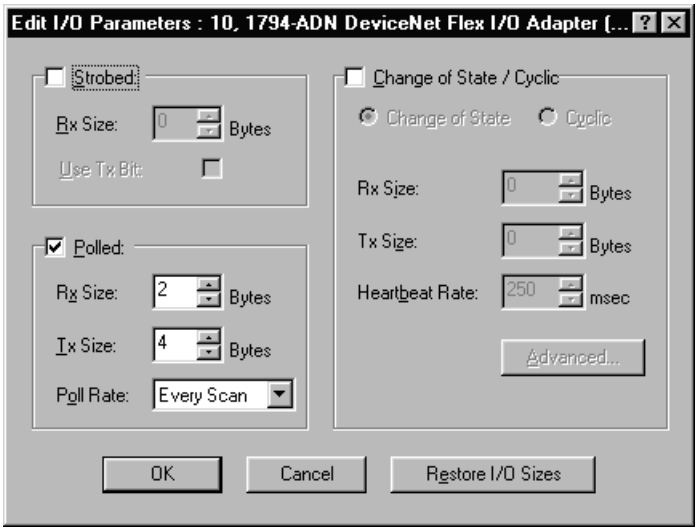
Every device on the network must be in Program or Remote Program mode for the software to re-write all of its connections. If a device is not in the correct mode, the software prompts you to let it change the device's mode.

continued

3. Use the ScanList tab to define the scanning order of the DeviceNet devices.

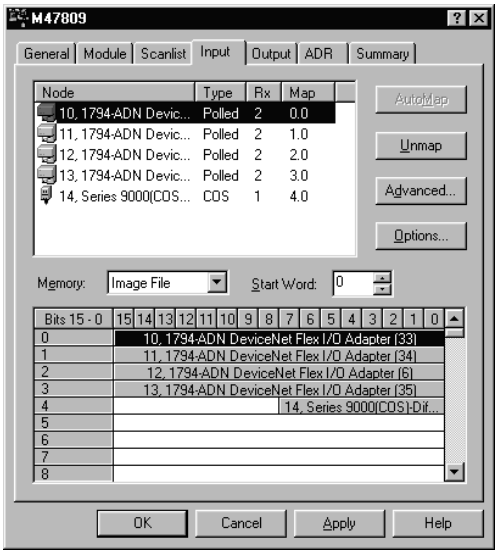


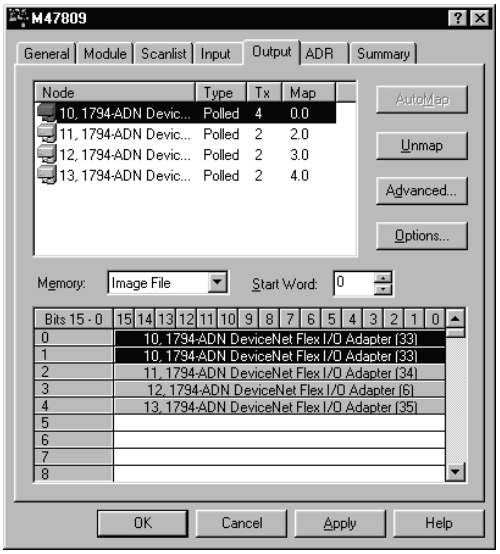
4. Click Edit I/O Parameters to define how many inputs (Rx) and Outputs (Tx) you expect from each DeviceNet device.



continued

5. Use the ScanList tab to define the scanning order of the DeviceNet devices.





If you place the SoftLogix5800 controller in Program mode with DeviceNet I/O currently mapped through a 1784-PCIDS module, and then you use RSNetWorx to change the data mapping on the network, the controller does not detect this change until the 1784-PCIDS module is reset. You can reset the module in the RSLogix 5000 Controller Organizer. Right-mouse click over the module and select Properties; then select the Module Info tab and click the Reset Module button. You can also reset the module by removing and re-inserting the module in the SoftLogix chassis. You can reset the module while the SoftLogix controller is running. The connections are automatically re-established after the 1784-PCIDS module is reset.

ATTENTION

Do not reset a module that is currently being used for control. The connection to the module will be broken and control might be interrupted.

The SoftLogix controller supports 32-bit words of data. You can have 124 words of input data, 123 words of output data, and 32 words of device status data. How you configure the DeviceNet devices determines how many words you use per device.

Most DeviceNet devices support 16-bit words. Take care how you map these into the 32-bit words used in RSLogix 5000 programming software. RSNetWorx for DeviceNet lets you word-align the device data. While this might simplify the organization of the data, it might also limit the data you have available.

Accessing DeviceNet I/O

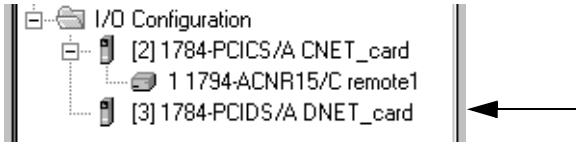
I/O information is presented as a structure of multiple fields, which depend on the specific features of the I/O module. The name of the structure is based on the location of the I/O module in the system. Each I/O tag is automatically created when you configure the I/O module through the programming software. Each tag name follows this format:

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

where:

This address variable:	Is:
Location	Identifies network location LOCAL = identifies communication card within the computer
SlotNumber	Slot number of I/O module in its chassis
Type	Type of data I = input O = output C = configuration S = status
MemberName	Specific data from the I/O module; depends on the type of data the module can store For example, Data and Fault are possible fields of data for an I/O module. Data is the common name for values the are sent to or received from I/O points.
SubMemberName	Specific data related to a MemberName.
Bit (optional)	Specific point on the I/O module; depends on the size of the I/O module (0-31 for a 32-point module)

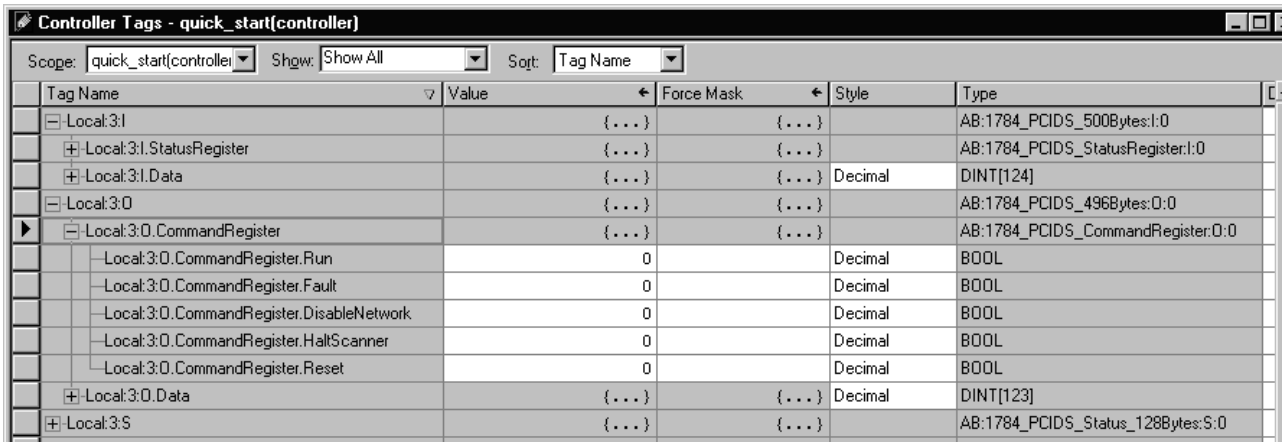
EXAMPLE



The 1784-PCIDS card in this example is in slot 3.

The data for a 1784-PCIDS card is always configured as a rack-optimized connection.

The rack-optimized connection creates a DINT element for each possible I/O module connected to the device in slot 3, “Local:3.” The array Local:3:I.Data contains the possible input elements; the Local:3:O.Data contains the possible output elements.



Tag Name	Value	Force Mask	Style	Type
Local:3:I	{...}	{...}		AB:1784_PCIDS_500Bytes:I:0
Local:3:I.StatusRegister	{...}	{...}		AB:1784_PCIDS_StatusRegister:I:0
Local:3:I.Data	{...}	{...}	Decimal	DINT[124]
Local:3:O	{...}	{...}		AB:1784_PCIDS_496Bytes:O:0
Local:3:O.CommandRegister	{...}	{...}		AB:1784_PCIDS_CommandRegister:O:0
Local:3:O.CommandRegister.Run	0		Decimal	BOOL
Local:3:O.CommandRegister.Fault	0		Decimal	BOOL
Local:3:O.CommandRegister.DisableNetwork	0		Decimal	BOOL
Local:3:O.CommandRegister.HaltScanner	0		Decimal	BOOL
Local:3:O.CommandRegister.Reset	0		Decimal	BOOL
Local:3:O.Data	{...}	{...}	Decimal	DINT[123]
Local:3:S	{...}	{...}		AB:1784_PCIDS_Status_128Bytes:S:0

The index number on the array element refers to the same numbered word mapped to the device in RSNetWorx for DeviceNet. Depending on the device, there can be several words mapped to one device. You can create aliases to the elements you actually use to more identify the data you need.

Determining how often to update data

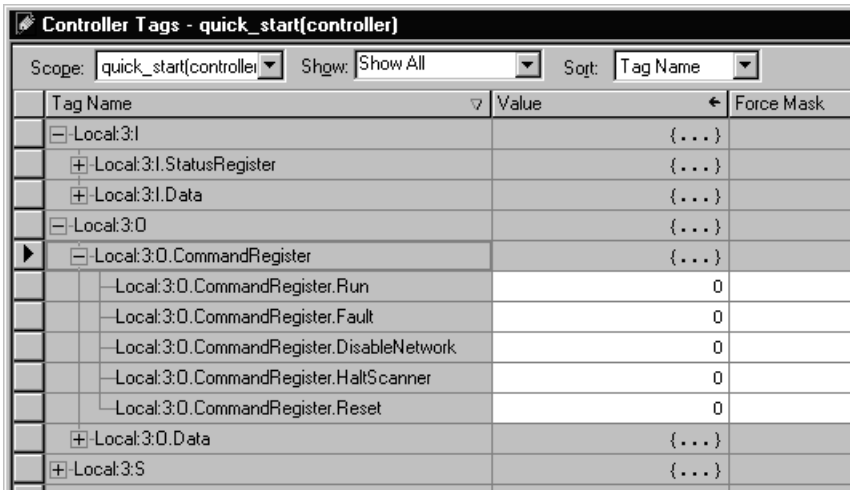
When you configure the 1784-PCIDS card, you can specify a requested packet interval (RPI) time. The RPI you select specifies the maximum amount of time between data updates.

The 1784-PCIDS card supports an RPI range of 2.0 to 750.0 ms. The default is 5.0 ms.

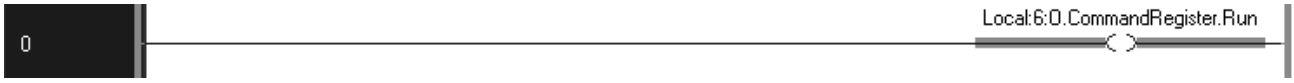
Placing the Communication Card in Run Mode

To place the 1784-PCIDS card in Run mode, your program logic needs to set the CommandRegister.Run bit in the output word for the card.

Set this bit →



For example:



Using the CommandRegister bits

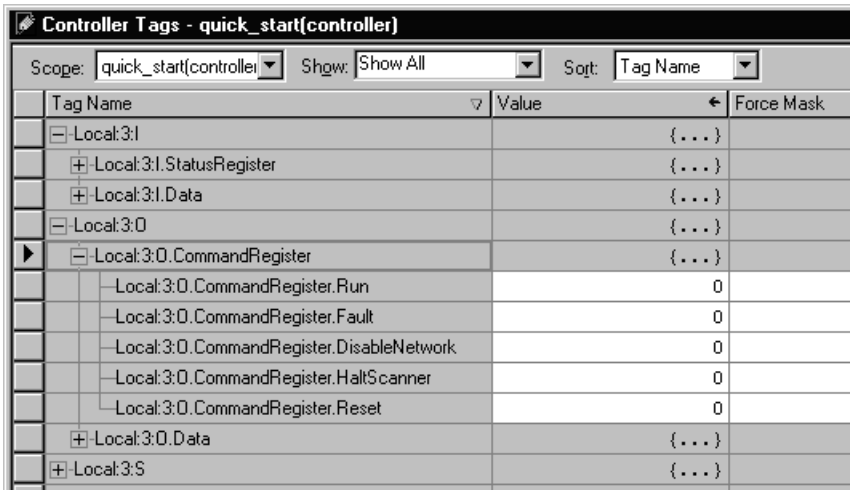
The following table describes how the 1784-PCIDS card uses the CommandRegister bits.

When CommandRegister.Run is set to:	The 1784-PCIDS card:
zero (0)	is in Idle mode In Idle mode, the card still receives inputs from its slave devices on the network, but the card does not send active output data to the devices.
one (1)	is in Run mode In Run mode, the card sends active outputs on the network and receives inputs.

Monitoring the 1784-PCIDS Card

The input data for the 1784-PCIDS card includes a StatusRegister.

StatusRegister bits →



The following table describes how the 1784-PCIDS card uses the StatusRegister bits.

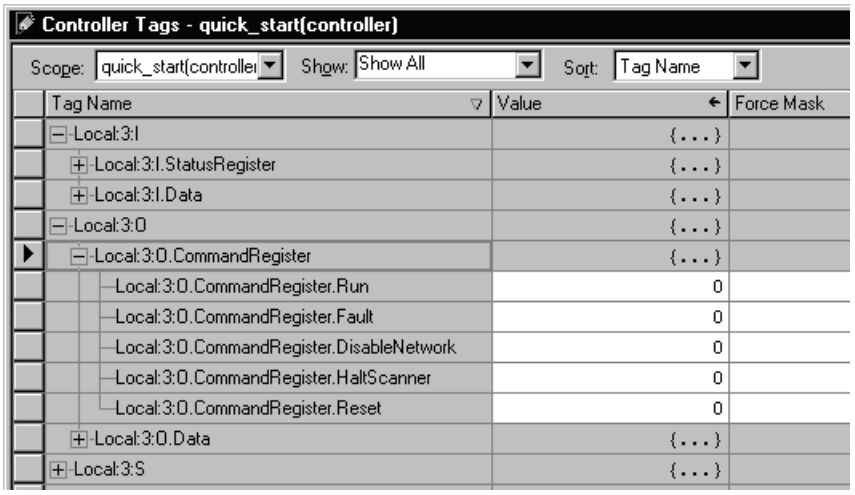
StatusRegister Bit:	Description:
StatusRegister.Run	This bit echoes the CommandRegister.Run bit to determine if the card is in Run or Idle mode. A 0 in this bit means the card is in Idle. A 1 means the card is in Run mode.
StatusRegister.Fault	This bit identifies whether the card is in Fault mode. The SoftLogix controller sets this bit based on the corresponding IOLinx status.
StatusRegister.DisableNetwork	The SoftLogix controller does not use this bit. The controller clears this bit to 0.
StatusRegister.DeviceFailure	This bit determines if general communication is OK between the card and its slave nodes. A node falling off the network or other communication problems to any device on the card's scan list sets this bit to 1. This bit is used in conjunction with the DeviceFailure table in the Status section to determine which node(s) are having communication problems. A 0 in this bit means that all the slave nodes are being successfully communicated to. A 1 means the card has at least one device with communication problems.
StatusRegister.Autoverify	This bit determines if the data Transmit and Receive sizes in the scan list are correct. Any node whose data sizes don't match the sizes defined in the scan list cause the bit to be set to 1. This bit is used in conjunction with the AutoVerify table in the Status section to determine which node(s) have incorrect data sizes. A 0 in this bit means that all the slaves have correct data sizes. A 1 means the card has at least one device on its scan list with an incorrect data size.

StatusRegister Bit:	Description:
StatusRegister.CommFailure	This bit identifies whether a channel wide communication fault is happening with the card. For example if the card detects severe communication problems on the network it will go into a Bus Off condition. This also cause the StatusRegister.CommFailure bit to turn on. A 0 in this bit means that the card is communicating correctly. A 1 means the card detected a channel wide communication problem.
StatusRegister.DupNodeFail	This bit shows if the card is attempting to go online on a DeviceNet network with the same node number as an existing device on the network. A 0 in this bit means that the card has NOT detected another node on the network with the same node number as the card. A 1 means that the card has the same node number as an existing device on the network.
StatusRegister.DnetPowerDetect	This bit shows if the card has detected that the DeviceNet 24VDC power is connected to its network connector and is also energized. A 0 in this bit means that the card has detected DeviceNet power on its network connector. A 1 means that the card has NOT detected DeviceNet power on its network connector.

Using the Status data

The status data for the 1784-PCIDS card includes several elements.

Status data elements

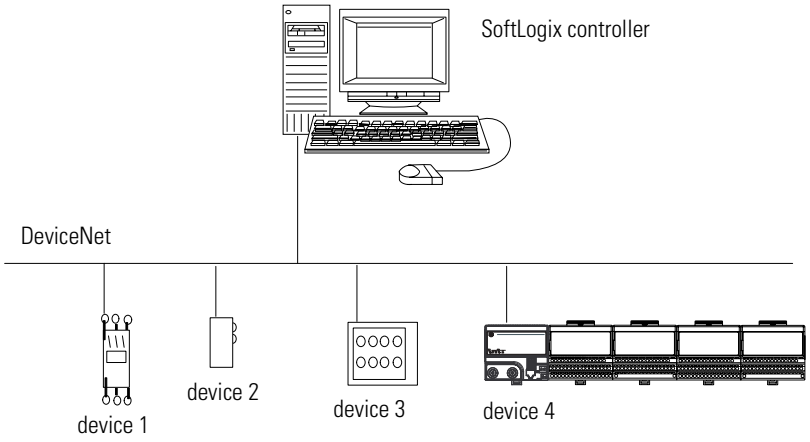


The following table describes the status data for a 1784-PCIDS card.

Status Element:	Description:
S.ScanCounter	This 32-bit word is incremented every time the card completes a network scan. By reading this value and counting how many network updates are done in a certain time, you can calculate an average scan time.
S.DeviceFailureRegister	This data area is an array of 8 bytes that make a 64-bit table. There is one bit for every one of the 64 possible node numbers on the network. The bit associated with a node number on the scan list is set to 1 if that node number is having communication problems.
S.AutoVerifyRegister	This data area is an array of 8 bytes that make a 64-bit table. There is one bit for every one of the 64 possible node numbers on the network. The bit associated with a node number on the scan list is set to 1 if that node number has a transmit and/or receive data size that does not match the scan list.
S.DeviceIdleRegister	This data area is an array of 8 bytes that make a 64-bit table. There is one bit for every one of the 64 possible node numbers on the network. The bit associated with a node number on the scan list is set to 1 if that node is not sending back input data to the card. This normally means that the node is in some kind of idle mode in which it stops sending output data back to the card's input table.
S.DeviceActiveRegister	This data area is an array of 8 bytes that make up a 64 bit table. There is one bit for every one of the 64 possible node numbers on the network. The bit associated with a node number on the scan list will go to a 1 if that node number has an active scan list entry in the 1784-PCIDS. This means that the 1784-PCIDS is communicating with that node.
S.DeviceStatusDisplay	This data area is an array of 4 bytes. There is one byte associated with each of the 4 characters of the alphanumeric display on the SoftLogix chassis monitor. Read these 4 bytes as ASCII characters to determine the exact message being displayed on the SoftLogix chassis monitor.

Example: SoftLogix Controller and DeviceNet I/O

In the following example, one SoftLogix controller controls I/O through a 1784-PCIDS communication card.



This example has a SoftLogix controller controlling four DeviceNet devices. The controller automatically creates a rack-optimized connection for the I/O data.

The tag name for the rack-optimized array tag is based on the slot number of the 1784-PCIDS card. For example, if you install the 1784-PCIDS card in slot 3 of the controller, the software automatically creates Local:3:I and Local:3:O data structures.

Creating alias tags

You might want to create alias tags to better represent the elements of the input and output array tags. An alias for an I/O point:

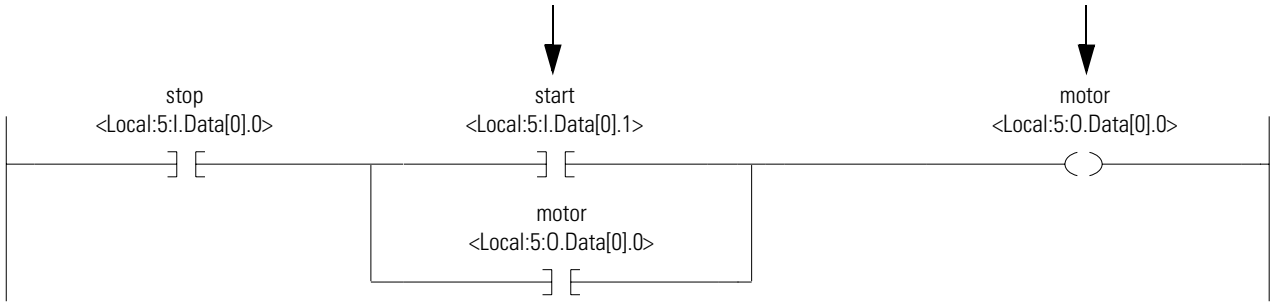
- provides a descriptive name for the device that is wired to the point
- represents the value of the point. When one changes, the other reflects the change.

When you enter an alias tag into programming logic, the programming software displays the original tag, along with the alias.

EXAMPLE The following logic was initially programmed using descriptive tag names, such as *start* and *motor*. Later, the tags were converted to aliases for the corresponding I/O devices.

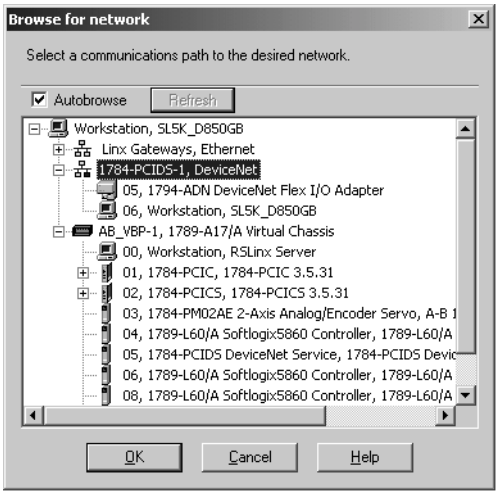
start is an alias for the push button at bit 1 of word 0 of the module in slot 5 of the local chassis. When the push button is on, *start* is on.

motor is an alias for the starter contactor at bit 0 of word 0 of the module in slot 5 of the local chassis. When *motor* turns on, the starter contactor turns on.



If you want to select the 1784-PCIDS card from an online list of available devices, such as Browse Network in RSNetWorx for DeviceNet software, select the 1784-PCIDS card from outside of the virtual chassis.

Select the 1784-PCIDS card from outside of the virtual chassis.



Notes:

Communicating with Devices on a ControlNet Link

Using This Chapter

Configuring Your System for a ControlNet Link

For the SoftLogix controller to operate on a ControlNet network, you need:

- a ControlNet communication card:
  - if you want to send messages and control I/O, including produced/consumed tags, over ControlNet, use a 1784-PCICS card

    This chapter shows how to configure the 1784-PCICS communication card.
  - if you want to only send messages over ControlNet, use a 1784-PCIC card
- RSLinx software to install the virtual backplane driver

  You only install the virtual backplane driver once on the computer where you run the SoftLogix controller. This chapter assumes you have already installed the driver. For an example of installing the driver, see chapter 1, "Getting Started."
- RSLogix 5000 programming software to configure the communication card as part of the SoftLogix system
- RSNetWorx for ControlNet software to schedule the SoftLogix system on the network

Step 1: Install the hardware

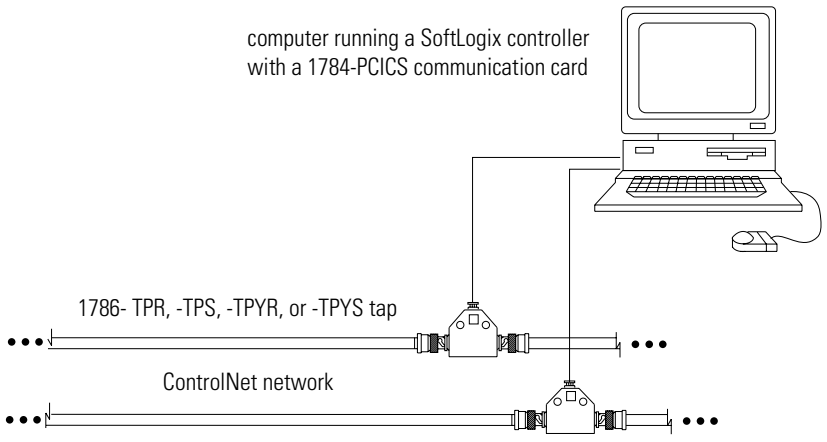
Make sure the 1784-PCICS communication card is properly installed in the computer. You need to:

- Install the card in any available PCI slot within the computer.

  It does not matter which PCI slot you use for the communication card. The PCI slot in the computer **does not** correspond to the backplane slot in the SoftLogix chassis. You use the SoftLogix chassis monitor to place the communication card in a specific backplane slot (see the next page).

- Make a label to place on the mounting bracket of the card, or use a pen to write on the mounting bracket of the card. The label should include the serial number of the card and a name you can use to identify the card from any others you might install in the computer.

  Remember the serial number and name of each communication card you install. You use this information to identify which card you want in which slot of the SoftLogix chassis.

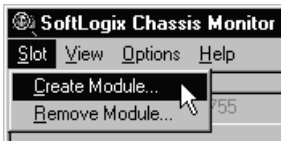


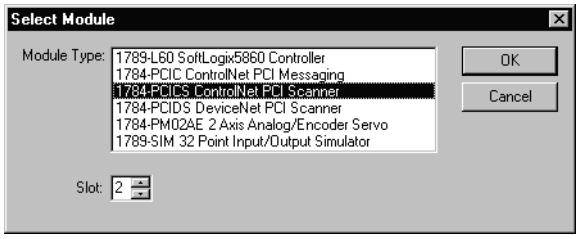
For more information about installing a 1784-PCICS communication card, see the *ControlNet PCI Interface Card Installation Instructions*, publication 1784-IN003.

Step 2: Create the communication card in the chassis

Before you can connect the SoftLogix system to the ControlNet network, you must create the 1784-PCICS card as part of the SoftLogix chassis.

1. From the SoftLogix chassis monitor, select Slot → Create Module or right click the appropriate slot and select Create. Select the 1784-PCICS card.

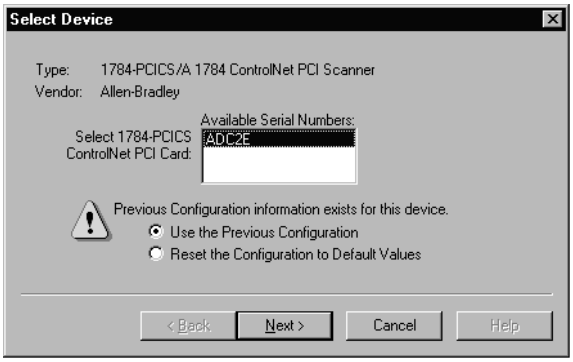




Specify the backplane slot number.

Click OK

2. Select the serial number of the 1784-PCICS card you want.



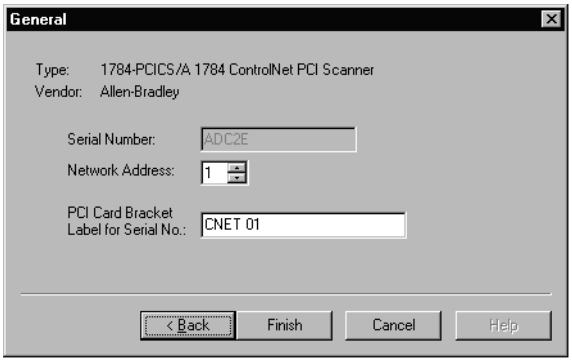
Select the serial number of the card.

If you previously configured the 1784-PCICS card that you selected by serial number, the chassis monitor remembers the configuration from the last time you used the card (whether in the same or different slot).

Click Next

3. Specify configuration settings for the 1784-PCICS card:

- specify the node address (MAC ID) on the ControlNet network
- enter the label name for the card (this is the name you wrote on the label of the card to help you identify the card from others in the same computer)



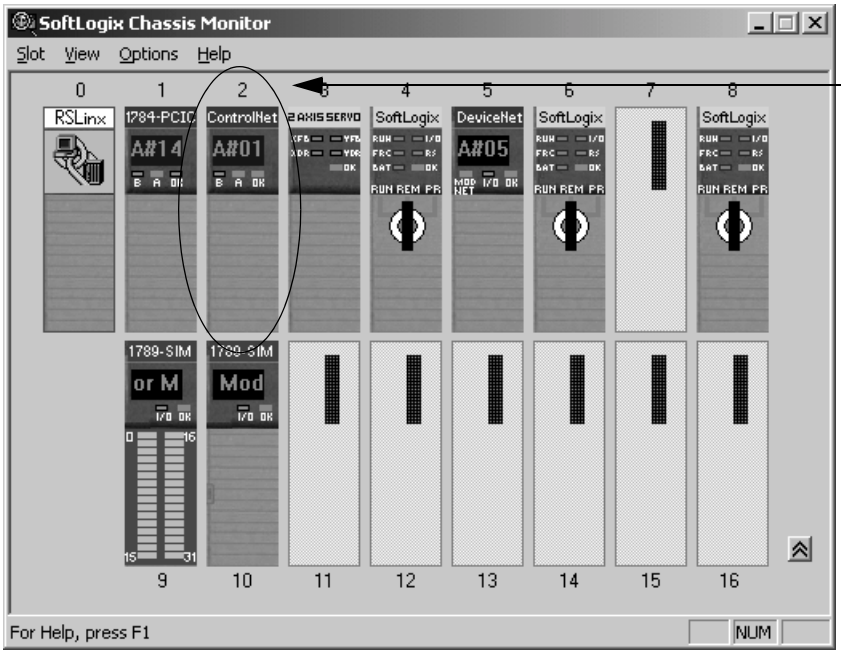
Click Finish

You can specify any slot number greater than 0 for the communication card. RSLinx software resides in slot 0.

By creating the card in the virtual chassis, you automatically install the communication driver information needed by the SoftLogix controller. Do not use RSLinx to install the communication driver for either the 1784-PCICS or 1784-PCIC communication card. Installing the communication driver through RSLinx adds the potential for conflicting configuration between RSLinx and the SoftLogix chassis monitor.

After you add the card to the chassis monitor, you can browse the network by expanding the Virtual Backplane driver and then expanding the port on the desired 1784-PCICS or 1784-PCIC communication card. Browsing ControlNet through the Virtual Backplane driver provides the same functionality as the RSLinx driver.

The chassis monitor shows the 1784-PCICS card as a virtual module in the SoftLogix chassis. The LEDs on the virtual monitor emulate a 1756-CNB communication module.



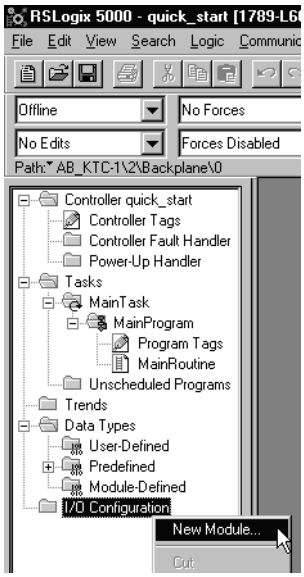
This chassis monitor has a 1784-PCICS card installed in slot 2.

Step 3: Configure the communication card as part of the project

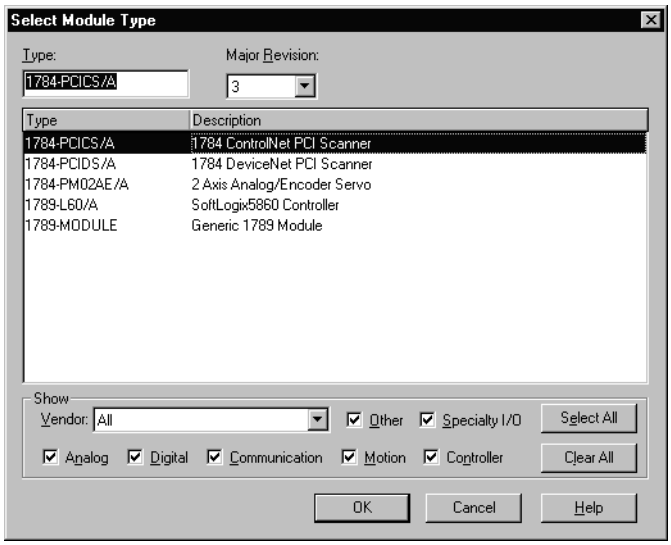
Use RSLogix 5000 programming software to map the 1784-PCICS communication card as part of the SoftLogix project. In the Controller Organizer, add the communication card to the I/O Configuration folder.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.

2 Right-click to select New Module and add a 1784-PCICS communication card.

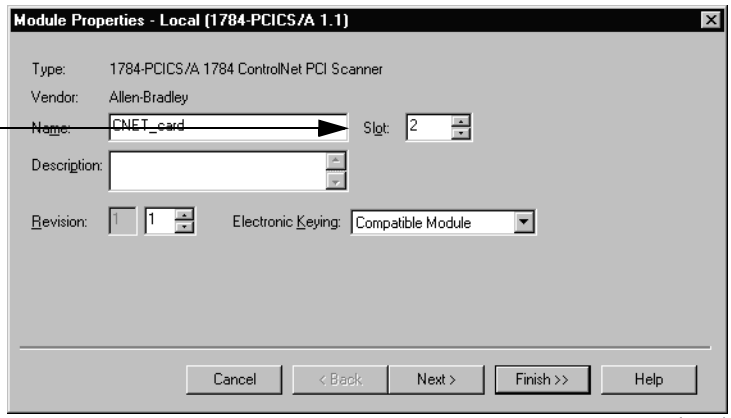


Click OK



3. Specify the appropriate communication card settings.

This must be the same slot number you specified on the SoftLogix chassis monitor.



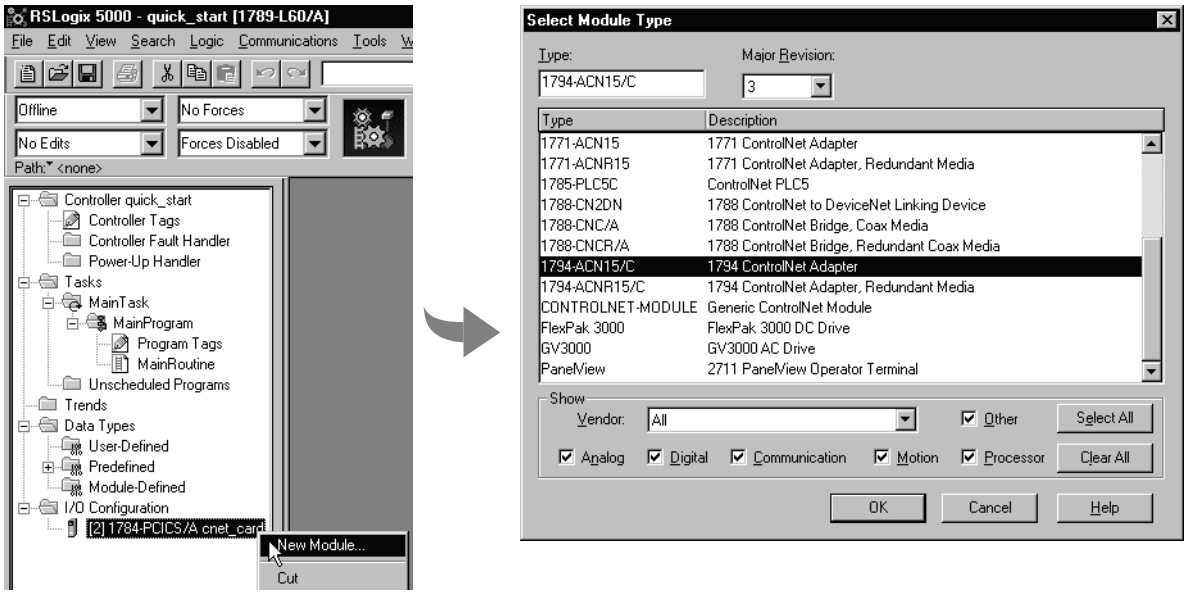
continued

The virtual backplane driver must be installed via RSLinx software before you can download a project to the SoftLogix controller.

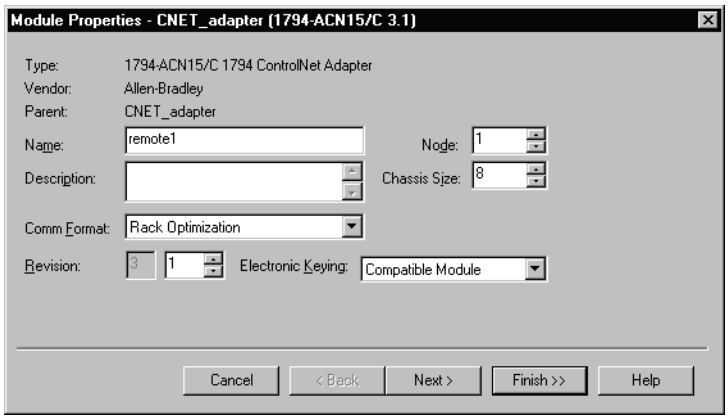
IMPORTANT Even if you plan to remotely program the controller over a ControlNet or Ethernet link, you must add the virtual backplane driver via RSLinx software. If you do not, persistent storage will not function and when you reboot the computer, the controller will come up with cleared memory (the program will not get re-loaded).

Complete your system configuration by adding the remote communication devices and appropriate I/O modules.

5. In the Controller Organizer, select the local 1784-PCICS communication you just added. Add and configure the remote communication device (1794-ACN15 in this example)



6. Specify the appropriate communication module settings.



7. Add and configure the I/O modules for the remote communication module you just added.

8. Save the current project and download it to the controller.

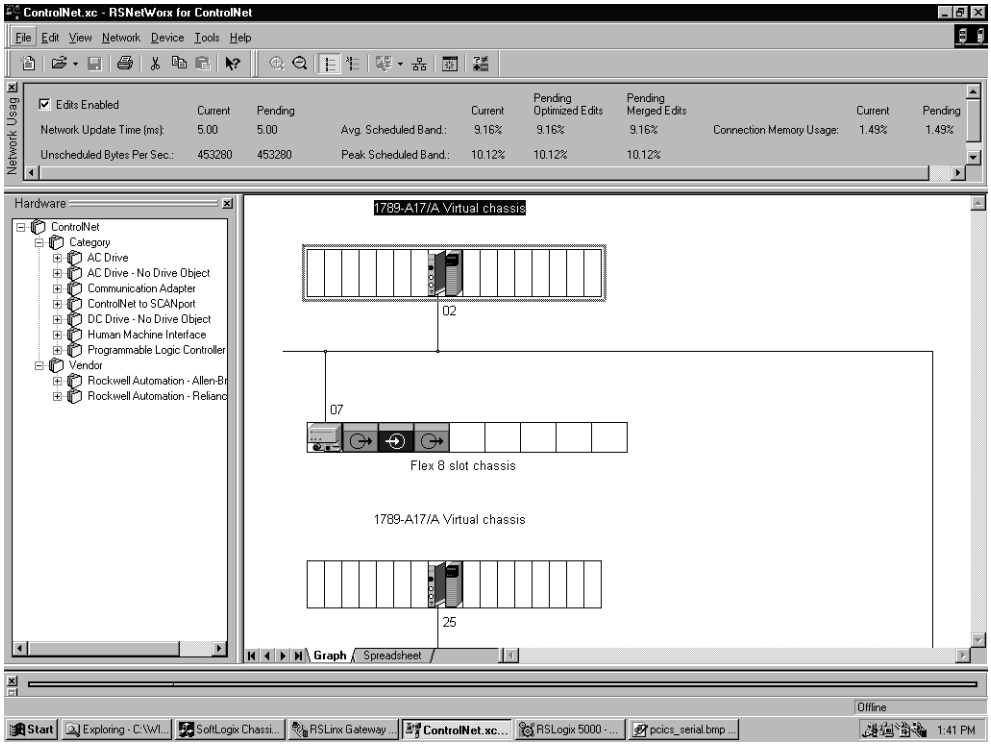
The local card becomes the “parent module” to the remote device. The controller organizer shows this parent/child relationship between local and remote communication devices.

Configure I/O modules for the remote communication module by adding them to the remote communication module (i.e., right-click the 1784-PCICS card and select New Module).

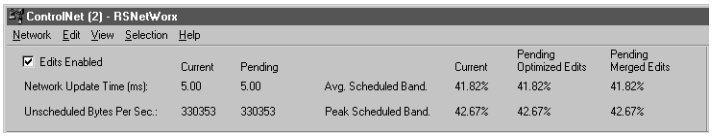
Step 4: Schedule the network

Use RSNetWorx software to schedule the ControlNet network. The controller project must already be downloaded from RSLogix 5000 programming software to the controller and the controller must be in Program or Remote Program mode.

1. In RSNetWorx software, go online, enable edits, and survey the network.



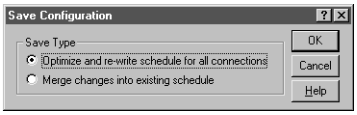
2. Specify the network update time (NUT).



The default NUT is 5ms.

The NUT you specify must be lower than or equal to the lowest RPI in your system.

3. After you specify the NUT, save and re-write the schedule for all connections.



Every device on the network must be in Program or Remote Program mode for the software to re-write all its connections. If a device is not in the correct mode, the software prompts you to let it change the device's mode.

Placing ControlNet I/O

The SoftLogix controller supports as many communication cards as you have PCI slots in the computer.

Each Logix-based communication module supports a limited number of scheduled and unscheduled connections. Take these limits into account when designing your system:

Device:	Description:	Maximum Connections per Module:
1784-PCICS	SoftLogix ControlNet communication module	127 scheduled connections and 128 unscheduled connections
1788-CNC, -CNCR 1788-CNF, -CNFR	FlexLogix ControlNet communication card	32 scheduled connections (depending on RPI, as many as 22 connections can be scheduled) any remaining connections (or all 32 if you have no scheduled connections) can be used for unscheduled connections
1756-CNB 1756-CNBR	ControlLogix ControlNet communication module	64 total connections, any combination of scheduled and unscheduled
1794-ACN15 1794-ACNR15	FLEX ControlNet adapter module	9 total connections, any combination of scheduled and unscheduled

Accessing I/O

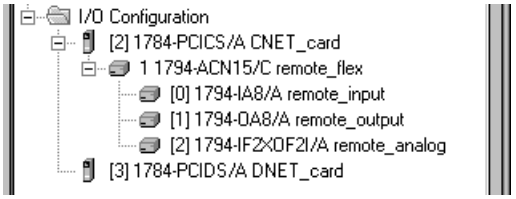
I/O information is presented as a structure of multiple fields, which depend on the specific features of the I/O module. The name of the structure is based on the location of the I/O module in the system. Each I/O tag is automatically created when you configure the I/O module through the programming software. Each tag name follows this format:

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

where:

This address variable:	Is:
Location	Identifies network location ADAPTER_NAME = identifies remote adapter or bridge device
SlotNumber	Slot number of I/O module in its chassis
Type	Type of data I = input O = output C = configuration S = status
MemberName	Specific data from the I/O module; depends on the type of data the module can store For example, Data and Fault are possible fields of data for an I/O module. Data is the common name for values the are sent to or received from I/O points.
SubMemberName	Specific data related to a MemberName.
Bit (optional)	Specific point on the I/O module; depends on the size of the I/O module (0-31 for a 32-point module)

EXAMPLE

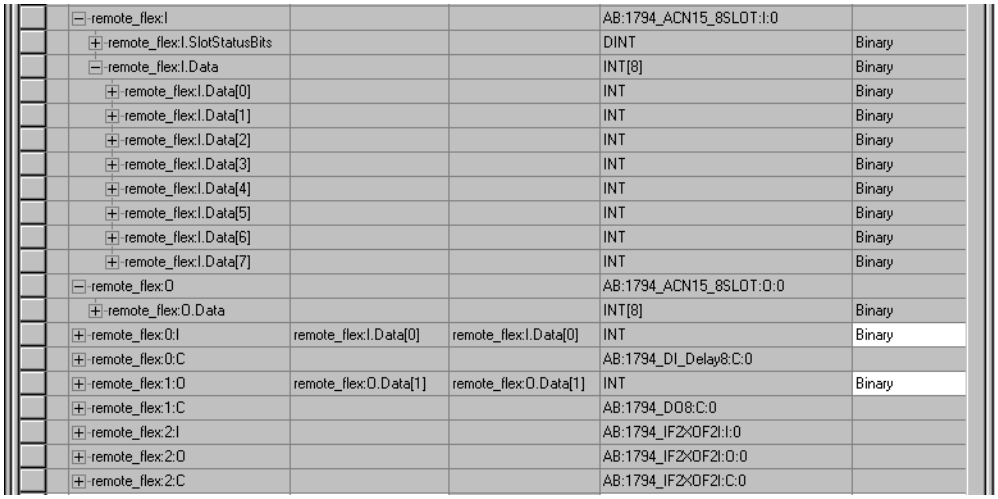


The tags created for the remote device (1794-ACN15 in this example) depend on the communication format you select for that device when you add the device to the I/O Configuration folder.

If you select:	The automatically-created tags are for a:
Rack Optimization	rack-optimized connection to the remote communication device
Listen Only - Rack Optimization	rack-optimized connection to the remote communication device (not available on all communication devices)
None	direct connection to the individual I/O modules with no connection to the remote communication device

Working with a rack-optimized connection

The rack-optimized connection creates a DINT element for each possible I/O module connected to the device “remote_flex.” The array remote_flex:I.Data contains the possible input elements; the remote_flex:O.Data contains the possible output elements.



remote_flex:I			AB:1794_ACN15_8SLOT:I:0	
remote_flex:I.SlotStatusBits			DINT	Binary
remote_flex:I.Data			INT[8]	Binary
remote_flex:I.Data[0]			INT	Binary
remote_flex:I.Data[1]			INT	Binary
remote_flex:I.Data[2]			INT	Binary
remote_flex:I.Data[3]			INT	Binary
remote_flex:I.Data[4]			INT	Binary
remote_flex:I.Data[5]			INT	Binary
remote_flex:I.Data[6]			INT	Binary
remote_flex:I.Data[7]			INT	Binary
remote_flex:O			AB:1794_ACN15_8SLOT:O:0	
remote_flex:O.Data			INT[8]	Binary
remote_flex:0:I	remote_flex:I.Data[0]	remote_flex:I.Data[0]	INT	Binary
remote_flex:0:C			AB:1794_DI_Delay8:C:0	
remote_flex:1:O	remote_flex:O.Data[1]	remote_flex:O.Data[1]	INT	Binary
remote_flex:1:C			AB:1794_DO8:C:0	
remote_flex:2:I			AB:1794_IF2XOF2I:I:0	
remote_flex:2:O			AB:1794_IF2XOF2I:O:0	
remote_flex:2:C			AB:1794_IF2XOF2I:C:0	

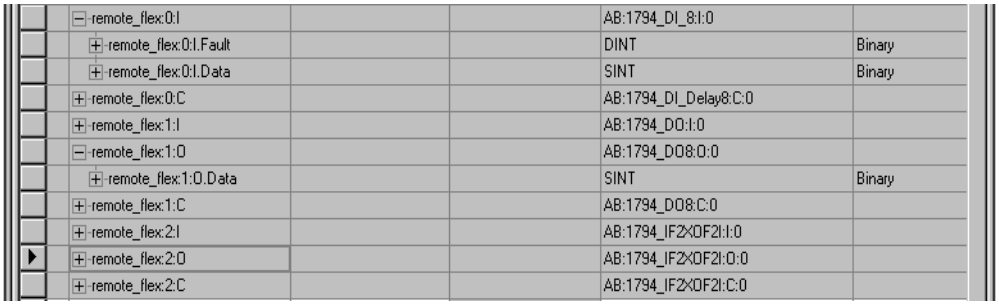
The tags for the individual, digital I/O modules are actually aliases back into the rack-optimized array tag. For example “remote_flex:0:I” is an alias to “remote_flex:I.Data[0]. These digital I/O modules were configured with a rack-optimized communication format to take advantage of the rack-optimized array tag created for the communication device.

The index number on the array element refers to the slot number on “remote_flex.” For example, Data[2] refers to the module in slot 2. You can have only one I/O module in a given slot, so Data[2] is only used in either the input or output array. That same element in the other array still exists even though it does not contain actual data. You can create aliases to the elements you actually use to more readily identify the data you need.

Note that the tags for the analog module (“remote_flex:2:I,” “remote_flex:2:O,” and “remote_flex:2:C”) are not aliases. Analog modules require direct connections to operate. Do not use the element of the rack-optimized array tag to control the analog module. Use the individual, slot-referenced tag.

Working with direct connections

If you select None for the communication format to the communication device, the software assumes that you want a direct connection for each I/O module connected to that device. The software creates slot-referenced tags for each I/O module, but not for the communication device.



Sending Messages

The SoftLogix controller can send MSG instructions to other controllers over a ControlNet link. Each MSG instruction requires you to specify a target and an address within the target. The number of messages that a device can support depends on the type of message and the type of device:

This device:	Support this many unconnected messages:	Support this many connected messages:
1756-CNB module (for a Logix5550 controller)	20	64
1784-PCICS card (for a SoftLogix controller)	50	128
1788-CNx daughtercard (for a FlexLogix controller)	5	32
ControlNet PLC-5 controller	32	32

MSG instructions over ControlNet are unscheduled connections. The type of MSG determines whether or not it requires a connection. If the MSG instruction requires a connection, it opens the needed connection when it is executed. You can configure the MSG instruction to keep the connection open (cache) or to close it after sending the message.

Sending MSGs to other controllers

Table 4.A Message to a Logix controller

If you want to:	For this item:	Type or select:
read (receive) data	Message Type	CIP Data Table Read
	Source Element	first element of the tag that contains data in the other controller
	Number of Elements	number of elements to transfer
	Destination Tag	first element of the tag (controller-scope) in this controller for the data
write (send) data	Message Type	CIP Data Table Write
	Source Tag	first element of the tag (controller-scope) in this controller that contains the data
	Number of Elements	number of elements to transfer
	Destination Element	first element of the tag for the data in the other controller

Table 4.B Message to a SLC 500 controller

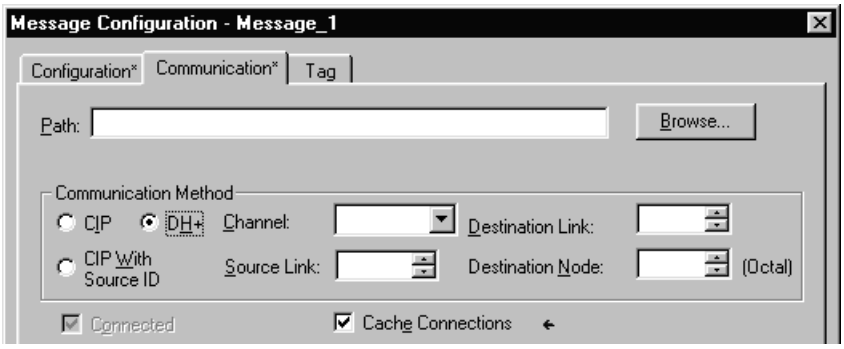
If the data is:	And you want to:	For this item:	Type or select:
integer (s)	read (receive) data	Message Type	*SLC Typed Read*
		Source Element	data table address in the SLC 500 controller (e.g., N7:10)
		Number Of Elements	number of integers to transfer
		Destination Tag	first element of `int_buffer`
	write (send) data	Message Type	*SLC Typed Write*
		Source Tag	first element of `int_buffer`
		Number Of Elements	number of integers to transfer
		Destination Element	data table address in the SLC 500 controller (e.g., N7:10)
floating-point (REAL)	read (receive) data	Message Type	*SLC Typed Read*
		Source Element	data table address in the SLC 500 controller (e.g., F8:0)
		Number Of Elements	number of values to transfer
		Destination Tag	first element of the tag (controller-scoped) in this controller for the data
	write (send) data	Message Type	*SLC Typed Write*
		Source Tag	first element of the tag (controller-scoped) in this controller that contains the data
		Number Of Elements	number of values to transfer
		Destination Element	data table address in the SLC 500 controller (e.g., F8:0)

Table 4.C Message to a PLC-5 controller

If the data is:	And you want to:	For this item:	Type or select:
integer (s)	read (receive) data	Message Type	*PLC5 Typed Read*
		Source Element	data table address in the PLC-5 controller (e.g., N7:10)
		Number Of Elements	number of integers to transfer
		Destination Tag	first element of `int_buffer`
	write (send) data	Message Type	*PLC5 Typed Write*
		Source Tag	first element of `int_buffer`
		Number Of Elements	number of integers to transfer
		Destination Element	data table address in the PLC-5 controller (e.g., N7:10)
floating-point (REAL)	read (receive) data	Message Type	*PLC5 Typed Read*
		Source Element	data table address in the PLC-5 controller (e.g., F8:0)
		Number Of Elements	number of values to transfer
		Destination Tag	first element of the tag (controller-scoped) in this controller for the data
	write (send) data	Message Type	*PLC5 Typed Write*
		Source Tag	first element of the tag (controller-scoped) in this controller that contains the data
		Number Of Elements	number of values to transfer
		Destination Element	data table address in the PLC-5 controller (e.g., F8:0)

Specifying the path to the target device

Once you configure the type of message instruction, you must specify the path of the target device.



In the path box, type the port from which the message exits the card and then type the address of the next device along the path to the destination:

Where:	For this:	Is:
port	ControlNet port from a ControlNet communication device	2
address	ControlLogix backplane	slot number
	DF1 network	station address (0-254)
	ControlNet network	node number (1-99 decimal)
	DH+ network	8# followed by the node number (1-77 octal) For example, to specify the octal node address of 37, type 8#37.
	EtherNet/IP network	You can specify a module on an EtherNet/IP network using any of these formats: IP address (e.g., 130.130.130.5) IP address:Port (e.g., 130.130.130.5:24) DNS name (e.g., tanks) DNS name:Port (e.g., tanks:24)

If the message goes through multiple devices, follow the same pattern of specifying the port the message exits the communication port of the device and the address of the next device in the path.

For example, a valid path could be 1, 5, 2, 12, 1, 3 where:

This value:	Specifies:
1	the SoftLogix virtual chassis
5	the sending device (the communication card in the controller's computer) is in slot 5 of the virtual chassis
2	sending the message out the ControlNet communication port
12	the node number of the target device
1	the backplane of the chassis that contains the target device
3	the target device is in slot 3 of the remote chassis

Producing and Consuming Data

The SoftLogix controller supports the ability to produce (multicast) and consume (receive) system-shared tags over a ControlNet link. Produced data is accessible by multiple controllers over a ControlNet network. Produced and consumed data are scheduled connections because the controller sends or receives data at a predetermined rate.

Produced and consumed tags must be controller-scoped tags of DINT or REAL data type, or in an array or structure.

Tag type:	Description:	Specify:
produced	These are tags that the controller produced for other controllers to consume.	• Enabled for producing • How many consumers allowed
consumed	These are tags whose values are produced by another controller.	• Controller name that owns the tag that the local controller wants to consume • Tag name or instance that the controller wants to consume • Data type of the tag to consume • Update interval of how often the local controller consumes the tag

The producer and consumer must be configured correctly for the specified data to be shared. A produced tag in the producer must be specified exactly the same as a consumed tag in the consumer. A produced tag must also be controller-scoped.

If any produced/consumed tag between a producer and consumer is not specified correctly, none of the produced/consumed tags for that producer and consumer will be transferred. However, other consumers can still access their shared tags, as long as their tags are specified correctly. One consumer failing to access shared data does not affect other consumers accessing the same data.

Maximum number of produced and consumed tags

The maximum number of produced/consumed tags that you can configure depends on the connection limits of the communication device that transfers the produced/consumed data.

Each produced tag uses one connection for the tag and the first configured consumer of the tag. Each consumer thereafter uses an additional connection.

Size limit of a produced or consumed tag

A produced or consumed tag can be as large as 488 bytes, but it must also fit within the bandwidth of the ControlNet network:

- As the number of connections over a ControlNet network increases, several connections, including produced or consumed tags, may need to share a network update.
- Since a ControlNet network can only pass 500 bytes in one update, the data of each connection must be less than or equal to 488 bytes to fit into the update.

If a produced or consumed tag is too large for your ControlNet network, make one or more of the following adjustments:

- Reduce the Network Update Time (NUT). At a faster NUT, less connections are able to share an update slot.
- Increase the Requested Packet Interval (RPI) of all connections. At a higher RPI, connections can take turns sending data during an update slot.
- For a ControlNet bridge module in a remote chassis, select the most efficient communication format for that chassis:.

Are most of the modules in the chassis non-diagnostic, digital I/O modules?	Then select this communication format for the remote communication module:
yes	rack optimization
no	none

  The Rack Optimization format uses an additional 8 bytes for each slot in its chassis. Analog modules or modules that are sending or getting diagnostic, fuse, or timestamp data require direct connections and cannot take advantage of the rack optimized form. Selecting “None” frees up the 8 bytes per slot for other uses, such as produced or consumed tags.

- Separate the tag into two or more smaller tags:
  - Group the data according to similar update rates. For example, you could create one tag for data that is critical and another tag for data that is not as critical.
  - Assign a different RPI to each tag.
- Create logic to transfer the data in smaller sections (packets).

Producing a tag

Produced data can be:

- tag of DINT or REAL data type.
- array of DINT or REAL elements.
- user-defined structure with any type elements. Use a user-defined structure to group BOOL, SINT, and INT data.

To create a produced tag:

1. You must be programming offline.

2. In the controller organizer, double-click the Controller Tags folder and then click the Edit Tags tab (at the bottom of the window).

3. Select the tag that you want to produce, or right-click to enter a new tag, and display the Tag Properties dialog box.

4. Select the “Produce this tag” check box. Specify how many controllers can consume the tag.

You can produce a base, alias, or consumed tag.

The consumed tag in a receiving controller must have the same data type as the produced tag in the originating controller. The controller performs type checking to ensure proper data is being received.

Produced tags require connections. The number of connections depend on the amount of data and how many controllers are producing and consuming tags.

Consuming a tag

A consumed tag represents data that is produced by one controller and received and stored by the consuming controller. To create a consumed tag:

1. You must be programming offline.
2. In the controller organizer, double-click the Controller Tags folder and then click the Edit Tags tab.
3. Select the tag that you want to consume, or enter a new tag, and display the Tag Properties dialog box.
4. Specify:

In this field:	Type or select:
Tag Type	Select Consumed.
Controller	Select the name of the other controller. You must have already created the controller in the controller organizer for the controller name to be available.
Remote Tag Name Remote Instance	Type a name for the tag in the other controller you want to consume. **Important:** The name must match the name in the remote controller exactly, or the connection faults. If the remote controller is a ControlNet PLC-5, this field is Remote Instance. Select the instance number (1-128) of the data on the remote controller.
RPI (requested packet interval)	Type the amount of time in msec between updates of the data from the remote controller. The local controller will receive data at least this fast.
Display Style	If you are creating a consumed tag that refers to a tag whose data type is BOOL, SINT, INT, DINT, or REAL, you can select a display style. This display style defines how the tag value will be displayed in the data monitor and ladder editor. The display style does not have to match the display style of the tag in the remote controller.

All consumed tags are automatically controller-scope.

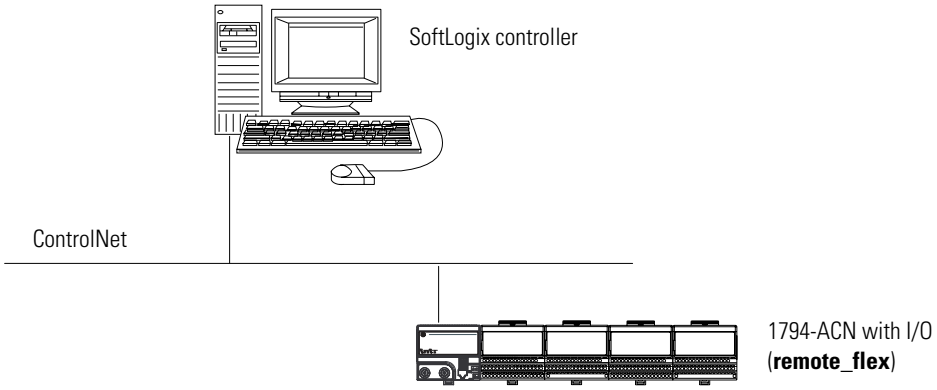
To consume data from a remote controller, use RSNetWorx software to schedule the connection over the ControlNet network.

The produced tag in the originating SoftLogix controller must have the same data type as the consumed tag in the other SoftLogix controller. The SoftLogix controller performs type checking to ensure proper data is being received.

IMPORTANT If a consumed-tag connection fails, all of the other tags being consumed from that remote controller stop receiving data.

Example 1: SoftLogix Controller and ControlNet I/O

In the following example, one SoftLogix controller controls I/O through a 1794-ACN15 module.



Example 1: Controlling I/O

This example has the SoftLogix controller controlling the I/O connected to the remote 1794-ACN15 module. The data the SoftLogix controller receives from the I/O modules depends on how you configure the I/O modules. You can configure each module as a direct connection or as a rack-optimized connection. One location (chassis or DIN rail) can have a combination of some modules configured as a direct connection and others as rack optimized.

Example 1: Total connections required by the SoftLogix controller

The following table calculates the connections used in this example.

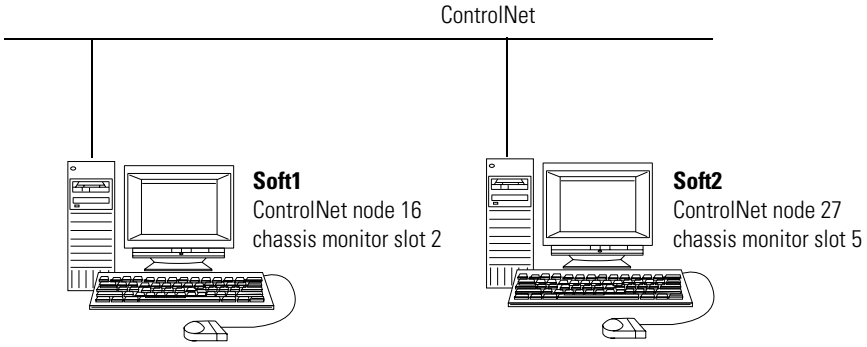
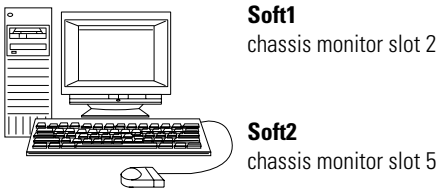
Connection:	Amount:
SoftLogix controller to 1784-PCICS card	0
SoftLogix controller to remote 1794-ACNR15 (communication format is “none”)	0
SoftLogix controller to 4 I/O modules (through 1794-ACNR15) all modules configured as direct connection	4
total connections used:	4

If you configure the 1794-ACNR15 as rack-optimized and the I/O modules as rack-optimized, you only use one connection to the 1794-ACN15 module, reducing the above example by 3 connections. The following table calculates the connections for this rack-optimized configuration.

Connection:	Amount:
SoftLogix controller to 1784-PCICS card	0
SoftLogix controller to remote 1794-ACNR15 (communication format is "rack optimization")	1
SoftLogix controller to 4 I/O modules (through 1794-ACNR15) all modules configured as rack optimized connections	0
total connections used:	1

Example 2: SoftLogix Controller to SoftLogix Controller

In the following example, one SoftLogix controller communicates with another SoftLogix controller over ControlNet. The two controllers can be in separate computers or in the same computer.

Example:	Illustration:
Each SoftLogix controller resides in its own computer	
Each SoftLogix controller resides in the same computer	

Example 2: Sending a MSG instruction

To send a MSG from Soft1 to Soft2:

1. For Soft1, create a controller-scoped tag and select the MESSAGE data type.

2. Enter a MSG instruction.

   In this example logic, a message is sent when a specific condition is met. When count_send is set, send count_msg.

3. Configure the MSG instruction. On the Configuration tab:

For this item:	Specify:
Message Type	CIP Data Table Read or CIP Data Table Write
Source Tag	Tag containing the data to be transferred
Number of Elements	Number of array elements to transfer
Destination Tag	Tag to which the data will be transferred

4. On the Communication tab, specify the communication path.

   A communication path requires pairs of numbers. The first number in the pair identifies the port from which the message exits. The second number in the pair designates the node address of the next device.

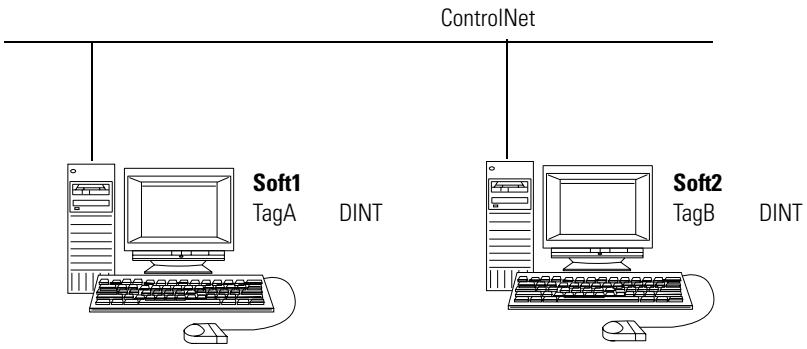
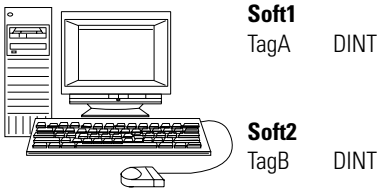
For this item:	Specify:
Communication Path (each SoftLogix controller resides in its own computer)	1,2,2,27,1,5 where: 1 is the SoftLogix backplane of Soft1 2 is 1784-PCICS card in slot 2 2 is the ControlNet port 27 is the ControlNet node of Soft2 1 is the SoftLogix backplane of Soft2 5 is the controller slot of Soft2
Communication Path (each SoftLogix controller resides in the same computer)	1,5 where: 1 is the SoftLogix backplane of Soft1 5 is the controller slot of Soft2

Example 2: Producing and consuming tags

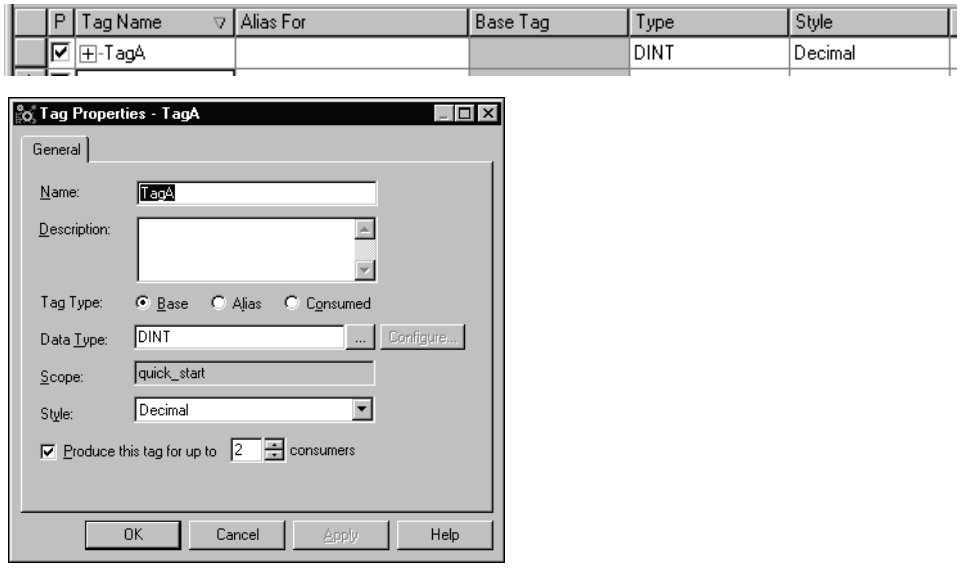
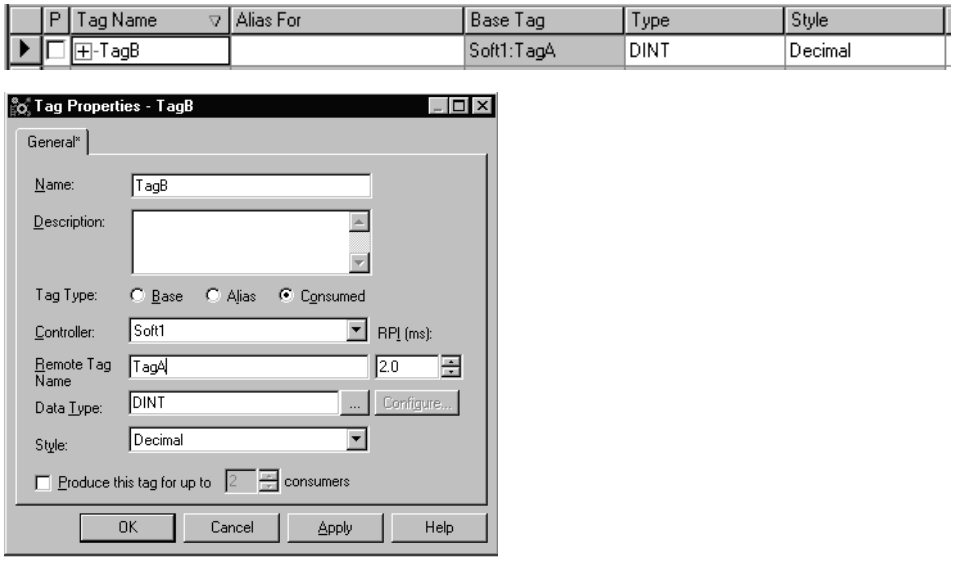
You can produce a base, alias, or consumed tag. Produced data can be:

- tag of DINT or REAL data type.
- array of DINT or REAL elements.
- user-defined structure with any type elements. Use a user-defined structure to group BOOL, SINT, and INT data.

The consumed tag must have the same data type as the produced tag in the originating controller. The controller performs type checking to ensure proper data is being received.

Example:	Illustration:
Each SoftLogix controller resides in its own computer	
Each SoftLogix controller resides in the same computer, using different CPUs	

This example shows Soft1 as producing TagA. Soft2 consumes TagA and stores is as TagB:

In this controller:	The tags look like:
Soft1	
Soft2	

Each produced tags requires one connection for the producing controller and an additional connection for each consuming controller. Each consumed tag requires one connection.

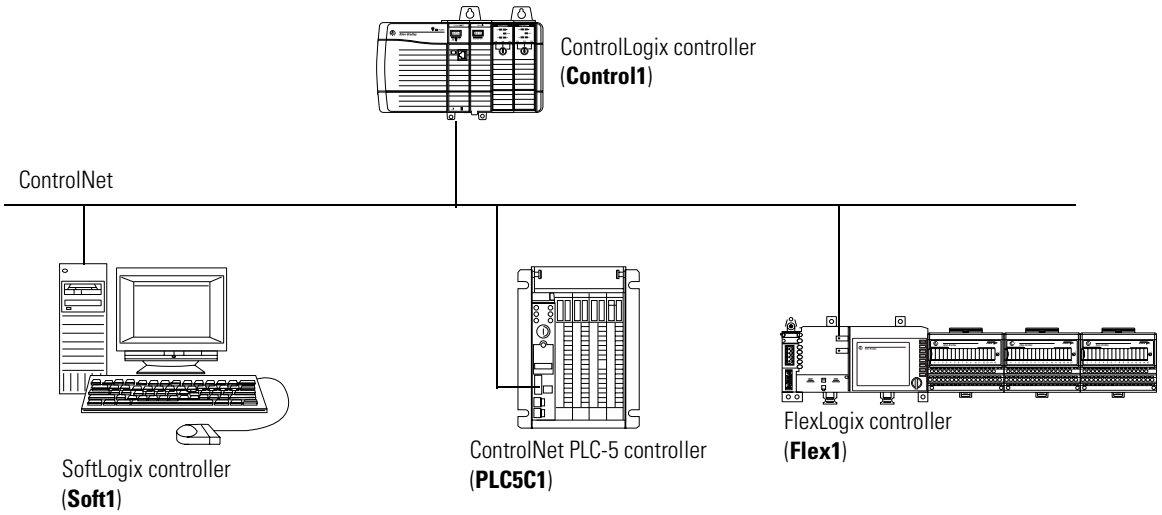
Example 2: Total connections required by Soft1 controller

The following table calculates the connections used in this example.

Connection:	Amount:
Soft1 controller to 1784-PCICS card	0
Soft1 controller to remote 1784-PCICS card	0
connected, cached MSG from Soft1 to Soft2	1
produced TagA produced from Soft1 to Soft2 other consumer (2 are configured)	 1 1
consumed TagB	1
total connections used:	4

Example 3: SoftLogix Controller to Other Devices

In the following example, one SoftLogix controller communicates with other controllers over ControlNet.



Example 3: Sending MSG instructions

You configure a MSG instruction to a ControlLogix and FlexLogix controller the same as you do for a SoftLogix controller. All Logix-based controllers follow the same MSG configuration requirements. See Example 2 above.

Configuring a MSG instruction for a PLC-5 controller depends on the originating controller.

For MSG instructions originating from the SoftLogix controller to the ControlNet PLC-5 controller:

Type of Logix MSG instruction:	Source:	Destination:
Typed Read	any integer element (such as B3:0, T4:0.ACC, C5:0.ACC, N7:0, etc.)	SINT, INT, or DINT tag
	any floating point element (such as F8:0, PD10:0.SP, etc.)	REAL tag
Typed Write	SINT or INT tag	any integer element (such as B3:0, T4:0.ACC, C5:0.ACC, N7:0, etc.)
	REAL tag	any floating point element (such as F8:0, PD10:0.SP, etc.)
Word Range Read	any data type (such as B3:0, T4:0, C5:0, R6:0, N7:0, F8:0, etc.)	SINT, INT, DINT, or REAL
Word Range Write	SINT, INT, DINT, or REAL	any data type (such as B3:0, T4:0, C5:0, R6:0, N7:0, F8:0, etc.)

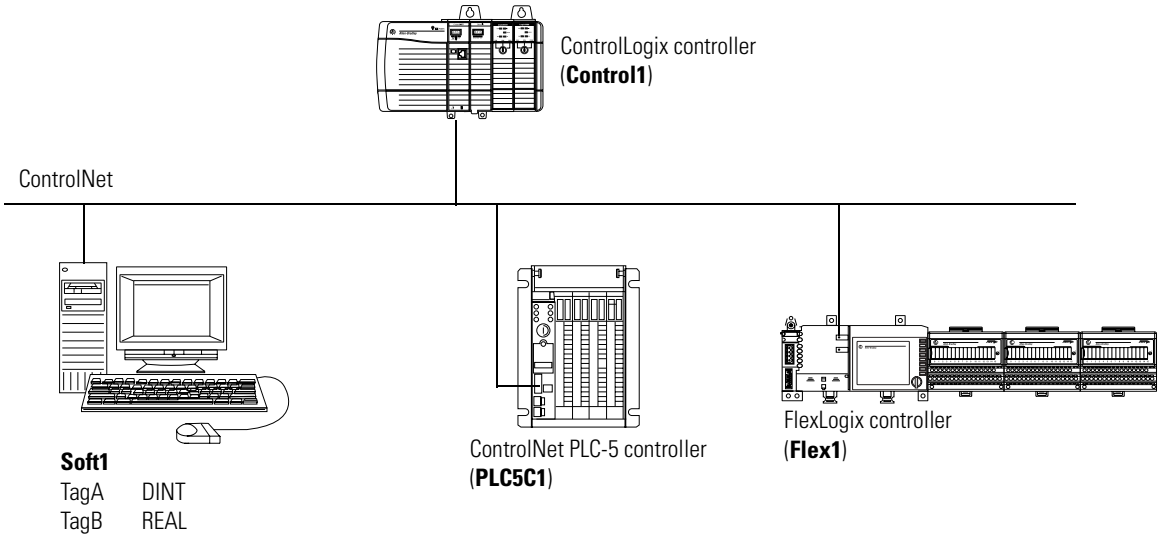
The PLC-5 controller supports logical ASCII addressing so you do not have to map a compatibility file for MSG instructions initiated by a PLC-5 controller. Place the SoftLogix tag name in double quotes (“).

Type of MSG Instruction:	Example Source and Destination:	
PLC-5 writes to SoftLogix	source element	*N7:10*
	destination tag	*“array_1”*
PLC-5 reads from SoftLogix	source tag	*“array_1”*
	destination element	*N7:10*

Example 3: Producing and consuming tags

You can produce and consume tags with any Logix controller the same as you do with a SoftLogix controller. All Logix controllers follow the same requirements for producing and consuming tags. See Example 2 above.

Producing and consuming tags with a ControlNet PLC-5 controller depends on the type of data.



Producing a tag to a ControlNet PLC-5 controller

To produce a tag that a ControlNet PLC-5 controller can consume:

1. Determine the type of data to produce?

If:	And you are producing:	Then:
INT	na	A. Create a user-defined data type that contains an array of INTs with an even number of elements, such as INT[2]. When you produce INTs, you must produce two or more. B. Create a produced tag and select the user-defined data type you created.
DINT or REAL	Only one DINT or REAL value	Create a produced tag and select the DINT or REAL data type, as appropriate.
	More than one DINT or REAL	A. Create a user-defined data type that contains an array of DINTs or REALs, as appropriate. B. Create a produced tag and select the user-defined data type you created.

2. In RSNetWorx software, open the ControlNet configuration for the target ControlNet PLC-5 controller, insert a Receive Scheduled Message and enter the following Message size:

If the produced tag contains:	Then, for the Message size, enter:
INTs	The number of integers in the produced tag
DINTs	Two times the number of DINTs or REALs in the produced tag. For example, if the produced tag contains 10 DINTs, enter 20 for the Message size.
REALs	

3. In the RSNetWorx software, reschedule (save) the network.

The ControlNet PLC-5 controller does not perform type checking. Make sure the PLC-5 data type can correctly receive the SoftLogix produced tag to ensure proper data is being received.

When a ControlNet PLC-5 controller consumes a tag that is produced by a Logix5000 controller, it stores the data in consecutive 16-bit integers. The ControlNet PLC-5 controller stores floating-point data, which requires 32-bits regardless of the type of controller, as follows:

- The first integer contains the upper (left-most) bits of the value.
- The second integer contains the lower (right-most) bits of the value.

To re-construct the floating point data within the ControlNet PLC-5 controller, first reverse the order of the integers and then copy them to a floating-point file.

Consuming a tag from a ControlNet PLC-5 controller

To consume a tag from a ControlNet PLC-5 controller,:

1. In RSNetWorx software, open the ControlNet configuration of the ControlNet PLC-5 controller, insert a Send Scheduled Message.

2. In RSLogix 5000 software, add the ControlNet PLC-5 controller to the Controller Organizer.

3. Create a user-defined data type that contains these members:

Data type:	Description:
DINT	Status
INT[x], where "x" is the output size of the data from the ControlNet PLC-5 controller. (If you are consuming only one INT, no dimension is required.)	Data produced by a ControlNet PLC-5 controller

4. Create a consumed tag with the following properties:

For this tag property:	Type or select:
Tag Type	Consumed
Controller	The ControlNet PLC-5 that is producing the data
Remote Instance	The message number from the ControlNet configuration of the ControlNet PLC-5 controller
RPI	A power of two times the NUT of the ControlNet network. For example, if the NUT is 5ms, select an RPI of 5, 10, 20, 40, etc.
Data Type	The user-defined data type that you created.

5. In the RSNetWorx for ControlNet software, reschedule (save) the network.

Example 3: Total connections required by Soft1

The following table calculates the connections used in this example.

Connection:	Amount:
Soft1 controller to 1784-PCICS card	0
Soft1 controller to remote 1756-CNB module	0
Soft1 controller to remote 1788-CNC card	0
Soft1 controller to remote PLC5C1	1
connected, cached MSG from Soft1 to Control1	1
connected, cached MSG from Soft1 to Flex1	1
connected, cached MSG from Soft1 to PLC5C1	1
produced TagA produced from Soft1 to Control1 consumed by PLC5C1	 1 1
consumed TagB from Control1	1
total connections used:	7

The remote 1756-CNB and 1788-CNC card are configured as “none” for the communication format, so the SoftLogix controller would require a direct connection for any I/O modules connected to these devices that you want in the configuration for the SoftLogix controller.

Example 4: Using SoftLogix as a Gateway

The SoftLogix controller supports bridging over a ControlNet network.

Any SoftLogix MSG instruction that bridges one network has multiple pairs of numbers in its communication path. To construct a communication path:

1. Specify the port where the message exits.

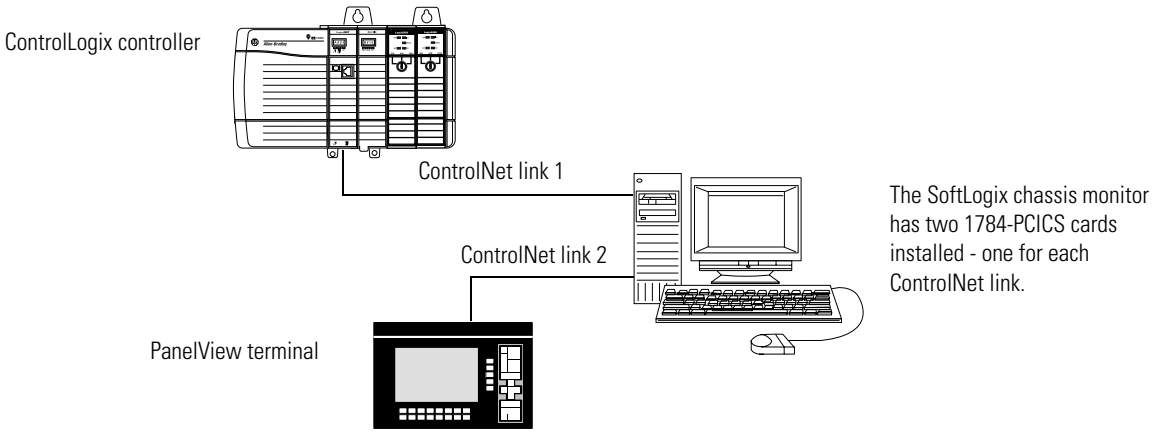
For this port:	Specify:
backplane port	1
DF1 port from the controller	2
ControlNet port from a communication card/module	
Ethernet port from a communication card/module	
DH+ port over channel A from a 1756-DHRIO module	
DH+ port over channel B from a 1756-DHRIO module	3

2. Specify the next device.

For a device on a:	Specify:
ControlLogix backplane	slot number
DF1 network	station address (0-254)
ControlNet network	node number (1-99 decimal)
DH+ network	node number (1-77 decimal)
Ethernet network	IP address

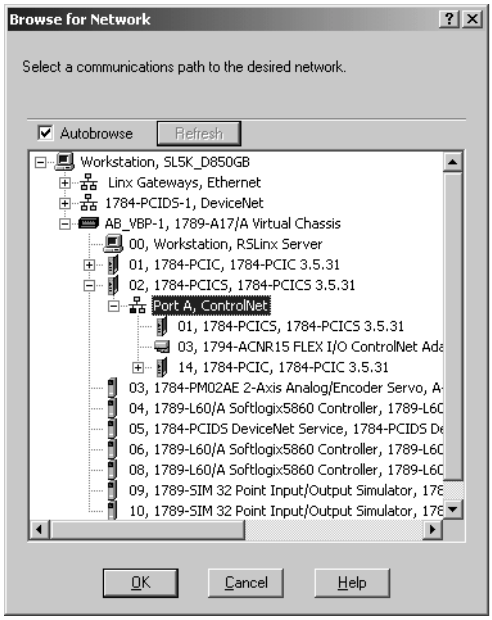
3. Repeat steps 1 and 2 until you specify the target device.

In the following example, the ControlLogix controller can remotely access a PanelView terminal over ControlNet links. The SoftLogix chassis monitor resides on the computer. A SoftLogix controller is not required for the gateway - you only need a 1784-PCICS card for each ControlNet link.



If you want to select the 1784-PCICS card from an online list of available devices, such as Browse Network in RSNetWorx for ControlNet software, select the 1784-PCICS card in the virtual chassis.

Select the 1784-PCICS card from within the virtual chassis. →



Notes:

Communicating with Device on an Ethernet Link

Using This Chapter

With the version 13 SoftLogix controller, the SoftLogix5800 EtherNet/IP module supports both messaging and I/O at the same time.

Configuring Your System for an Ethernet Link

For the SoftLogix controller to operate on an EtherNet/IP network, you need:

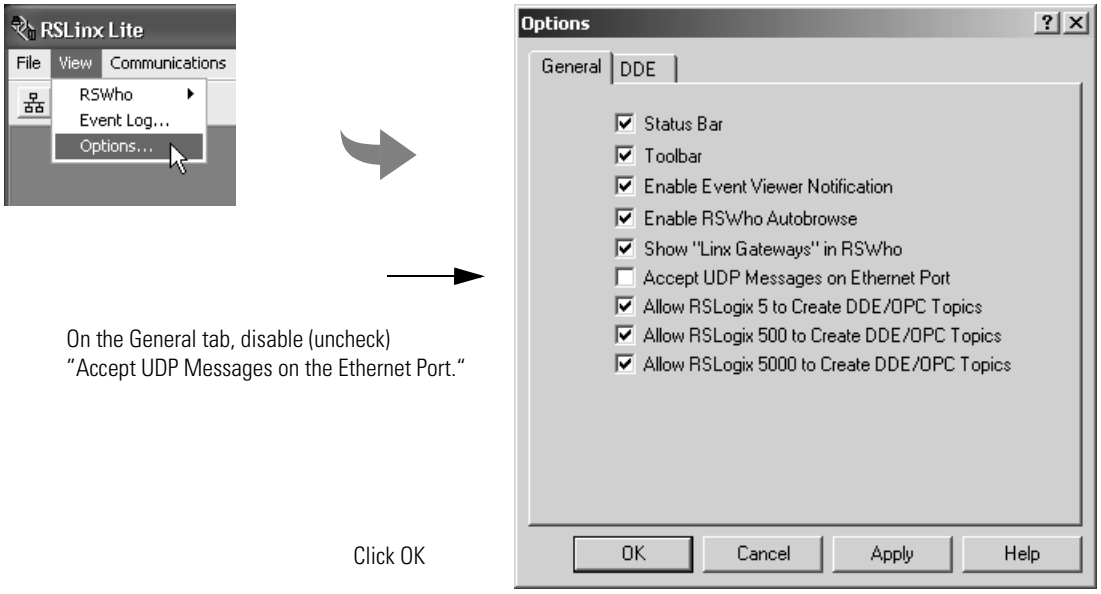
- the computer where the SoftLogix controller resides must have an Ethernet communication port.
- the computer where the RSLogix 5000 programming software resides must have an Ethernet communication port.
- RSLinx software
- RSLogix 5000 programming software

You can use any commercially-available Ethernet port that meets the system requirements for running a SoftLogix controller on your computer. Use the Ethernet driver that comes with the device.

Step 1: Disable UDP messages in RSLinx

To send messages or control I/O, you must change the RSLinx configuration so that it does not accept UDP messages.

1. From RSLinx software on the computer with the controller, select the General tab under Options.



On the General tab, disable (uncheck) "Accept UDP Messages on the Ethernet Port."

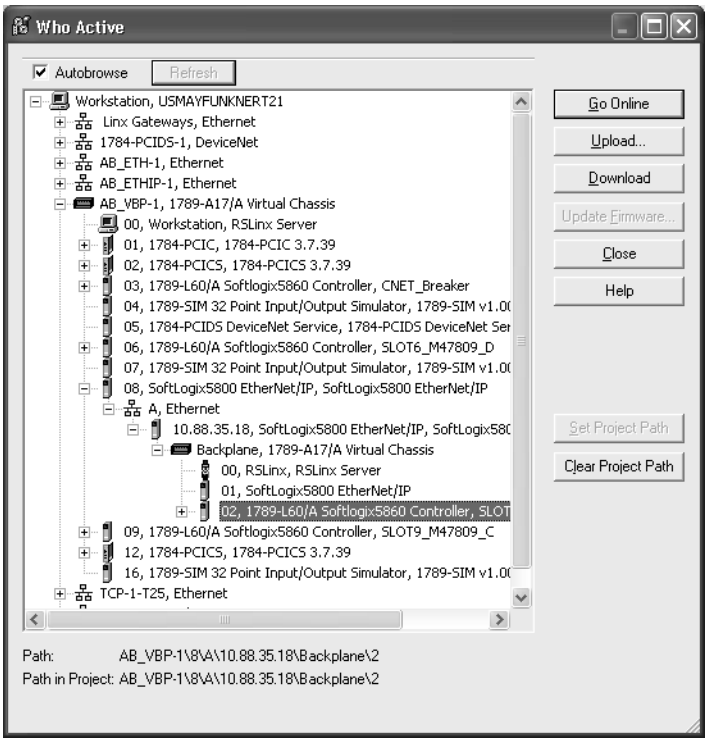
Click OK

Disabling the UDP option

Disabling the UDP option lets RSLinx software and the SoftLogix5800 EtherNet/IP functionality coexist on the same PC. Disabling the UDP option also disables RSLinx software's Gateway functionality. RSLinx software still functions, but the Gateway options are removed while RSLinx software continues to display that it has a full Gateway activation. This affects how remote computers can browse through a local computer with UDP disabled.

If UDP is disabled on a local computer, a remote computer browsing through the local computer has this functionality:

- If the local computer has a DeviceNet module in the virtual chassis, you **cannot** remotely browse the DeviceNet network. Replacing RSLinx Lite with RSlinx Gateway on the local or remote computer **does not** enable remote browsing of the DeviceNet network.
- If the local computer has a ControlNet module in the virtual chassis, you can remotely browse the ControlNet network.
- If the local computer has a SoftLogix controller in the virtual chassis, you can browse out the serial port of the computer
- If the local computer has an EtherNet/IP module in the virtual chassis, you can browse the EtherNet/IP network configured to that module.



- The local computer supports a remote connection to the SoftLogix controller over an Ethernet link so you can remotely progams the SoftLogix controller.

If UDP is disabled on a local computer, you can browse out all networks and see configured devices, as long as the appropriate module is installed in the virtual chassis on the local computer.

Enabling the UDP option

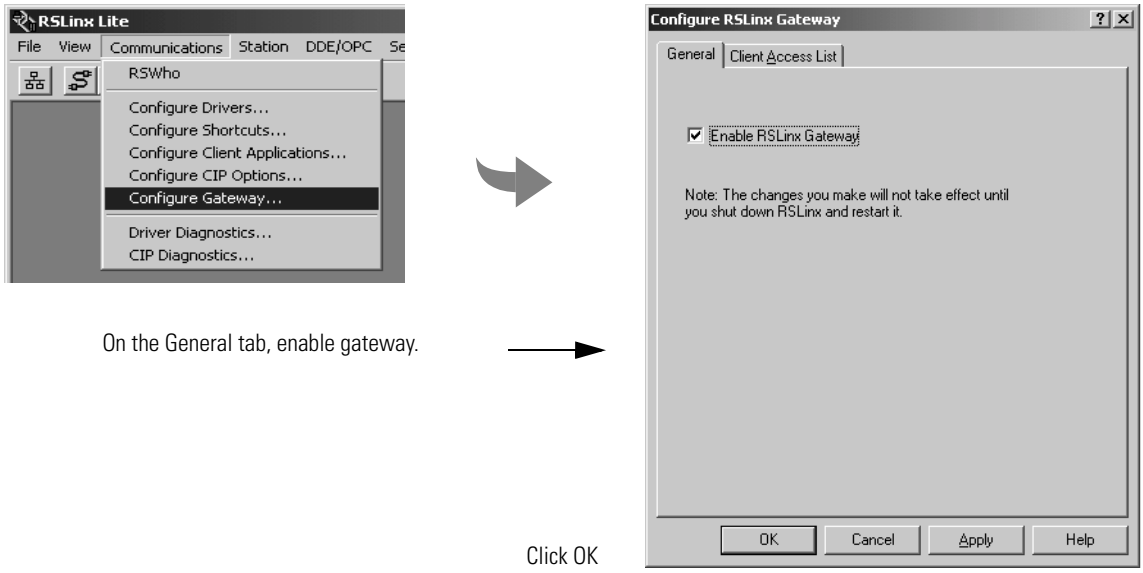
Enabling the UDP option lets the RSLinx Gateway functionality operate as expected. Installing a Softlogix5800 EtherNet/IP module into a system with the UDP option enabled and RSLinx Gateway functionality enabled, causes the SoftLogix EtherNet/IP module red X in RSLinx and the chassis monitor. This does not affect the operation of the RSLinx software. It does prohibit the SoftLogix controller from sending/receiving messages and controlling I/O.

Enable the UDP option (and the RSLinx Gateway functionality - see the next page) when you need the computer to configure or commission ControlNet and DeviceNet networks and devices. With the UDP option enabled, you have this functionality in the computer where the controller resides:

- To browse a DeviceNet network., you must browse from the 1784-PCIDS driver in RSLinx software.
- If there is a ControlNet module in the virtual chassis, you can remotely browse the ControlNet network.
- If there is a SoftLogix controller in the virtual chassis, you can browse out the serial port of the computer.
- If there is an EtherNet/IP module in the virtual chassis, it will not function as expected. You **cannot** control I/O or send messages.
- The local computer supports a remote connection to the SoftLogix controller over an Ethernet link so you can remotely progams the SoftLogix controller.

If you want the UDP option enabled, you should also enable the Gateway functionality within RSLinx software.

1. From RSLinx software on the computer with the controller, enable RSLinx gateway.

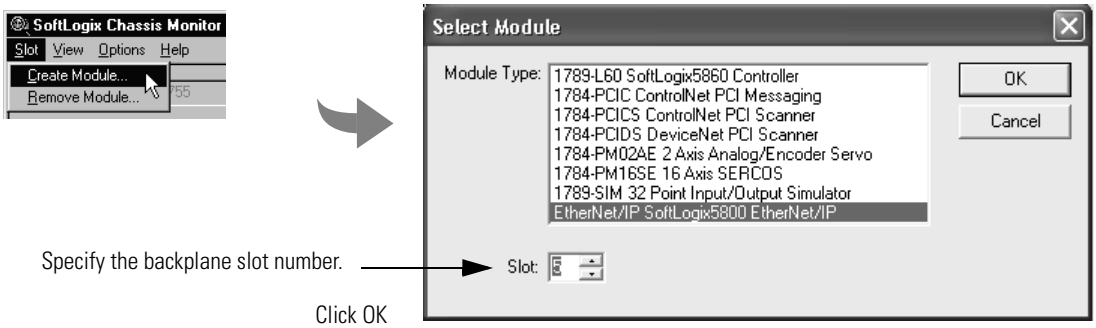


In many applications, initially leave UDP enabled so you can configure ControlNet and DeviceNet networks. After this work is done, disable UDP so the SoftLogix controller can have EtherNet/IP functionality. If you change the UDP setting, you must restart RSLinx software for the change to take affect. To restart RSLinx software, you can reboot the computer (recommended) or you can stop RSLinx software and then restart it. If you choose to stop RSLinx software, you must stop any applications that use RSLinx software, such as RSLogix 5000 software or network configuration software, before you can stop RSLinx software.

Step 2: Create the EtherNet/IP module in the virtual chassis

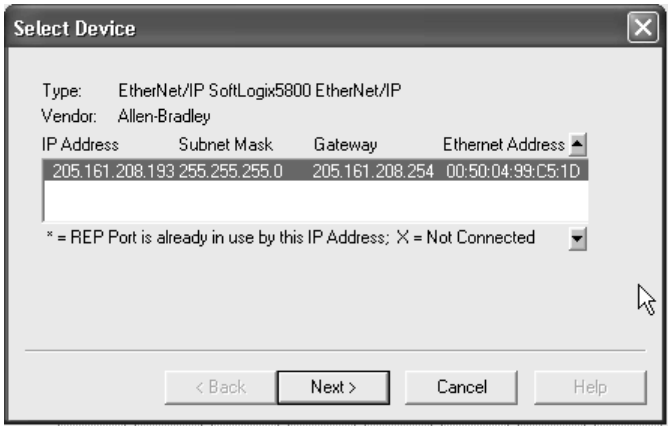
If you are controlling I/O or sending messages over Ethernet, add the EtherNet/IP module to the SoftLogix virtual chassis.

1. From the SoftLogix chassis monitor, select Slot → Create Module or right click the appropriate slot and select Create. Select EtherNet/IP Messaging.



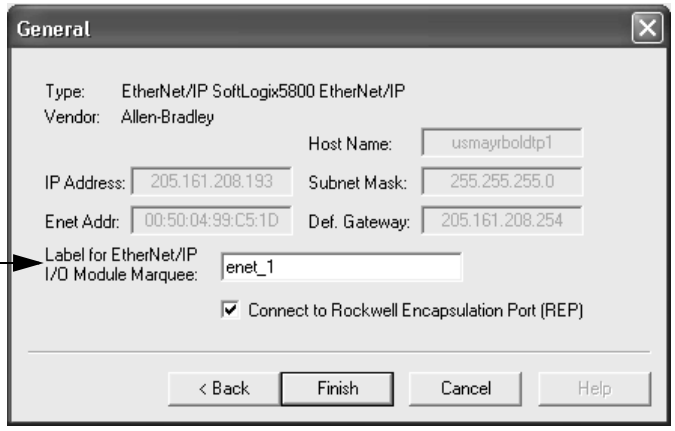
2. Select the serial number of the port you want.

If you previously had an Ethernet port configured in this slot, the chassis monitor remembers the configuration of that previous port.



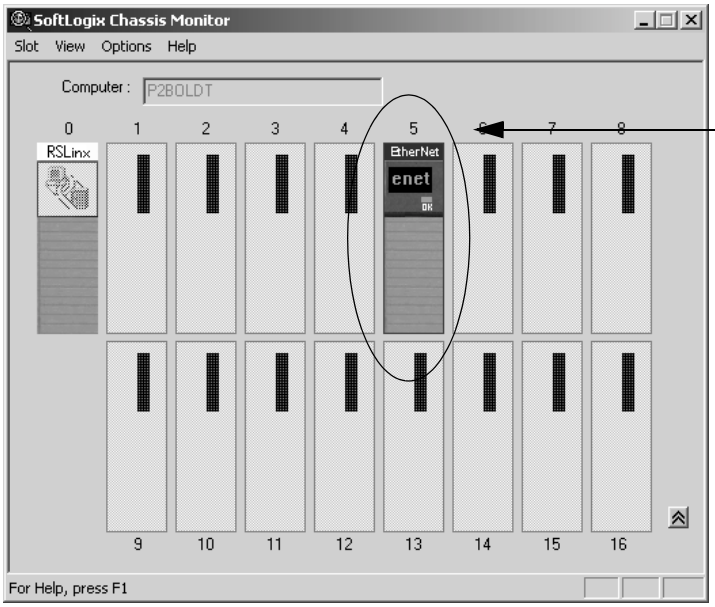
3. Specify a label (marquee) for the Ethernet port.

Enter the label. The IP address, status, and the text you enter here scrolls across the front of the module.



You can specify any slot number greater than 0 for the module. RSLinx software resides in slot 0.

The chassis monitor shows the selected IP address as a virtual module in the SoftLogix chassis.



Adding multiple EtherNet/IP modules to the virtual chassis

If the computer has multiple IP addresses, you can install multiple Ethernet modules in the virtual chassis. Select the appropriate IP address when you configure the EtherNet/IP module in the virtual chassis.

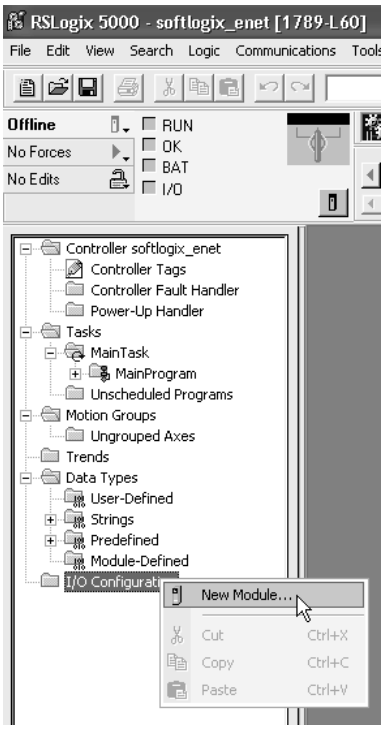
The configuration you select for the UDP option in RSLinx software applies to every IP address in the same computer. RSLinx does not support separate UDP configurations per IP address or actual Ethernet device. If you enable UDP functionality in RSLinx software, RSLinx consumes to all IP addresses on the computer.

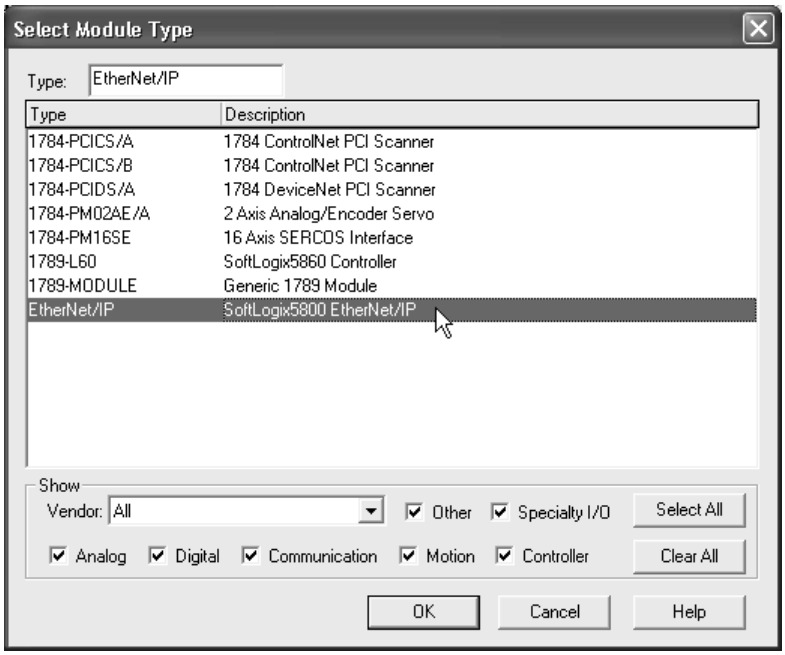
Step 3: Configure the EtherNet/IP module as part of the project

If you plan to control I/O over an EtherNet/IP link, use RSLogix 5000 programming software to add the SoftLogix5800 EtherNet/IP module to the I/O Configuration folder.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.

2 Right-click to select New Module and add a SoftLogix5800 EtherNet/IP.

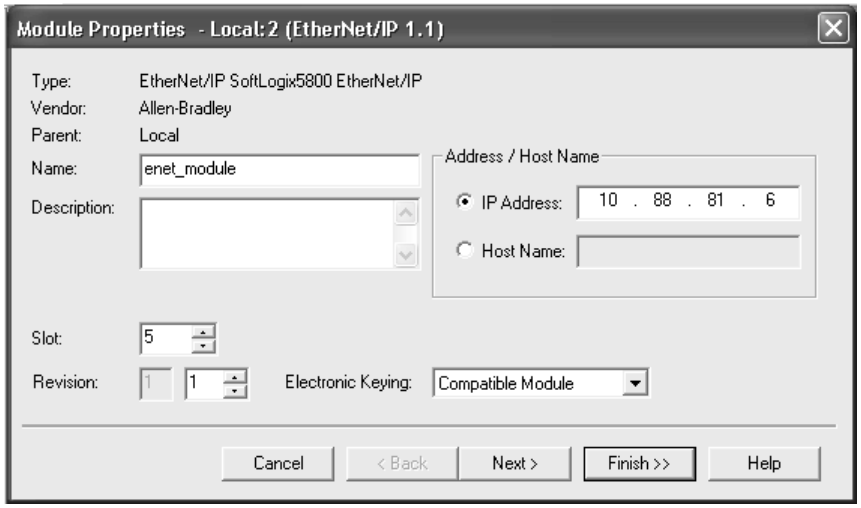




Click OK

3. Specify the appropriate communication port settings.

This must be the same slot number you specified on the SoftLogix chassis monitor.



Configuring a SoftLogix5800 EtherNet/IP module is similar to configuring a 1756-ENBT module in a ControlLogix project, except for these differences:

- If the SoftLogix5800 EtherNet/IP module is in the same virtual chassis as the SoftLogix controller, you do not have to enter an IP address. The configuration defaults to the IP address of the computer.

  If there are multiple IP addresses or if the SoftLogix5800 EtherNet/IP module is not in the same virtual chassis as the SoftLogix controller, you must enter an IP address. This is similar to configuring a 1756-ENBT module.

- A 1756-ENBT module requires that you select a communication format. This is not required for a SoftLogix5800 EtherNet/IP module.

Controller Connections Over EtherNet/IP

A Logix system uses a connection to establish a communication link between two devices. Connections can be:

- controller to distributed I/O or remote communication modules
- produced and consumed tags
- messages

All EtherNet/IP connections are unscheduled. An unscheduled connection is a message transfer between controllers that is triggered by the requested packet interval (RPI) or the program (such as a MSG instruction). Unscheduled messaging lets you send and receive data when needed.

Over EtherNet/IP, the SoftLogix controller supports:

- 64 TCP/IP connections
- 128 Logix connections (I/O and information)

Supported functionality of the SoftLogix5800 EtherNet/IP module

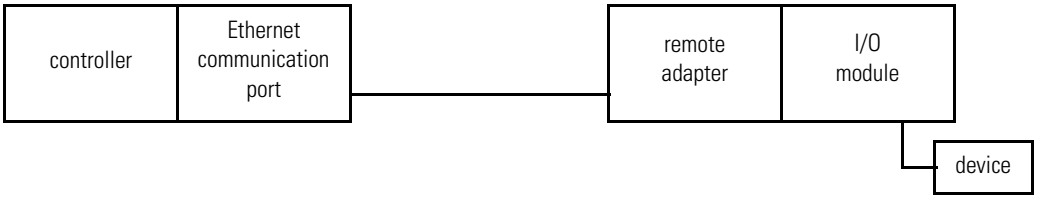
Compared to a 1756-ENBT EtherNet/IP module in a ControlLogix system, the SoftLogix5800 EtherNet/IP module:

- supports the same number of I/O connections
- supports the same number of messaging connections
- supports the same bridging functionality
- does not support a web-based interface
- does not support email via MSG instruction

Configuring Distributed I/O

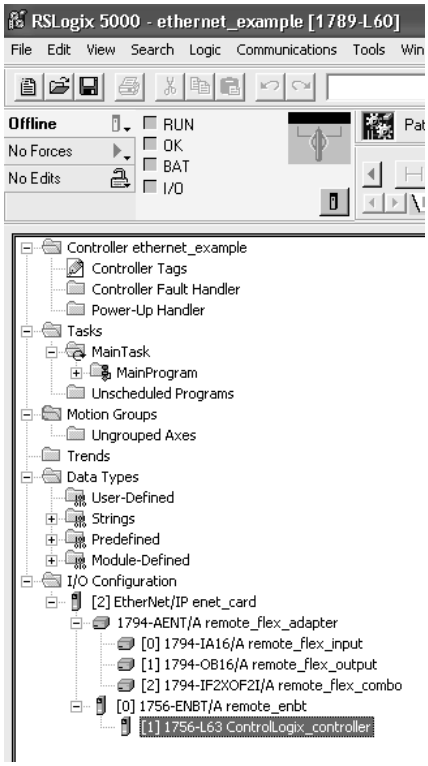
The SoftLogix controller supports distributed I/O over a EtherNet/IP link. Use RSLogix 5000 software to add the SoftLogix5800 EtherNet/IP module for the local controller. Then add a remote adapter and I/O modules to the I/O Configuration folder of the controller project.

For a typical SoftLogix distributed I/O network…



…you build the I/O configuration in this order

1. Add the remote adapter to the SoftLogix5800 EtherNet/IP module of the controller.
2. Add the I/O modules to the remote adapter.



Accessing distributed I/O

I/O information is presented as a structure of multiple fields, which depend on the specific features of the I/O module. The name of the structure is based on the location of the I/O module in the system. Each I/O tag is automatically created when you configure the I/O module through the programming software. Each tag name follows this format:

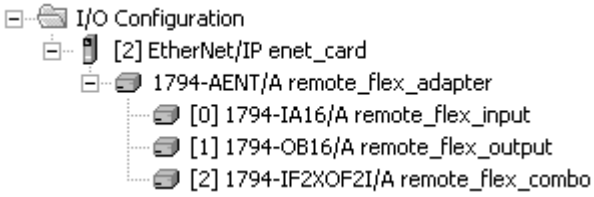
Location:SlotNumber:Type.MemberName.SubMemberName.Bit

where:

This address variable:	Is:
Location	Identifies network location ADAPTER_NAME = identifies remote adapter or bridge
SlotNumber	Slot number of I/O module in its chassis
Type	Type of data I = input O = output C = configuration S = status
MemberName	Specific data from the I/O module; depends on the type of data the module can store For example, Data and Fault are possible fields of data for an I/O module. Data is the common name for values the are sent to or received from I/O points.
SubMemberName	Specific data related to a MemberName.
Bit (optional)	Specific point on the I/O module; depends on the size of the I/O module (0-31 for a 32-point module)

For example

EXAMPLE



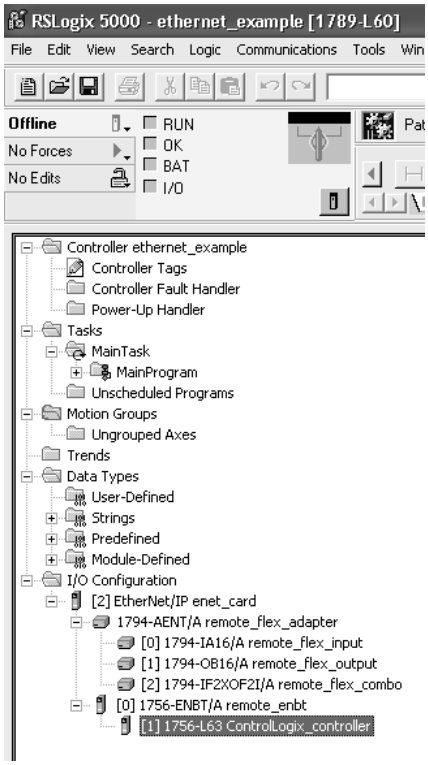
Device:	Example Tag Names (automatically created by the software):
remote adapter "remote_flex_adapter"	remote_flex_adapter:I remote_flex_adapter:I.SlotStatusBits remote_flex_adapter:I.Data remote_flex_adapter:O remote_flex_adapter:O.Data
"remote_flex_input" in slot 0 rack-optimized connection	remote_flex_adapter:0:C remote_flex_adapter:0:C.Config remote_flex_adapter:0:C.DelayTime_0 remote_flex_adapter:0:C.DelayTime_1 remote_flex_adapter:0:C.DelayTime_2 remote_flex_adapter:0:C.DelayTime_3 remote_flex_adapter:0:C.DelayTime_4 remote_flex_adapter:0:C.DelayTime_5 remote_flex_adapter:0:I
"remote_flex_output" in slot 1 rack-optimized connection	remote_flex_adapter:1:C remote_flex_adapter:1:C.SSData remote_flex_adapter:1:O remote_flex_adapter:1:O.Data
"remote_flex_combo" in slot 2 direct connection	remote_flex_adapter:2:C remote_flex_adapter:2:C.InputFIlter remote_flex_adapter:2:C.InputConfiguration remote_flex_adapter:2:C.OutputConfiguration remote_flex_adapter:2:C.RTSInterval remote_flex_adapter:2:C.SSCh0OuputData remote_flex_adapter:2:C.SSCH1OutputData remote_flex_adapter:2:I

Adding a Remote Controller

If you want to add the controller as a remote consumed controller to the I/O configuration, add the controller to the EtherNet/IP module.

To add a remote controller, you build the I/O configuration in this order

1. You add devices to the EtherNet/IP port of the controller.
2. For a controller that requires a communication module, add the module first and then add the controller.



Sending Messages

The SoftLogix controller can send MSG instructions to other controllers and devices over an EtherNet/IP link. Each MSG instruction requires you to specify a target and an address within the target.

MSG instructions are unscheduled. The type of MSG determines whether or not it requires a connection. If the MSG instruction requires a connection, it opens the needed connection when it is executed. You can configure the MSG instruction to keep the connection open (cache) or to close it after sending the message.

Sending MSGs to other controllers

Table 5.A Message to a Logix controller

If you want to:	For this item:	Type or select:
read (receive) data	Message Type	CIP Data Table Read
	Source Element	first element of the tag that contains data in the other controller
	Number of Elements	number of elements to transfer
	Destination Tag	first element of the tag (controller-scope) in this controller for the data
write (send) data	Message Type	CIP Data Table Write
	Source Tag	first element of the tag (controller-scope) in this controller that contains the data
	Number of Elements	number of elements to transfer
	Destination Element	first element of the tag for the data in the other controller

Table 5.B Message to a SLC 500 controller

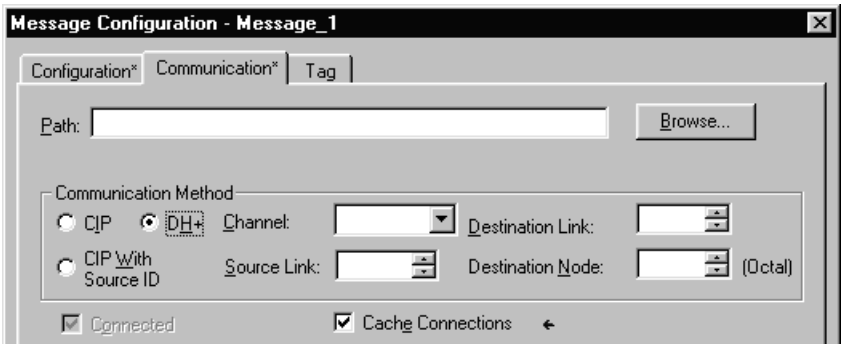
If the data is:	And you want to:	For this item:	Type or select:
integer (s)	read (receive) data	Message Type	*SLC Typed Read*
		Source Element	data table address in the SLC 500 controller (e.g., N7:10)
		Number Of Elements	number of integers to transfer
		Destination Tag	first element of `int_buffer`
	write (send) data	Message Type	*SLC Typed Write*
		Source Tag	first element of `int_buffer`
		Number Of Elements	number of integers to transfer
		Destination Element	data table address in the SLC 500 controller (e.g., N7:10)
floating-point (REAL)	read (receive) data	Message Type	*SLC Typed Read*
		Source Element	data table address in the SLC 500 controller (e.g., F8:0)
		Number Of Elements	number of values to transfer
		Destination Tag	first element of the tag (controller-scoped) in this controller for the data
	write (send) data	Message Type	*SLC Typed Write*
		Source Tag	first element of the tag (controller-scoped) in this controller that contains the data
		Number Of Elements	number of values to transfer
		Destination Element	data table address in the SLC 500 controller (e.g., F8:0)

Table 5.C Message to a PLC-5 controller

If the data is:	And you want to:	For this item:	Type or select:
integer (s)	read (receive) data	Message Type	*PLC5 Typed Read*
		Source Element	data table address in the PLC-5 controller (e.g., N7:10)
		Number Of Elements	number of integers to transfer
		Destination Tag	first element of `int_buffer`
	write (send) data	Message Type	*PLC5 Typed Write*
		Source Tag	first element of `int_buffer`
		Number Of Elements	number of integers to transfer
		Destination Element	data table address in the PLC-5 controller (e.g., N7:10)
floating-point (REAL)	read (receive) data	Message Type	*PLC5 Typed Read*
		Source Element	data table address in the PLC-5 controller (e.g., F8:0)
		Number Of Elements	number of values to transfer
		Destination Tag	first element of the tag (controller-scoped) in this controller for the data
	write (send) data	Message Type	*PLC5 Typed Write*
		Source Tag	first element of the tag (controller-scoped) in this controller that contains the data
		Number Of Elements	number of values to transfer
		Destination Element	data table address in the PLC-5 controller (e.g., F8:0)

Specifying the path to the target device

Once you configure the type of message instruction, you must specify the path of the target device.



In the path box, type the port from which the message exits the port and then type the address of the next device along the path to the destination:

Where:	For this:	Is:
port	Ethernet port from an Ethernet communication device	2
address	ControlLogix backplane	slot number
	DF1 network	station address (0-254)
	ControlNet network	node number (1-99 decimal)
	DH+ network	8# followed by the node number (1-77 octal) For example, to specify the octal node address of 37, type 8#37.
	EtherNet/IP network	You can specify a module on an EtherNet/IP network using any of these formats: IP address (e.g., 130.130.130.5) IP address:Port (e.g., 130.130.130.5:24) host name (e.g., tanks) host name:Port (e.g., tanks:24) If you use the host name, specify the host name of the computer where the controller resides.

If the message goes through multiple devices, follow the same pattern of specifying the port the message exits the communication port of the device and the address of the next device in the path.

For example, a valid path could be 1, 5, 2, 130.151.255.43, 1, 3 where:

This value:	Specifies:
1	the SoftLogix virtual chassis
5	the sending device (the Ethernet port in the controller's computer) is in slot 5 of the virtual chassis
2	sending the message out the Ethernet communication port
130.151.255.43	the IP address of the target device You can also identify the target device by host name. Enter the host name of the computer where the controller resides.
1	the backplane of the chassis that contains the target device
3	the target device is in slot 3 of the remote chassis

Producing and Consuming Data

The SoftLogix controller supports the ability to produce (multicast) and consume (receive) system-shared tags over an EtherNet/IP link. Produced and consumed data is accessible by multiple controllers over an Ethernet network. The controller sends or receives data at a predetermined RPI rate. This is the recommended method of communication between Logix controllers.

Produced and consumed tags must be controller-scoped tags of DINT or REAL data type, or in an array or structure.

Tag type:	Description:	Specify:
produced	These are tags that the controller produced for other controllers to consume.	• Enabled for producing • How many consumers allowed
consumed	These are tags whose values are produced by another controller.	• Controller name that owns the tag that the local controller wants to consume • Tag name or instance that the controller wants to consume • Data type of the tag to consume • Update interval of how often the local controller consumes the tag

The producer and consumer must be configured correctly for the specified data to be shared. A produced tag in the producer must be specified exactly the same as a consumed tag in the consumer.

If any produced/consumed tag between a producer and consumer is not specified correctly, none of the produced/consumed tags for that producer and consumer will be transferred. For example, if a SoftLogix controller is consuming three tags that another controller consumes but the first tag is specified incorrectly, none of the tags are transferred to the consuming SoftLogix controller.

However, one consumer failing to access shared data does not affect other consumers accessing the same data. For example, if the producing SoftLogix controller from the previous example also produced tags for other consuming controllers but did so correctly, those tags are still transferred to the additional consuming controllers.

Maximum number of produced and consumed tags

The maximum number of produced/consumed tags that you can configure depends on the connection limits of the Ethernet port on the controller. You can have a maximum of 32 connections through the Ethernet port.

Each produced tag uses one connection for the tag and the first configured consumer of the tag. Each consumer thereafter uses an additional connection.

If you have a lot of data to produce or consume, organize that data into an array. An array is treated as one tag, so it uses only one connection. An array can be as large as 488 bytes, which is also the limit of a produced or consumed tag.

Size limit of a produced or consumed tag

A produced or consumed tag can be as large as 488 bytes, but it must also fit within the bandwidth of the EtherNet/IP network.

Producing a tag

Produced data must be of DINT or REAL data type or a structure. You can use a user-defined structure to group BOOL, SINT, and INT data to be produced. To create a produced tag:

1. You must be programming offline.
2. In the controller organizer, double-click the Controller Tags folder and then click the Edit Tags tab.
3. Select the tag that you want to produce, or enter a new tag, and display the Tag Properties dialog box.
4. Make sure the tag is controller scope.
5. Select the “Produce this tag” check box. Specify how many controllers can consume the tag.

You can produce a base, alias, or consumed tag.

The consumed tag in a receiving controller must have the same data type as the produced tag in the originating controller. The controller performs type checking to ensure proper data is being received.

Produced tags require connections. The number of connections depends on how many controllers are consuming the tags. The controller requires one connection for the produced tag and the first consumer. Then, the controller requires an additional connection for each subsequent consumer.

Consuming a tag

A consumed tag represents data that is produced by one controller and received and stored by the consuming controller. To create a consumed tag:

1. You must be programming offline.
2. In the controller organizer, double-click the Controller Tags folder and then click the Edit Tags tab.
3. Select the tag that you want to consume, or enter a new tag, and display the Tag Properties dialog box.
4. Specify:

In this field:	Type or select:
Tag Type	Select Consumed.
Controller	Select the name of the other controller. You must have already created the controller in the controller organizer for the controller name to be available.
Remote Tag Name Remote Instance	Type a name for the tag in the other controller you want to consume. **Important:** The name must match the name in the remote controller exactly, or the connection faults.
RPI (requested packet interval)	Type the amount of time in msec between updates of the data from the remote controller. The local controller will receive data at least this fast. Virtual-backplane controllers, such as CompactLogix and FlexLogix controllers, only produce data at RPIs in powers of two milliseconds (such as 2, 4, 8, 16, 32, 64, etc.), or when triggered by an IOT instruction.
Display Style	If you are creating a consumed tag that refers to a tag whose data type is BOOL, SINT, INT, DINT, or REAL, you can select a display style. This display style defines how the tag value will be displayed in the data monitor and ladder editor. The display style does not have to match the display style of the tag in the remote controller.

All consumed tags are automatically controller-scope.

The produced tag in the originating SoftLogix controller must have the same data type as the consumed tag in the consuming controller. The SoftLogix controller performs type checking to make sure proper data is being received.

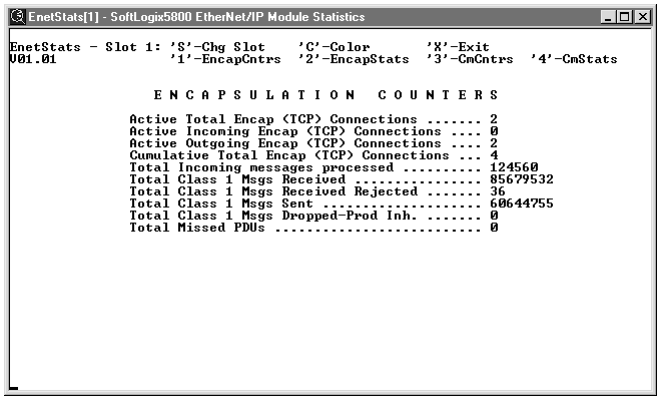
IMPORTANT If a consumed-tag connection fails, none of the tags are transferred from the producing controller to the consuming controller.

Checking EtherNet/IP Statistics

The SoftLogix controller installs with an EtherNet/IP statistics utility that displays different counters for the EtherNet/IP module.

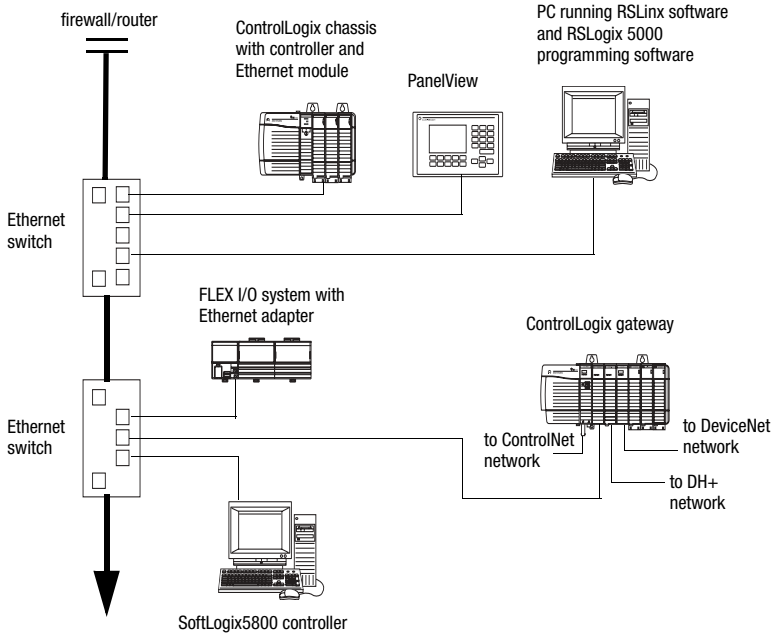
1. Select the EtherNet/IP Statistics Utility from the folder where you installed the SoftLogix controller.

2 Use the character key at the top of the utility screen to display information and change screen characteristics.

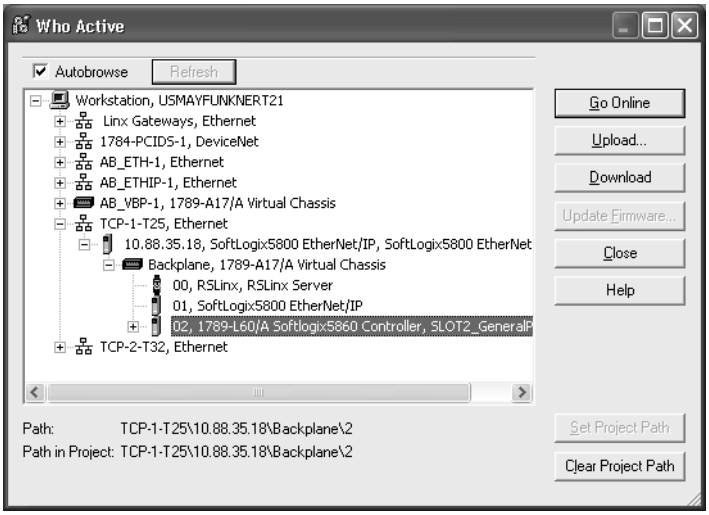


Example 1: Workstation Remotely Connected to a SoftLogix Controller

In the following example, a workstation remotely connects to a SoftLogix controller over an Ethernet link in order to remotely program the SoftLogix controller.

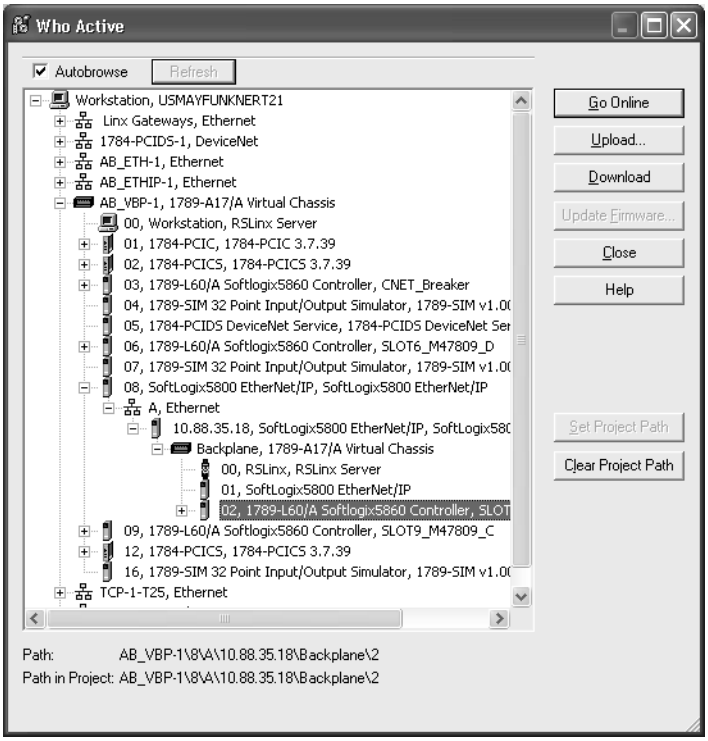


Browsing from a computer to a remote SoftLogix controller looks like:



In this example, a remote computer is browsing to a SoftLogix controller. The remote computer can be RSLinx gateway or not and the computer can have UDP enabled or disabled. Either way, the remote computer can browse the controller.

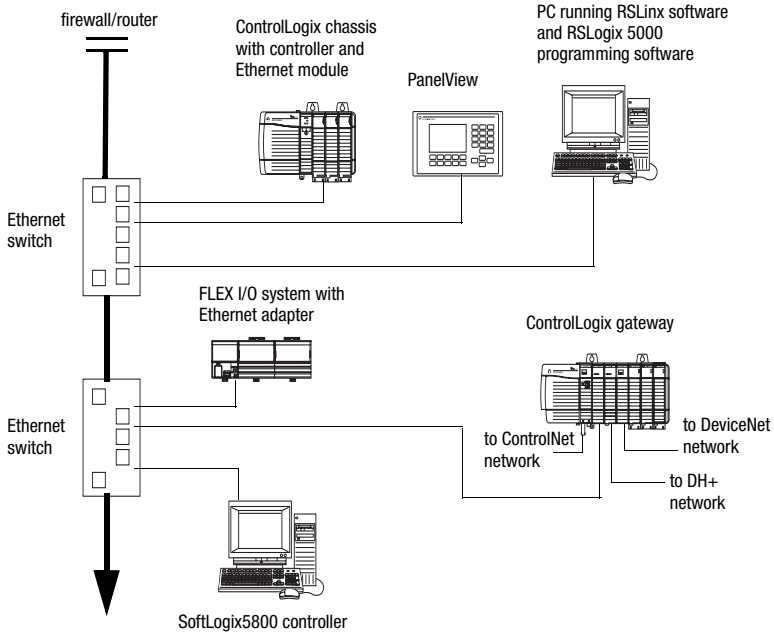
Browsing from the EtherNet/IP module in a SoftLogix controller to remote devices looks like:



In this example, the computer cannot be an RSLinx gateway. The UDP option in RSLinx must be disabled.

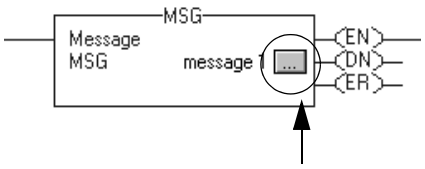
Example 2: Sending Messages Over Ethernet

In the following example, the SoftLogix controller can send messages to the other devices on the Ethernet network.

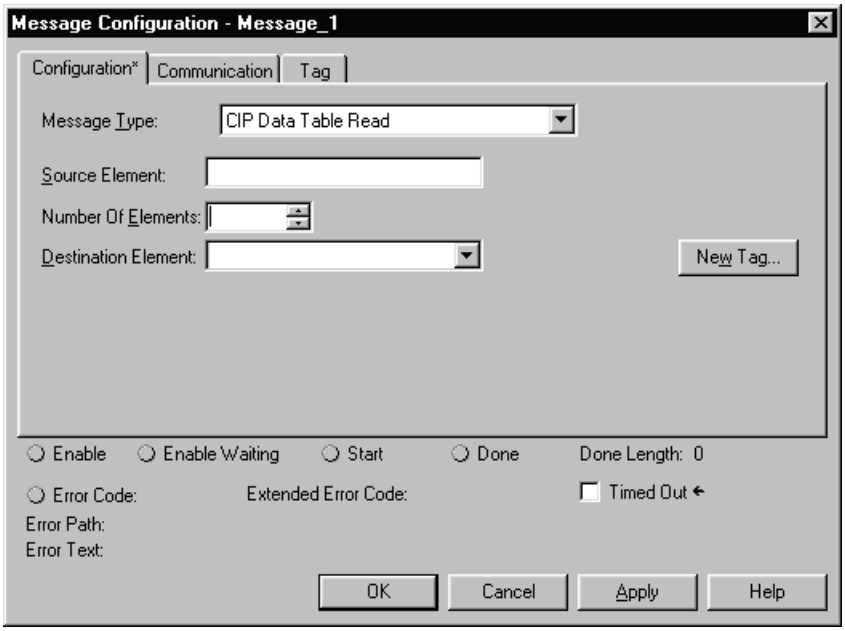


Configuring a MSG instruction

How you configure the MSG instruction depends on the target device.



Click here to configure the MSG instruction



On the Configuration tab, configure:

For this item:	Specify:
Message Type	CIP Data Table Read or CIP Data Table Write
Source Tag	Tag containing the data to be transferred
Number of Elements	Number of array elements to transfer
Destination Tag	Tag to which the data will be transferred

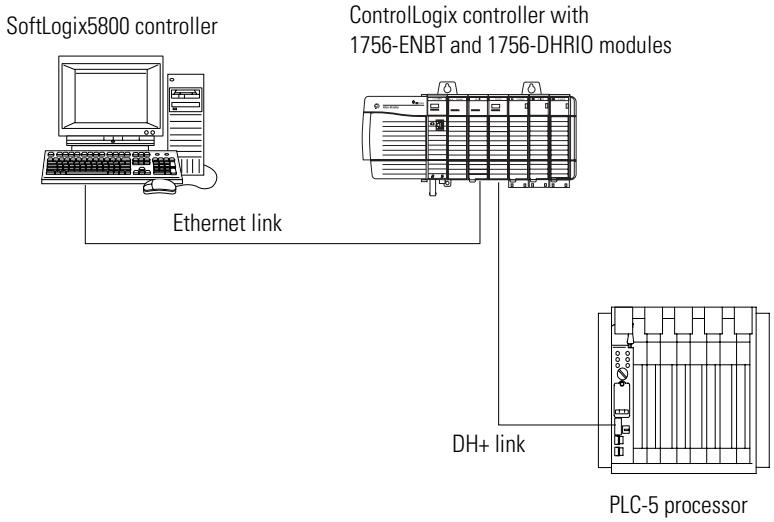
On the Communication tab, specify the communication path.

A communication path requires pairs of numbers. The first number in the pair identifies the port from which the message exits. The second number in the pair designates the node address of the next device.

For this item:	Specify:
Communication Path (each SoftLogix controller resides in its own computer)	1,2,130.151.255.43,1,5 where: 1 is the SoftLogix backplane of Soft1 2 is Ethernet port in slot 5 130.151.255.43 is IP address of the target 1 is the SoftLogix backplane of Soft2 5 is the controller slot of Soft2
Communication Path (each SoftLogix controller resides in the same computer)	1,5 where: 1 is the SoftLogix backplane of Soft1 5 is the controller slot of Soft2

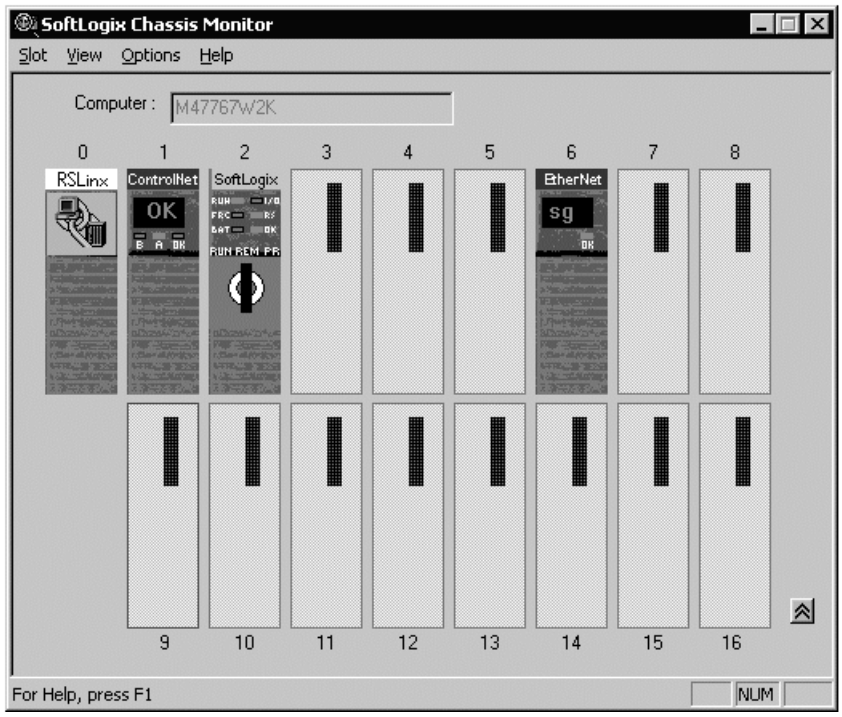
Example 3: Sending Messages Over Ethernet to a PLC-5 Processor

In the following example, the SoftLogix controller sends a message through the 1756-ENBT, out the 1756-DHRIO, and to a PLC-5 processor at DH+ node 2.



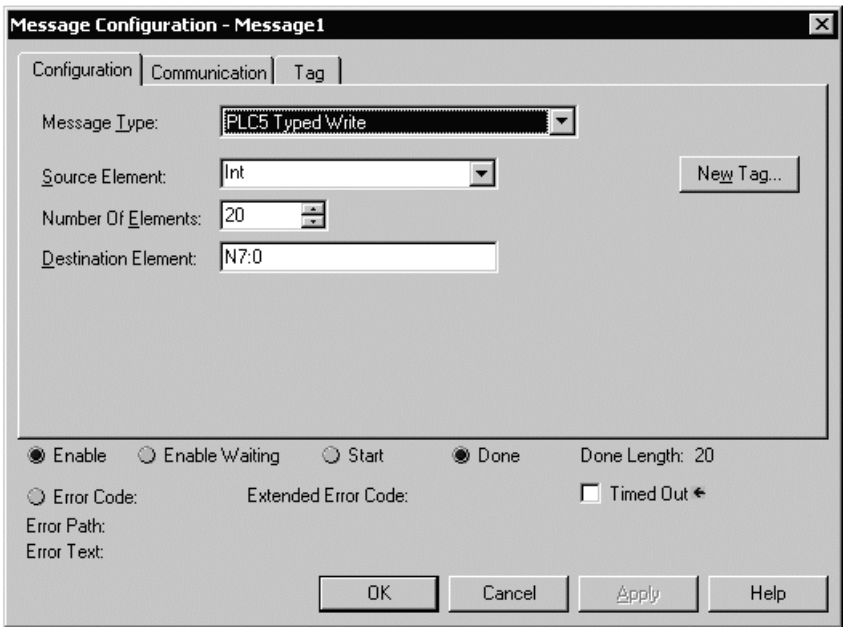
Configuring the SoftLogix controller

This example uses a SoftLogix controller configured with this virtual chassis:

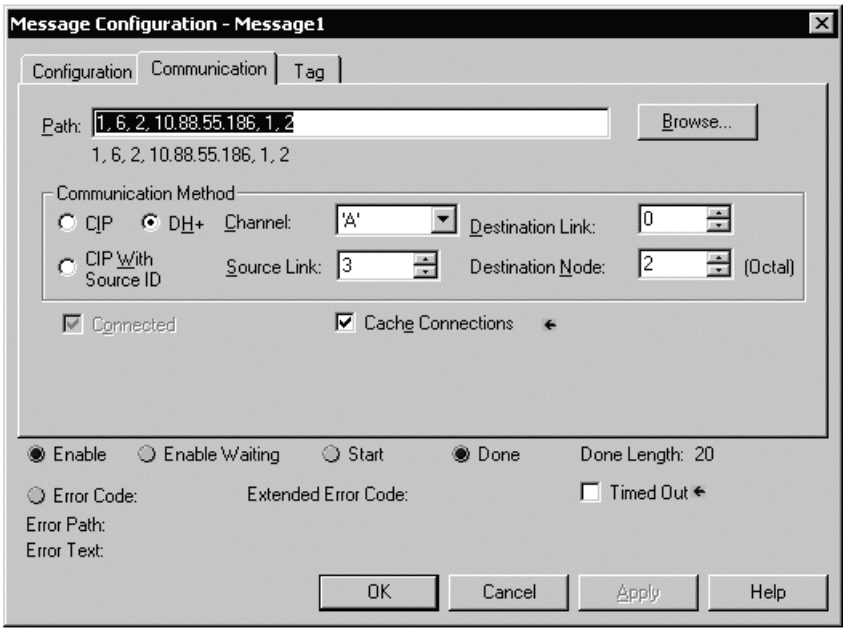


Configuring a MSG instruction

Use a PLC-5 Typed Write MSG instruction.



And specify the path and communication method as:



The example path is 1, 6, 2, 10.88.55.186, 1, 2 where:

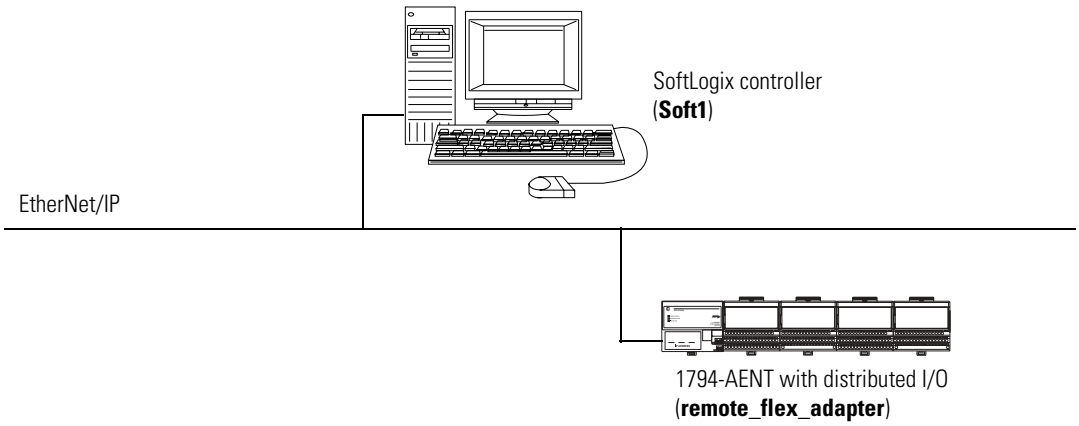
This value:	Specifies:
1	the SoftLogix virtual chassis
6	the sending device (the Ethernet port in the controller's computer) is in slot 6 of the virtual chassis
2	sending the message out the Ethernet communication port
10.88.55.186	the IP address of the 1756-ENBT module
1	the 1756 backplane
2	the slot where the 1756-DHRIO module is in the 1756 chassis

The example communication method is:

This field:	Specifies:
Channel = A	channel A on the 1756-DHRIO module
Source Link = 3	the DH+ address of the 1756-DHRIO module is node 3
Destination Node = 2	the DH+ address of the PLC-5 processor is node 2

Example 4: Controlling Distributed I/O

In the following example, one SoftLogix controller controls distributed I/O through a 1794-AENT module.

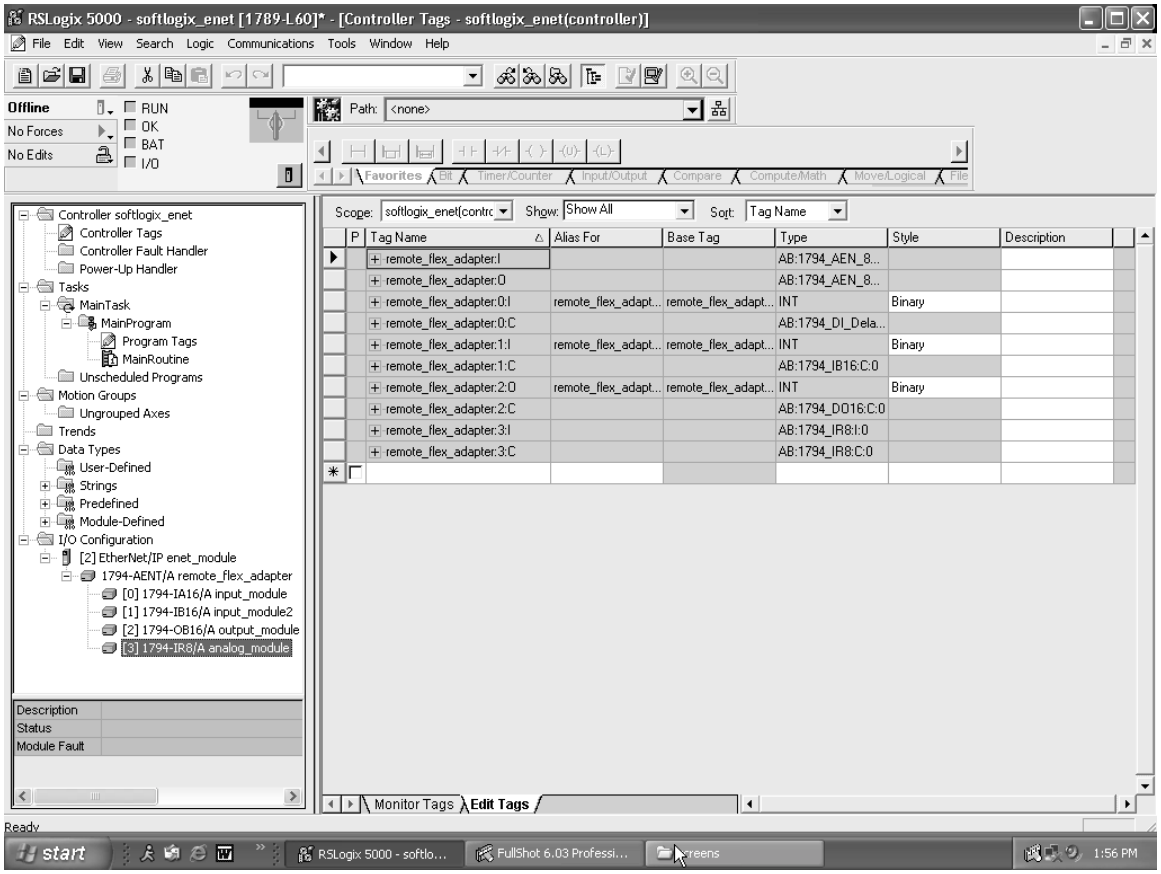


Controlling distributed I/O

This example has Soft1 controlling the I/O connected to the remote 1794-AENT module. The data the SoftLogix controller receives from the distributed I/O modules depends on how you configure the I/O modules. You can configure each module as a direct connection or as rack optimized. One chassis can have a combination of some modules configured as a direct connection and others as rack optimized.

All analog modules require direct connections. Diagnostic modules support rack-optimized connections, but require direct connections to take full advantage of their diagnostic features.

In RSLogix 5000 software, the controller project and associated tags looks like:



Throughput is based on the performance of the PC running the SoftLogix controller.

Notes:

Communicating with Devices on a Serial Link

Using This Chapter

IMPORTANT Limit the length of serial (RS-232) cables to 15.2m (50 ft.).

Configuring Your System for a Serial Link

For the SoftLogix controller to operate on a serial network, you need:

- the computer where the SoftLogix controller resides must have a serial port
- RSLinx software to configure the serial communication driver

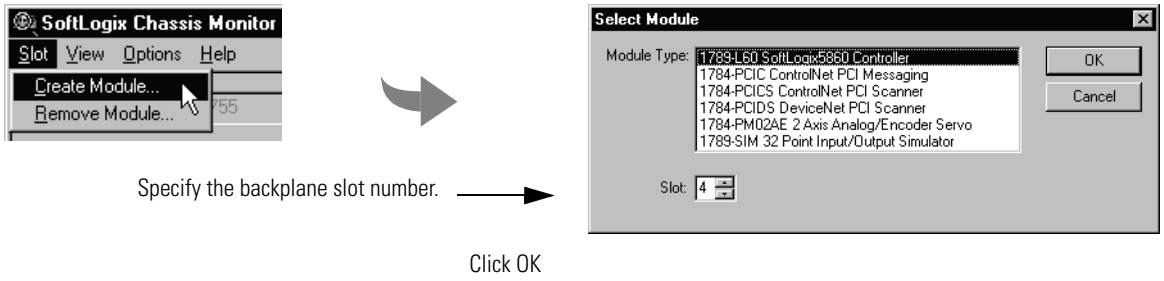
  If a remote computer communicates with the SoftLogix controller via a serial connection, the remote computer must have the serial driver installed. The computer with the SoftLogix controller does not need the serial driver to connect to other devices over a serial link.

- RSLogix 5000 programming software to configure the serial port of the controller

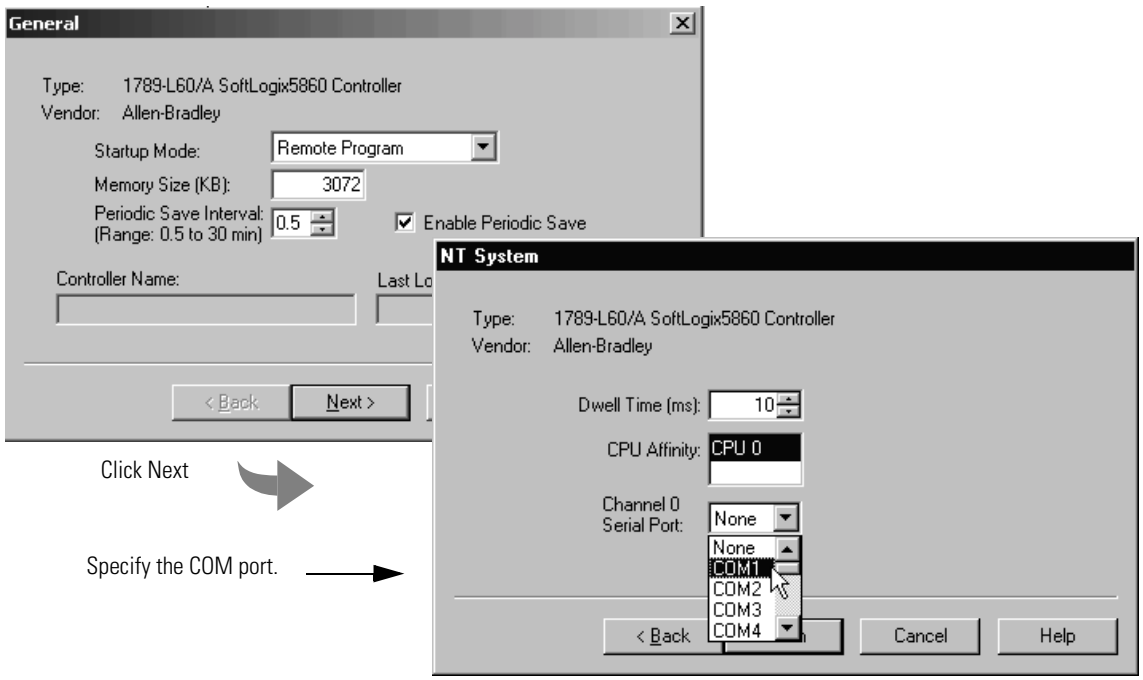
Step 1: Configure the serial port

Use the SoftLogix chassis monitor to select which COM port to use for serial communications. The controller supports only one COM port for DF1 communications.

1. From the SoftLogix chassis monitor, select Slot → Create Module or right click the appropriate slot and select Create. Select the controller.



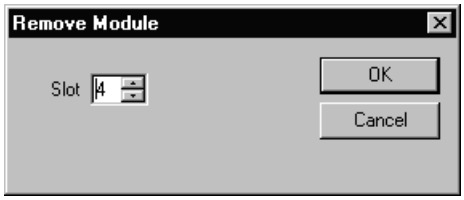
2. Specify configuration settings for the controller. On the second window, select the COM port for serial communications.



Changing the COM port setting

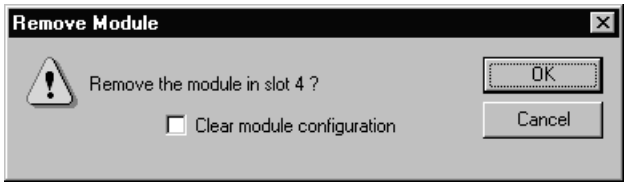
Once you select a COM port for the controller, you can only change the setting by removing the controller from the chassis and reinstalling the controller.

1. From the SoftLogix chassis monitor, select Slot → Remove Module. Accept the chassis monitor prompts to remove the controller

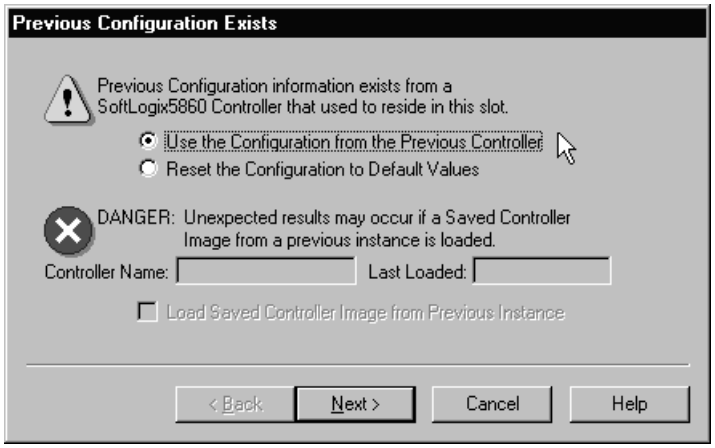


Click OK

Do not clear the module configuration, unless you want to enter all new information.



2. In the same slot, re-add the controller. The chassis monitor prompts whether to use the same configuration that was previously installed.



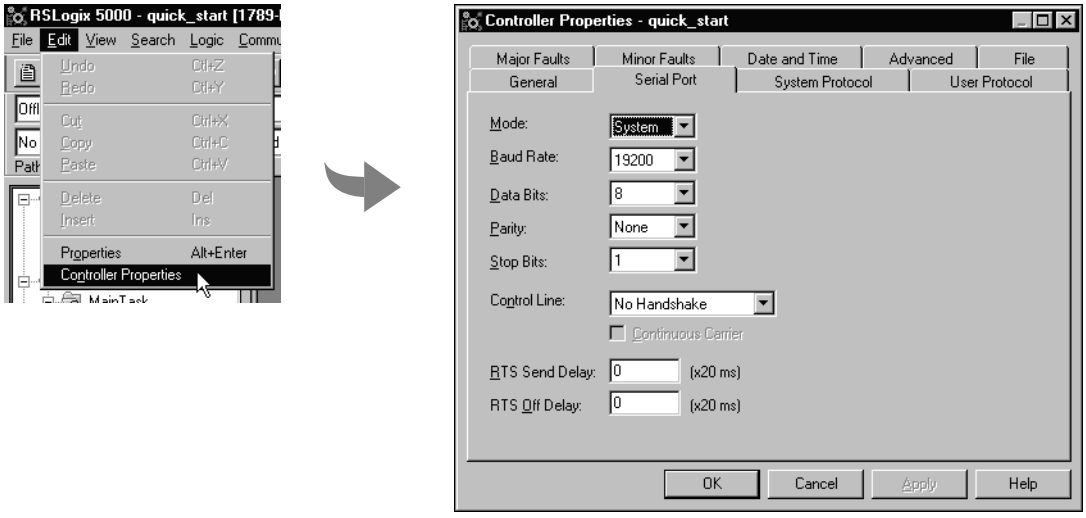
Click Next

3. Follow the steps on the previous page to select a COM port for serial communications.

Step 2: Configure the serial port of the controller

1. In RSLogix 5000 software, select Edit → Controller Properties.

2. On the Serial Port tab, specify the appropriate serial communication settings.



3. On the System Protocol tab, select the appropriate DF1 communication mode for point-to-point or master/slave communications. Or on the User Protocol tab, select ASCII to communicate with an ASCII device.

Specifying serial port characteristics

Specify these characteristics on the Serial Port tab (default values are shown in bold):

Characteristic:	Description (default is shown in bold):
Mode	Select **System** (for DF1 communication) or User mode (for ASCII communication).
Baud rate	Specifies the communication rate for the serial port. Select a baud rate that all devices in your system support. Select 110, 300 600, 1200, 2400, 4800, 9600, or **19200** Kbps.
Data bits	Specifies the number of bits per message packet. Select **8**.
Parity	Specifies the parity setting for the serial port. Parity provides additional message-packet error detection. Select **None** or Even.
Stop bits	Specifies the number of stop bits to the device with which the controller is communicating. Select **1** or 2.

Characteristic:	Description (default is shown in bold):
Control line	Specifies the mode in which the serial driver operates. Select **No Handshake**, Full-Duplex, Half-Duplex with Continuous Carrier, or Half-Duplex without Continuous Carrier. If you are not using a modem, select No Handshake If both modems in a point-to-point link are full-duplex, select Full-Duplex for both controllers. If the master modem is full duplex and the slave modem is half-duplex, select Full-Duplex for the master controller and select Half-Duplex with Continuous Carrier for the slave controller. If all the modems in the system are half-duplex, select Half-Duplex without Continuous Carrier for the controller.
RTS send delay	Enter a count that represents the number of 20msec periods of time that elapse between the assertion of the RTS signal and the beginning of a message transmission. This time delay lets the modem prepare to transmit a message. The CTS signal must be high for the transmission to occur. The range is **0**-32767 periods.
RTS off delay	Enter a count that represents the number of 20msec periods of time that elapse between the end of a message transmission and the de-assertion of the RTS signal. This time delay is a buffer to make sure the modem successfully transmits the entire message. The range is **0**-32767 periods. Normally leave at zero.

Specifying system protocol characteristics

The available system modes are:

Use this mode:	For:	See page:
DF1 point-to-point	communication between the controller and one other DF1-protocol-compatible device. This is the default system mode. This mode is typically used to program the controller through its serial port.	6-7
DF1 master mode	control of polling and message transmission between the master and slave nodes. The master/slave network includes one controller configured as the master node and as many as 254 slave nodes. Link slave nodes using modems or line drivers. A master/slave network can have node numbers from 0-254. Each node must have a unique node address. Also, at least 2 nodes must exist to define your link as a network (1 master and 1 slave station are the two nodes).	6-9
DF1 slave mode	using a controller as a slave station in a master/slave serial communication network. When there are multiple slave stations on the network, link slave stations using modems or line drivers. When you have a single slave station on the network, you do not need a modem to connect the slave station to the master; you can configure the control parameters for no handshaking. You can connect 2-255 nodes to a single link. In DF1 slave mode, a controller uses DF1 half-duplex protocol. One node is designated as the master and it controls who has access to the link. All the other nodes are slave stations and must wait for permission from the master before transmitting.	6-9
User mode	communicating with ASCII devices This requires your program logic to use the ASCII instructions to read and write data from and to an ASCII device.	6-12

Monitoring the Controller LEDs

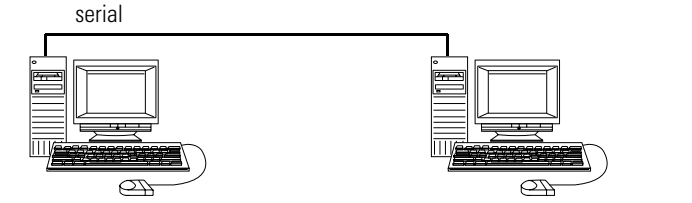
The SoftLogix controller has an RS-232 LED that follows this behavior:

This state:	Means:
off	You selected "None" for the COM port selection of the controller.
green	The COM port you selected was successfully assigned to channel 0 of the controller.
red	There is conflict with COM port or the COM port number you selected is invalid.

Please note that these LED states are different than for the ControlLogix controller.

Example 1: Workstation Directly Connected to a SoftLogix Controller

In the following example, a workstation directly connects to a SoftLogix controller over a serial link.



Use RSLogix 5000 programming software to configure the controller's serial port for the DF1 point-to-point (full-duplex) protocol. This type of protocol supports simultaneous transmission between two devices in both directions. The DF1 point-to-point protocol controls message flow, detects and signals errors, and retries if errors are detected.

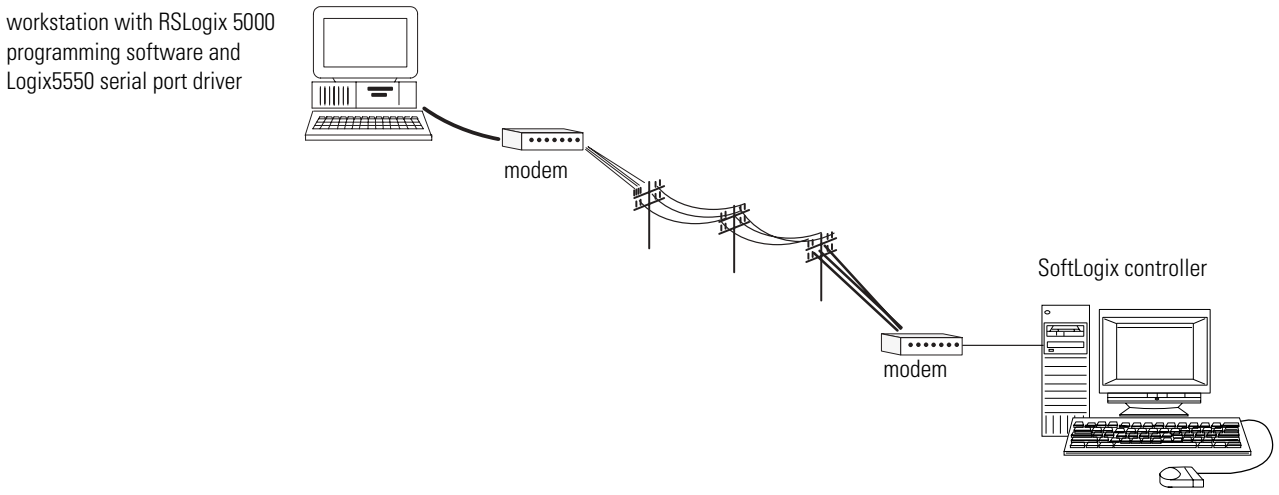
IMPORTANT The workstation with RSLogix 5000 programming software must also have the Logix5550 serial port driver installed through RSLinx software.

Configuring a DF1 point-to-point station

This field:	Description:
Station address	The station address for the serial port on the DF1 point-to-point network. Enter a valid DF1 address (0-254). Address 255 is reserved for broadcast messages. The default is 0.
NAK receive limit	Specifies the number of NAKs the controller can receive in response to a message transmission. Enter a value 0-127. The default is 3.
ENQ transmit limit	Specifies the number of inquiries (ENQs) you want the controller to send after an ACK timeout. Enter a value 0-127. The default is 3.
ACK timeout	Specifies the amount of time you want the controller to wait for an acknowledgment to its message transmission. Enter a value 0-32767. Limits are defined in 20ms intervals. The default is 50 (1000ms).
Embedded response	Specifies how to enable embedded responses. Select Autodetect (enabled only after receiving one embedded response) or Enabled. The default is Autodetect.
Error detection	Select BCC or CRC error detection. Configure both stations to use the same type of error checking. **BCC:** the controller sends and accepts messages that end with a BCC byte for error checking. BCC is quicker and easier to implement in a computer driver. This is the default. **CRC:** the controller sends and accepts messages with a 2-byte CRC for error checking. CRC is a more complete method.
Enable duplicate detection	Select whether or not the controller should detect duplicate messages. The default is duplicate detection enabled.

Example 2: Workstation Remotely Connected to a SoftLogix Controller

In the following example, a workstation remotely connects to a SoftLogix controller over a serial link. A modem is connected to the controller to provide remote access.



If you use a modem to remotely connect the controller to one workstation, use RSLogix 5000 programming software to configure the serial port of the controller for the DF1 point-to-point (full-duplex) protocol, as in the previous example. If the controller is part of a master/slave serial network, configure the serial port of the controller for either the DF1 master or DF1 slave protocol (both half-duplex).

Master/slave communication methods

A master station can communicate with a slave station in two ways:

Name:	This method:	Benefits:
standard communication mode	Initiates polling packets to slave stations according to their position in the polling array(s). Polling packets are formed based on the contents of the normal poll array and the priority poll array.	This communication method is most often used for point-to-multipoint configurations. This method provides these capabilities: • slave stations can send messages to the master station (polled report-by-exception) • slave stations can send messages to each other via the master • master maintains an active station array The poll array resides in a user-designated data file. You can configure the master: • to send messages during its turn in the poll array *or* • for between-station polls (master transmits any message that it needs to send before polling the next slave station) In either case, configure the master to receive multiple messages or a single message per scan from each slave station.
message-based communication mode	initiates communication to slave stations using only user-programmed message (MSG) instructions. Each request for data from a slave station must be programmed via a MSG instruction. The master polls the slave station for a reply to the message after waiting a user-configured period of time. The waiting period gives the slave station time to formulate a reply and prepare the reply for transmission. After all of the messages in the master's message-out queue are transmitted, the slave-to-slave queue is checked for messages to send.	If your application uses satellite transmission or public switched-telephone-network transmission, consider choosing message-based communication. Communication to a slave station can be initiated on an as-needed basis. Also choose this method if you need to communicate with non-intelligent remote terminal units (RTUs).

Configuring a DF1 slave station

This field:	Description:
Station address	The station address for the serial port on the DF1 slave. Enter a valid DF1 address (0-254). Address 255 is reserved for broadcast messages. The default is 0.
Transmit retries	The number of times the remote station retries a message after the first attempt before the station declares the message undeliverable. Enter a value 0-127. The default is 3.
Slave poll timeout	Specifies the amount of time the slave station waits to be polled by a master before indicating a fault. Enter a value 0-32767. Limits are defined in 20ms intervals. The default is 3000 (60,000ms).
EOT suppression	Select whether or not to suppress sending EOT packets in response to a poll. The default is not to suppress sending EOT packets.
Error detection	Select BCC or CRC error detection. Configure both stations to use the same type of error checking. **BCC:** the controller sends and accepts messages that end with a BCC byte for error checking. BCC is quicker and easier to implement in a computer driver. This is the default. **CRC:** the controller sends and accepts messages with a 2-byte CRC for error checking. CRC is a more complete method.
Enable duplicate detection	Select whether or not the controller should detect duplicate messages. The default is duplicate detection enabled.

Configuring a DF1 master station

This field:	Description:
Station address	The station address for the serial port on the DF1 master. Enter a valid DF1 address (0-254). Address 255 is reserved for broadcast messages. The default is 0.
Transmit retries	Specifies the number of times a message is retried after the first attempt before being declared undeliverable. Enter a value 0-127. The default is 3.
ACK timeout	Specifies the amount of time you want the controller to wait for an acknowledgment to its message transmission. Enter a value 0-32767. Limits are defined in 20ms intervals. The default is 50 (1000ms).
Reply message wait	**Message-based polling mode only** Specifies the amount of time the master station waits after receiving an ACK to a master-initiated message before polling the slave station for a reply. Enter a value 0-65535. Limits are defined in 20ms intervals. The default is 5 (100ms).
Polling mode	Select one of these: • Message Based (slave cannot initiate messages) • Message Based (slave can initiate messages) - default • Standard (multiple message transfer per node scan) • Standard (single message transfer per node scan)
Master transmit	**Standard polling modes only** Select when the master station sends messages: • between station polls (default) • in polling sequence

This field:	Description:
Normal poll node tag	**Standard polling modes only** An integer tag array that contains the station addresses of the slave stations. Create a single-dimension array of data type INT that is large enough to hold all the normal station addresses. The minimum size is three elements. This tag must be controller-scoped. The format is: *list[0]* contains total number of stations to poll *list[1]* contains address of station currently being polled *list[2]* contains address of first slave station to poll *list[3]* contains address of second slave station to poll *list[n]* contains address of last slave station to poll
Normal poll group size	**Standard polling modes only** The number of stations the master station polls after polling all the stations in the priority poll array. Enter 0 (default) to poll the entire array.
Priority poll node tag	**Standard polling modes only** An integer tag array that contains the station addresses of the slave stations you need to poll more frequently. Create a single-dimension array of data type INT that is large enough to hold all the priority station addresses. The minimum size is three elements. This tag must be controller-scoped. The format is: *list[0]* contains total number of stations to be polled *list[1]* contains address of station currently being polled *list[2]* contains address of first slave station to poll *list[3]* contains address of second slave station to poll *list[n]* contains address of last slave station to poll
Active station tag	**Standard polling modes only** An array that stores a flag for each of the active stations on the DF1 link. Both the normal poll array and the priority poll array can have active and inactive stations. A station becomes inactive when it does not respond to the master's poll. Create a single-dimension array of data type SINT that has 32 elements (256 bits). This tag must be controller-scoped.
Error detection	Select BCC or CRC error detection. Configure both stations to use the same type of error checking. **BCC:** the controller sends and accepts messages that end with a BCC byte for error checking. BCC is quicker and easier to implement in a computer driver. This is the default. **CRC:** the controller sends and accepts messages with a 2-byte CRC for error checking. CRC is a more complete method.
Enable duplicate detection	Select whether or not the controller should detect duplicate messages. The default is duplicate detection enabled.

If you choose one of the standard polling modes

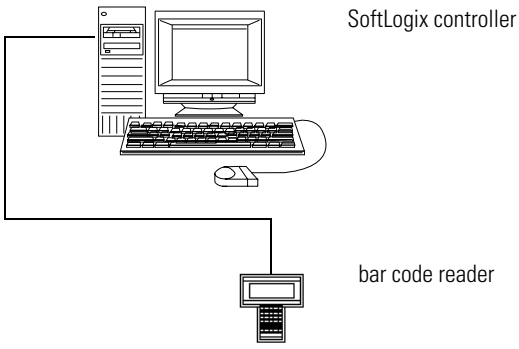
The master station polls the slave stations in this order:

1. all stations that are active in the priority poll array
2. one station that is inactive in the priority poll array
3. the specified number (normal poll group size) of active stations in the normal poll array
4. one inactive station, after all the active stations in the normal poll array have been polled

Use the programming software to change the display style of the active station array to binary so you can view which stations are active.

Example 3: SoftLogix Controller to a Bar Code Reader

In the following example, the SoftLogix controller connects to a bar code reader. A bar code reader is an ASCII device, so you configure the serial port differently than in the previous examples. Configure the serial port for user mode, rather than a DF1 mode.

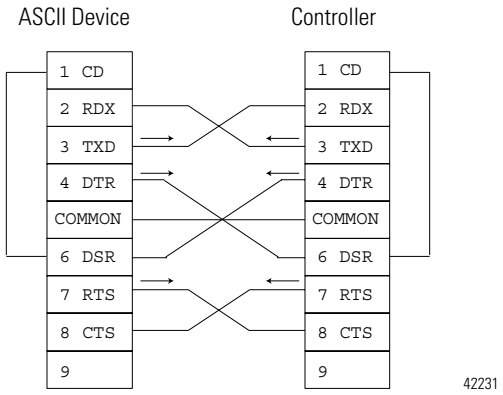
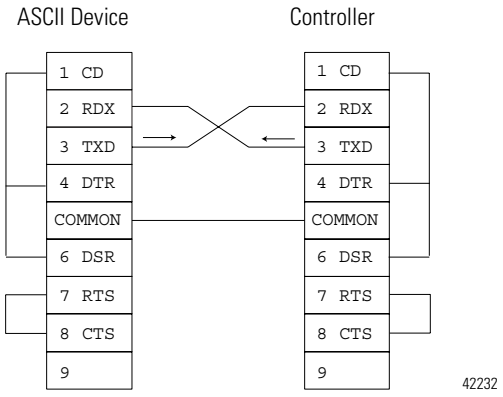


Connect the ASCII device to the controller

To connect the ASCII device to the serial port of the controller:

1. For the serial port of the ASCII device, determine which pins send signals and which pins receive signals.

2. Connect the sending pins to the corresponding receiving pins and attach jumpers:

If the communications:	Then wire the connectors as follows:
handshake	
do not handshake	

3. Attach the cable shield to both connectors and tie the cable to both connectors.

4. Connect the cable to the controller and the ASCII device.

The following table lists the default serial port configuration settings for the ASCII protocol. You specify these settings on the User Protocol tab under Controller Properties.

Configuring user mode

This field:	Description:
Buffer size	Specify the maximum size (in bytes) of the data array you plan to send and receive. The default is 82 bytes.
Termination characters	Specify the characters you will use to designate the end of a line. The default characters are '$r' and '$FF'.
Append characters	Specify the characters you will append to the end of a line. The default characters are '$r' and '$l'.
XON/XOFF	Select whether or not to regulate the flow of incoming data. The default is disabled.
Echo mode	Select whether or not to echo data back to the device from which it was sent. The default is disabled.
Delete mode	Select Ignore, CTR, or Printer for the delete mode. The default is Ignore.

Programming ASCII instructions

The controller supports ASCII instructions to communicate with ASCII devices. Your RSLogix 5000 programming software CDROM includes programming examples using ASCII instructions.

For information about using these examples, see the *Logix5000 Controllers Common Procedures Programming Manua*l, publication 1756-PM001.

Notes:

Configuring and Using Simulated I/O

Using This Chapter

The 1789-SIM module is a software-only module that comes with the SoftLogix controller, absolutely no hardware required. You can put as many SIM modules as you have available slots according to your activation level.

The 1789-SIM module lets you change inputs and monitor outputs of your application by toggling input bits and monitoring output bits on the 1789-SIM module. Use this module to test logic without having physical I/O attached to the system.

Configuring Your System for a 1789-SIM Module

For the SoftLogix controller to simulate local I/O, you need:

- a 1789-SIM module (comes with the SoftLogix5800 controller)

  You are limited by the activation level of your SoftLogix controller as to how many modules you can install.

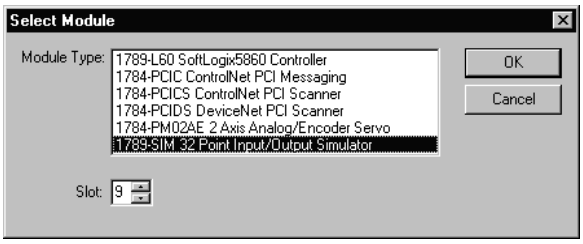
- RSLogix 5000 programming software to configure the 1789-SIM module

Even though the 1789-SIM module is a software-based module, each module you create uses communication resources. If you are controlling actual I/O and simulating I/O, the 1789-SIM module(s) in your application use communication resources that could impact control performance. If this occurs, increase the RPI of your 1789-SIM module(s). This maintains control performance because the greater RPI of the 1789-SIM module(s) lessens the load on the SoftLogix system.

Step 1: Create the 1789-SIM module in the chassis

Before you can operate the module, you must create the 1789-SIM module as part of the SoftLogix chassis. You can install as many 1789-SIM modules as allowed by your activation level of the controller.

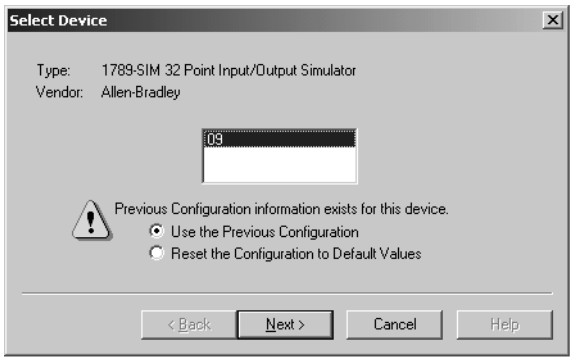
1. From the SoftLogix chassis monitor, select Slot → Create Module or right click the appropriate slot and select Create. Select the 1789-SIM module.



Specify the backplane slot number.

Click OK

2. Verify the slot number for the 1789-SIM module.

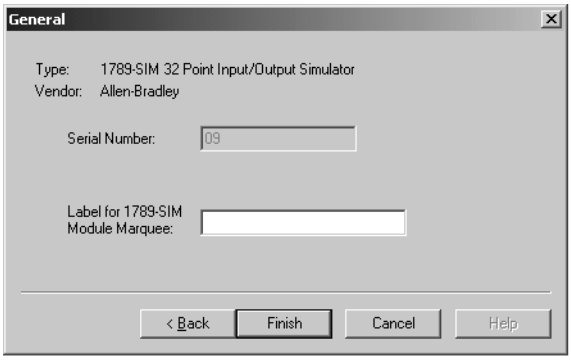


Verify the slot number.

If you previously had a 1789-SIM module configured in this slot, the chassis monitor remembers the configuration of that previous module.

Click Next

3. Specify a label for the 1789-SIM module.

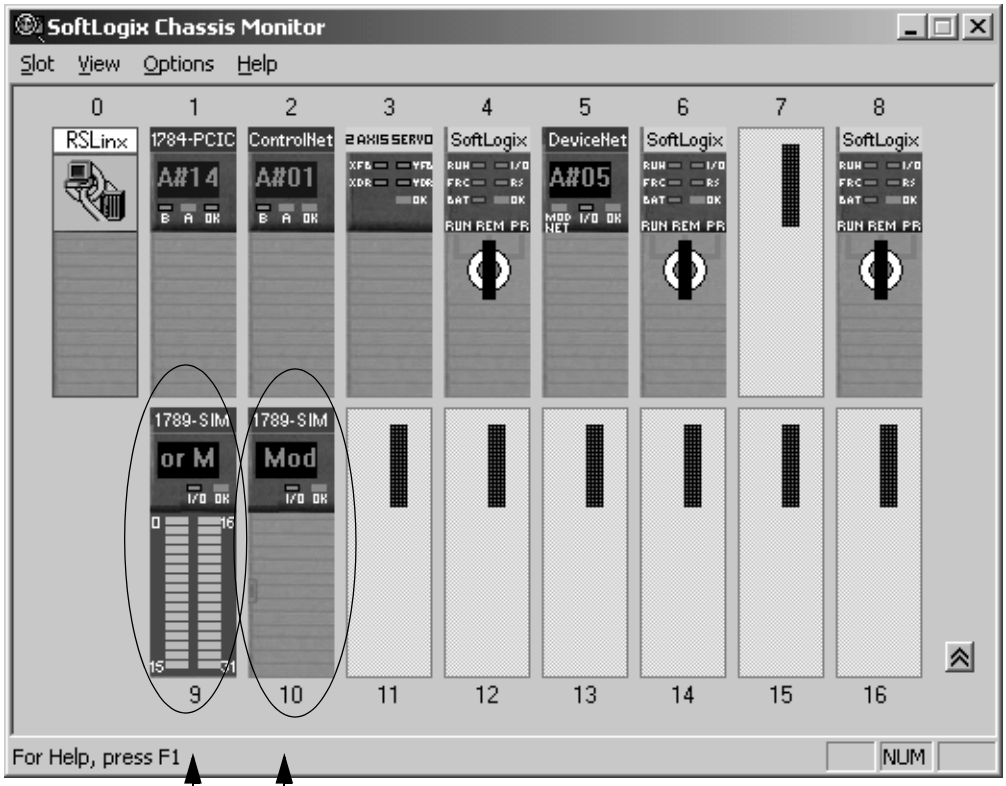


Enter the label. The text you enter here scrolls across the front of the module in the chassis monitor. If you do not enter a label, the default label is "Simulator Module."

Click Finish

You can specify any slot number greater than 0 for the 1789-SIM module. RSLinx software resides in slot 0.

The chassis monitor shows the 1789-SIM module as a virtual module in the SoftLogix chassis. Note that the door of the 1789-SIM module opens to display the output bits. Left-click to open or close the module door.



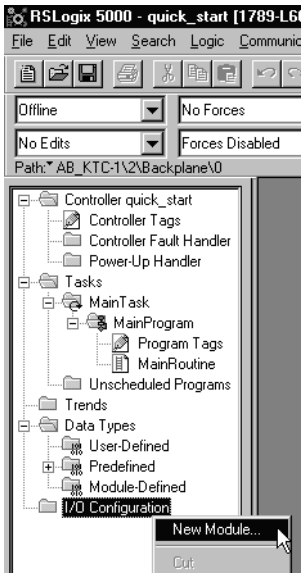
The module door is open. The module door is closed.

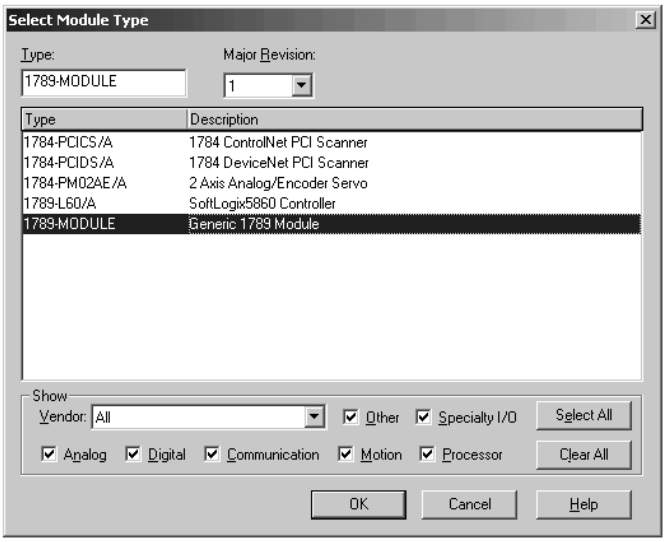
Step 2: Configure the 1789-SIM module as part of the project

Use RSLogix 5000 programming software to map the 1789-SIM module as part of the SoftLogix project.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.

2 Right-click to select New Module and add a generic 1789 module.



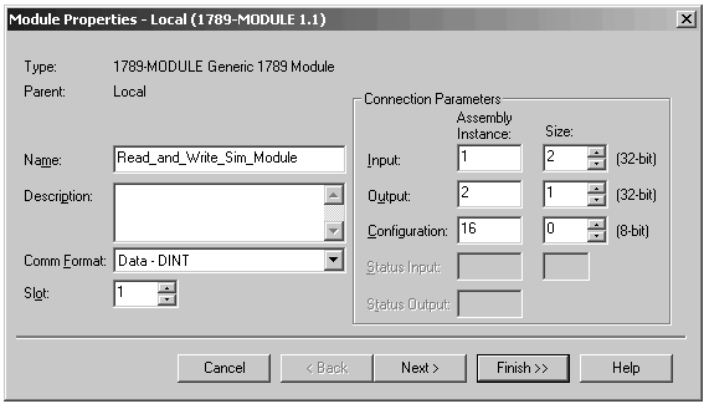


Click OK

3. Specify the connection parameters.

For this connection:	Set the connection parameters to:	
read and write This connection lets the controller read inputs and write outputs	Input Assembly Instance	1
	Input Size	2
	Output Assembly Instance	2
	Output Size	1
	Configuration Assembly Instance	16
	Configuration Size	0
listen only This connection lets the controller read inputs, but not write outputs	Input Assembly Instance	1
	Input Size	2
	Output Assembly Instance[1]	3
	Output Size	1
	Configuration Assembly Instance	16
	Configuration Size	0

(1) This is the only parameter setting that is different from the read and write connection.

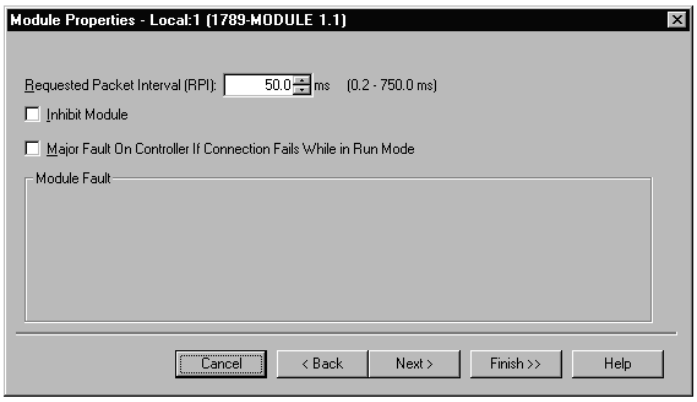


Click OK

continued

4. Specify the requested packet interval (RPI).

You must enter at least 50.0ms for the RPI. The connection will fail if the RPI is less than 50.0 ms. The default RPI is 5.0ms



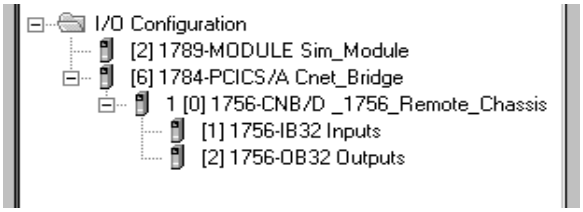
Click Finish

IMPORTANT You must specify an RPI of at least 50.0 ms for each 1789-SIM module or the connection to the module will fail. Because this module uses the generic module profile, the default RPI is 5.0 ms.

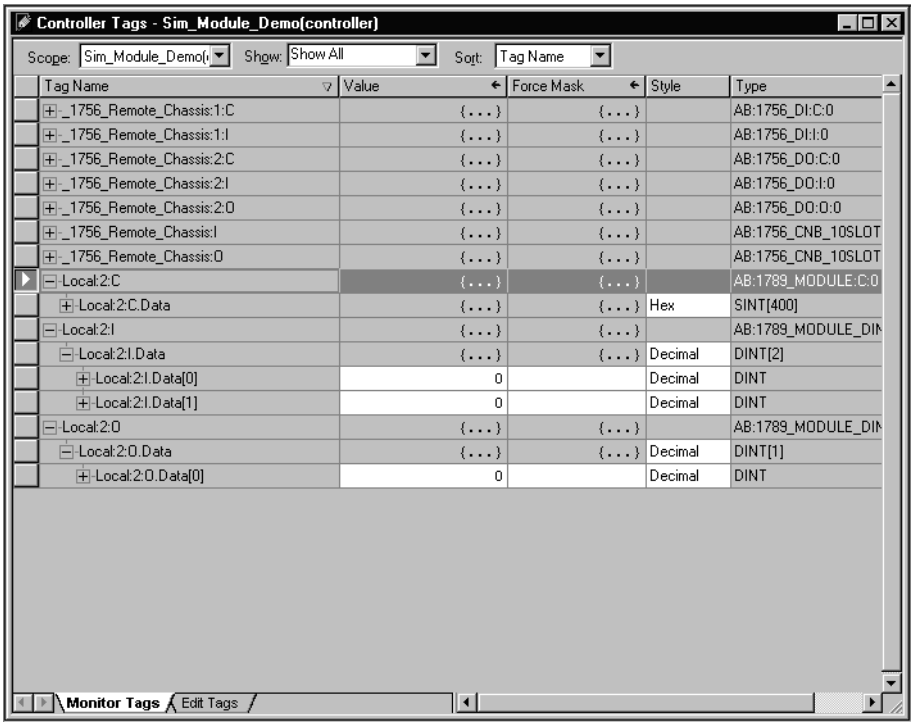
Mapping I/O Data to the 1789-SIM Module

When you add a 1789-SIM module to an RSlogix 5000 project, the programming software automatically assigns input and output data structures for the module. For example, this I/O configuration generates these I/O data structures:

The 1789-SIM module is in slot 2.



The programming software assigns these controller-scoped tags to the 1789-SIM module in slot 2.

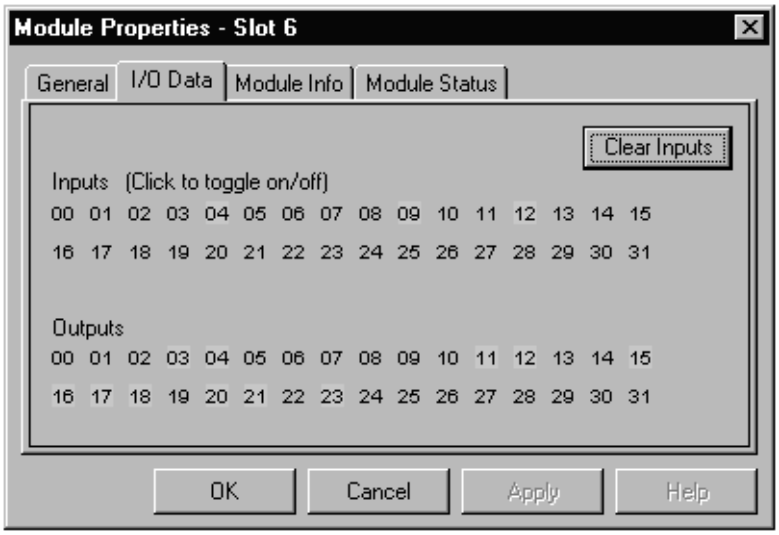


Toggling Inputs and Monitoring Outputs

Once the 1789-SIM module is installed in the chassis monitor, you can monitor the module.

1. In the chassis monitor, right-click the 1789-SIM module and select Properties.

2 Select the I/O Data tab.



Select the I/O Data tab of the module properties. Left mouse click a specific input bit to toggle it on or off.

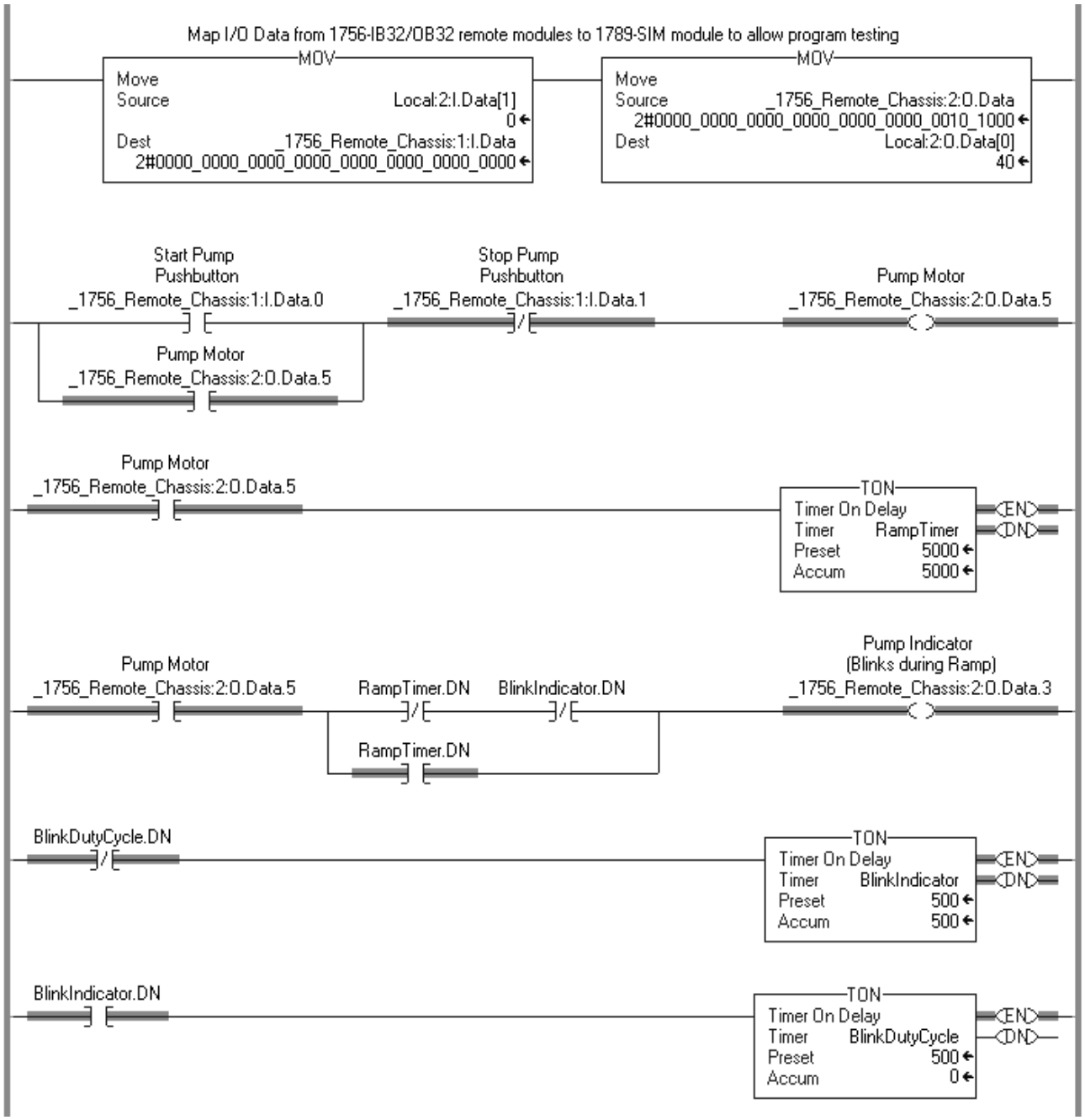
This tab also shows the state of the output bits. This is the same state that is displayed when you open the module door from the chassis monitor.

Outputs remain in last state when the controller is in program mode (this is not user configurable). If the I/O connection is broken to the module, all of the outputs will reset to OFF.

Example: Moving Application Data into the 1789-SIM Tags

The following example uses MOV instructions to copy:

- the input data from the 1789-SIM module into the application's input tags
- the application's output tags into the output data for the 1789-SIM module



Using External Routines

Using This Chapter

External routines are programs or functions developed outside of the RSLogix 5000 programming environment using commonly available programming languages, such as C or C++. If an external routine is properly developed as a Windows DLL, the SoftLogix controller can execute the routine as part of an RSLogix 5000 project.

Configuring Your System to Execute an External Routine

For the SoftLogix controller to execute an external routine, you need to:

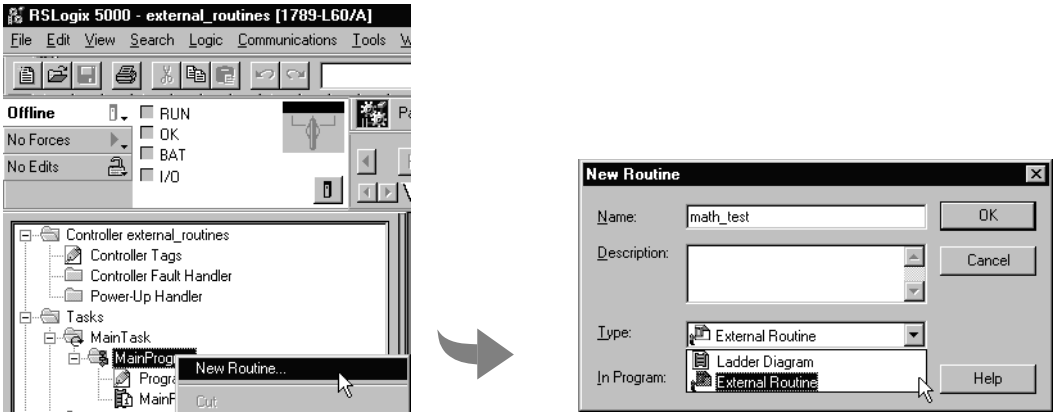
- add the external routine to the Controller Organizer
- use a JXR instruction within a relay ladder routine to call the external routine

Adding an External Routine to the Controller Organizer

You add an external routine to the Controller Organizer the same way you create a new ladder routine.

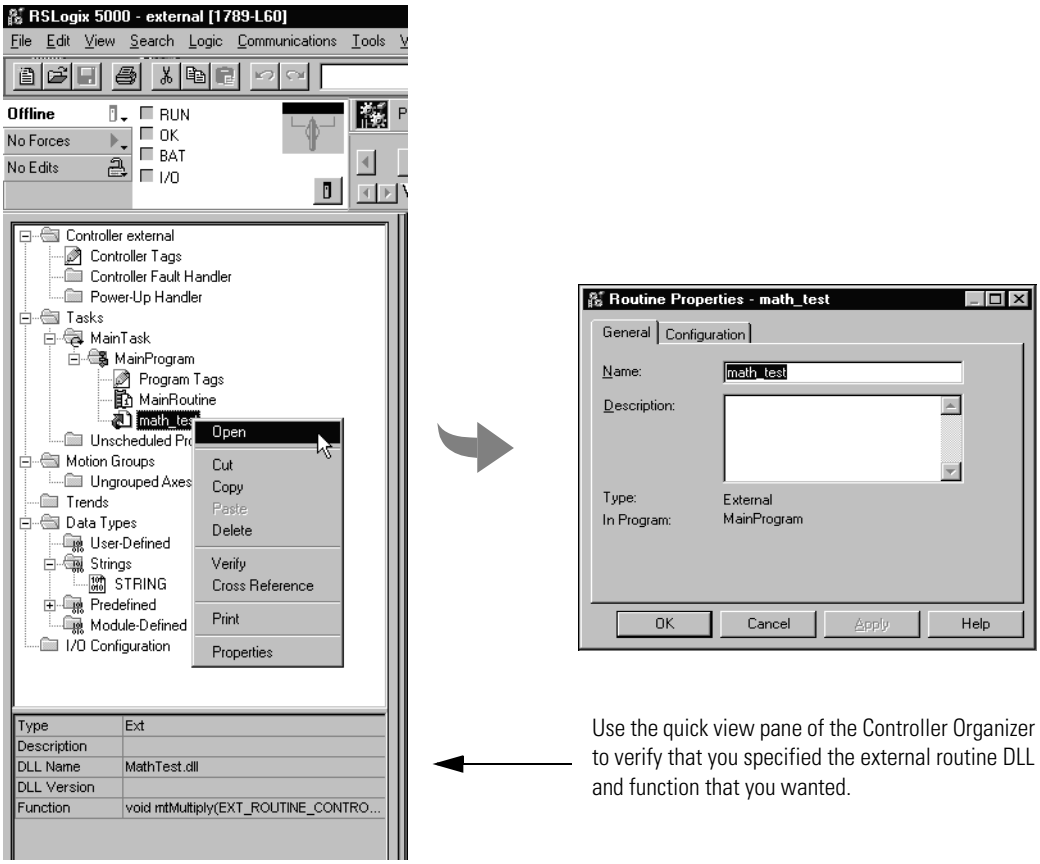
1. In RSLogix 5000 programming software, select the Main Program folder

2 Right-click to select New Routine and specify the routine.



Click OK

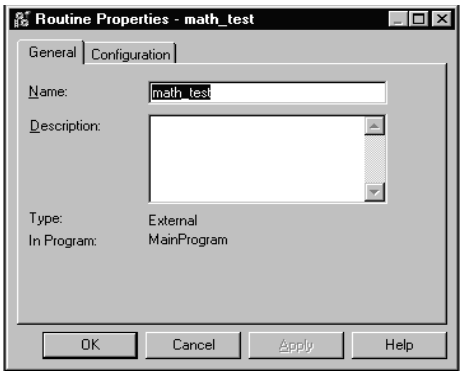
3. Select the routine and right-click to select open so you can specify which function in the DLL to execute.

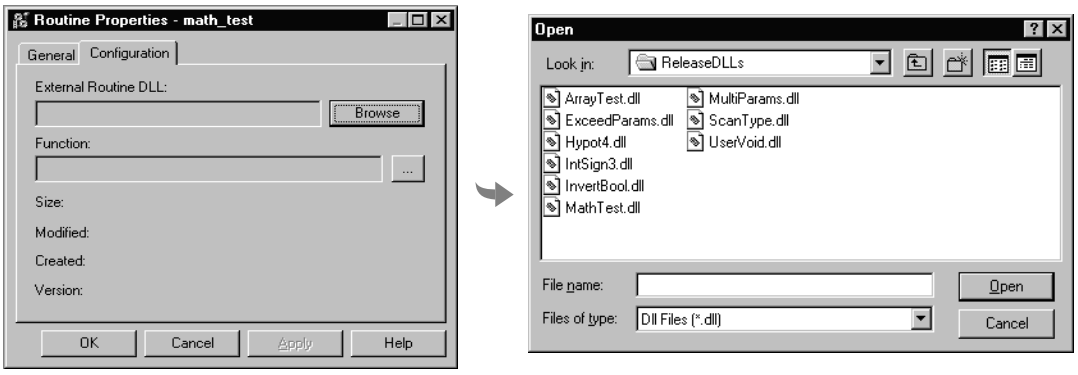


Use the quick view pane of the Controller Organizer to verify that you specified the external routine DLL and function that you wanted.

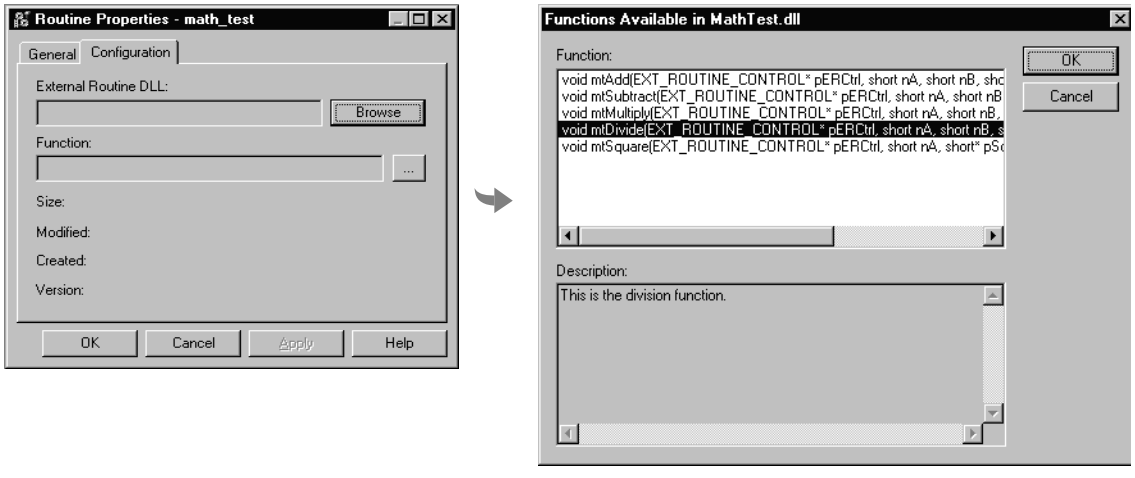
When specifying which DLL to execute:

On this tab:	Do this:
General	Verify the general information for the external routine is correct, as it should appear in the Controller Organizer.
Configuration	1. Browse to select the DLL file that contains the function you want to execute.





2. Select which function to execute within the DLL.



How the project stores and downloads an external routine

To use an external routine, you must associate (also known as “map”) a DLL file to an external routine that you create in the Controller Organizer of an RSLogix 5000 project (as shown on the previous two pages). You select the DLL file that contains the function you want to execute. RSLogix 5000 software makes a copy of that DLL and stores it in the external routine folder located in the RSLogix 5000 project directory, in a sub-folder named the same as the project file. For example, if the project MyProject.ACD is in C:\RSLogix5000\Projects, mapping a DLL to that project stores a copy of the DLL file in the directory C:\RLogix5000\Projects\ExternalRoutines\MyProject\.

When the you download the project to the controller, the mapped DLL is also downloaded to the target controller and a copy of the DLL is placed in the slot directory of the controller. For example, if you download MyProject.ACD to a controller in slot 4, the external routine DLL file is downloaded to the location C:\Program Files\Rockwell Automation\SoftLogix5800\Data\slot04 on the SoftLogix controller.

Because this process creates copies of the original DLL file, you can execute different versions of the same DLL on SoftLogix controllers in different slots of the same virtual chassis. The DLL used by a controller in one slot is completely independent of the DLL used by a controller in another slot. For this reason, if you update a DLL, you re-map the DLL in each RSLogix 5000 project and re-download the updated projects to the appropriate controllers.

Linking an individual DLL file to a specific controller and slot can be useful for debugging changes or testing new versions of an external routine. You can load different versions into controllers in different slots without having to actually update controllers that are performing plant control.

IMPORTANT If you want to use a copy of an RSLogix 5000 project on another workstation, take care when making a copy of a project that includes an external routine. In addition to the ACD file, you must also copy the external routines folder that contains all of the DLL files associated with that ACD file.

Calling an External Routine

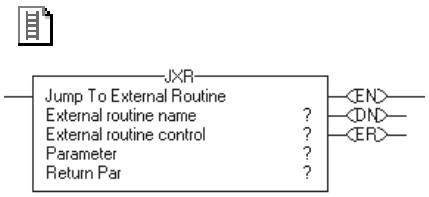
Use the Jump to External Routine (JXR) instruction to call the external routine from a ladder routine in your project. The JXR instruction supports multiple parameters so you can pass values between the ladder routine and the external routine.

Jump to External Routine (JXR)

The JXR instruction executes an external routine. This instruction is only supported by the SoftLogix5800 controllers.

Operands:

Relay Ladder



.

Operand:	Type:	Format:	Description:
External routine name	ROUTINE	name	external routine to execute
External routine control	EXT_ROUTINE_ CONTROL	tag	control structure (see the next page)
Parameter	BOOL SINT INT DINT REAL structure	immediate tag array tag	data from this routine that you want to copy to a variable in the external routine • Parameters are optional. • Enter multiple parameters, if needed. • You can have as many as 10 parameters.
Return parameter	BOOL SINT INT DINT REAL	tag	tag in this routine to which you want to copy a result of the external routine • The return parameter is optional. • You can have only one return parameter

EXT_ROUTINE_CONTROL Structure

Mnemonic:	Data Type:	Description:	Implementation:
ErrorCode	SINT	If an error occurs, this value identifies the error. Valid values are from 0-255.	There are no predefined error codes. The developer of the external routine must provide the error codes.
NumParams	SINT	This value indicates the number of parameters associated with this instruction.	Display only - this information is derived from the instruction entry.
ParameterDefs	EXT_ROUTINE_ PARAMETERS[10]	This array contains definitions of the parameters to pass to the external routine. The instruction can pass as many as 10 parameters.	Display only - this information is derived from the instruction entry.
ReturnParamDef	EXT_ROUTIN_ PARAMETERS	This value contains definitions of the return parameter from the external routine. There is only one return parameter.	Display only - this information is derived from the instruction entry.
EN	BOOL	When set, the enable bit indicates that the JXR instruction is enabled.	The external routine sets this bit.
ReturnsValue	BOOL	If set, this bit indicates that a return parameter was entered for the instruction. If cleared, this bit indicates that no return parameter was entered for the instruction.	Display only - this information is derived from the instruction entry.
DN	BOOL	The done bit is set when the external routine has executed once to completion.	The external routine sets this bit.
ER	BOOL	The error bit is set if an error occurs. The instruction stops executing until the program clears the error bit.	The external routine sets this bit.
FirstScan	BOOL	This bit identifies whether this is the first scan after switching the controller to Run mode. Use FirstScan to initialize the external routine, if needed.	The controller sets this bit to reflect scan status.
EnableOut	BOOL	Enable output.	The external routine sets this bit.
EnableIn	BOOL	Enable input.	The controller sets this bit to reflect rung-condition-in. The instruction executes regardless of rung condition. The developer of the external routine should monitor this status and act accordingly.
User1	BOOL	These bits are available for the user. The controller does not initialize these bits.	Either the external routine or the user program can set these bits.
User0	BOOL		
ScanType1	BOOL	These bits identify the current scan type: **Bit Values: Scan Type:** 00 Normal 01 Pre Scan 10 Post Scan (not applicable to relay ladder programs)	The controller sets these bits to reflect scan status.
ScanType0	BOOL		

Description: The JXR instruction is similar to the Jump to Subroutine (JSR) instruction. The JXR instruction initiates the execution of the specified external routine:

- The external routine executes one time.
- After the external routine executes, logic execution returns to the routine that contains the JXR instruction.

Arithmetic Status Flags: Arithmetic status flags are not affected.

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
• an exception occurs in the external routine DLL • the DLL could not be loaded • the entry point was not found in the DLL	4	88

Execution: The JXR can be synchronous or asynchronous depending on the implementation of the DLL. The code in the DLL also determines how to respond to scan status, rung-condition-in status, and rung-condition-out status.

Type Checking

The following table describes the type checking that occurs between RSLogix 5000 software and the external routine for the parameters that they pass.

RSLogix 5000 Data Type:	C++ Type:	Passing Method	Type Checking:	Arrays:
BOOL	bool	by reference by value	strong type checking	by reference Array sizes are not checked, but the size information is passed through the control structure
INT	short			
DINT	long int			
SINT	char			
REAL	float			
user-defined structure	structure	by reference	weak type checking The check only determines whether a structure is being passed to a structure parameter. Data types and sizes are not checked.	by reference Array sizes are not checked, but the size information is passed through the control structure

For more details on creating a DLL using Visual Studio, see chapter 10 *Developing External Routines*.

Notes:

Developing External Routines

Using This Chapter

This chapter shows how to use Microsoft Visual Studio to create an external routine. A SoftLogix5800 controller executes such an external routine as specified by a JXR instruction.

Considerations when using external routines

The external routines feature is an extremely flexible and powerful capability of the SoftLogix5800 product. The routines can be written in C or C++ using any commercial off-the-shelf development tool, such as Microsoft Visual Studio, that can generate a Windows compatible DLL (dynamic link library). The SoftLogix controller at runtime performs a “LoadLibrary” to invoke the external routine DLL's code from within the memory and process space of the SoftLogix5800 controller.

Because the user's external routine DLL runs in the memory and process space of the SoftLogix5800 controller, it is possible that incorrectly written user code can errantly overwrite memory locations that are being used by the controller. Care must also be taken when creating threads and assigning priorities since this can also impact the operation of the SoftLogix5800 controller. If proper procedures are not followed, it is possible that the controller may respond in an unpredictable manner.

Due to the requirements of this feature it is not possible for Rockwell Automation to safeguard and protect from certain scenarios that may interfere with the operation of the controller and result in unpredictable behavior.

Carefully read and follow the recommendations in this chapter. This chapter describes how to test an external routine in the Microsoft debugger and/or run it under normal operation with the controller placed in "Test Mode" so that outputs are not energized. Thorough testing must be performed in Test Mode prior to running the external routine with the controller's outputs enabled.

Because of the variety of uses for external routines, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that the application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards. Rockwell Automation does not assume responsibility or liability (to include intellectual property liability) for actual use of the external routines feature in a control system application.

How the SoftLogix Controller Uses External Routines

An external routine lets the SoftLogix controller execute a function that was developed outside of the RSLogix 5000 programming environment. The external routine must be a standard Windows DLL and it can contain one or more functions. An external routine can execute synchronously or asynchronously, depending on how the code is written in the external routine DLL. You can develop the DLL in C or C++ and export the functions of that DLL in C or C++.

You make these exported functions available to the SoftLogix controller by including XML information. The XML information describes the exported function.

Once you add an external routine to an RSLogix 5000 project, the DLL is downloaded to the SoftLogix controller when you download the project. The SoftLogix controller loads the DLL using a LoadLibrary Windows call during the download process. The SoftLogix controller uses a GetProcAddress Windows call to locate the exported function so that the function can be executed by an associated JXR instruction in the ladder program.

The external routine is executed in the process space of the SoftLogix controller. Spawning threads and processes are techniques you can use to make the JXR instruction for the external routine execute asynchronous to the ladder scan. If you spawn a thread, the external routine and the spawned thread both run in the process space of the controller. If you spawn another process, the external routine runs in the process space of the controller while the newly spawned process runs in its own process space.

How the project stores and downloads an external routine

To use an external routine, you must associate (also known as “map”) a DLL file to an external routine that you create in the Controller Organizer of an RSLogix 5000 project (as shown in chapter 9). You select the DLL file that contains the function you want to execute. RSLogix 5000 software makes a copy of that DLL and stores it in the external routine folder located in the RSLogix 5000 project directory, in a sub-folder named the same as the project file. For example, if the project MyProject.ACD is in C:\RSLogix5000\Projects, mapping a DLL to that project stores a copy of the DLL file in the directory C:\RLogix5000\Projects\ExternalRoutines\MyProject\.

When the you download the project to the controller, the mapped DLL is also downloaded to the target controller and a copy of the DLL is placed in the slot directory of the controller. For example, if you download MyProject.ACD to a controller in slot 4, the external routine DLL file is downloaded to the location C:\Program Files\Rockwell Automation\SoftLogix5800\Data\slot04 on the SoftLogix controller.

Because this process creates copies of the original DLL file, you can execute different versions of the same DLL on SoftLogix controllers in different slots of the same virtual chassis. The DLL used by a controller in one slot is completely independent of the DLL used by a controller in another slot. For this reason, if you update a DLL, you re-map the DLL in each RSLogix 5000 project and re-download the updated projects to the appropriate controllers.

Linking an individual DLL file to a specific controller and slot can be useful for debugging changes or testing new versions of an external routine. You can load different versions into controllers in different slots without having to actually update controllers that are performing plant control.

IMPORTANT If you want to use a copy of an RSLogix 5000 project on another workstation, take care when making a copy of a project that includes an external routine. In addition to the ACD file, you must also copy the external routines folder that contains all of the DLL files associated with that ACD file.

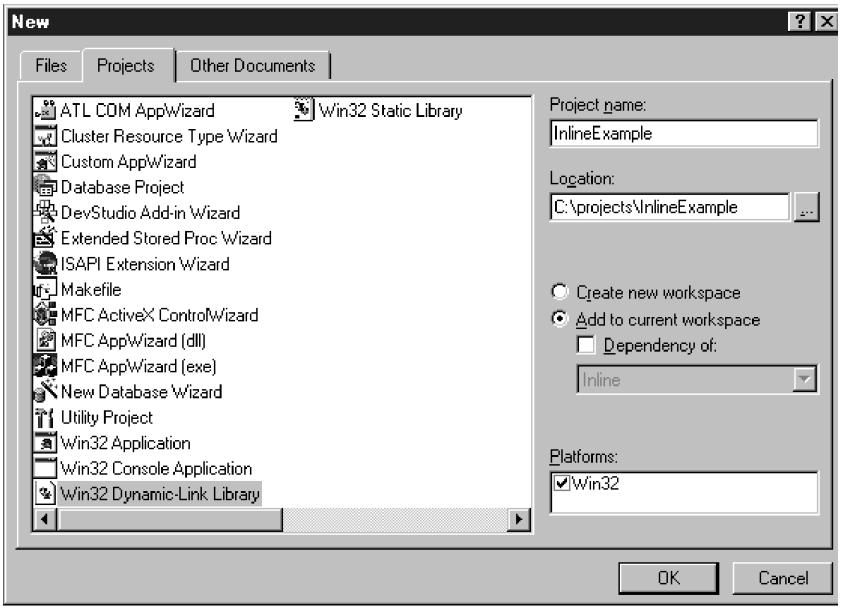
Creating Synchronous, Single-Threaded External Routines

This type of external routine runs synchronously in the process space of the control engine. Use this type of routine when the execution time of the function does not significantly impact the overall ladder scan time or cause a watchdog fault in the controller. A watchdog fault is a major fault that occurs because a scan of the routine did not complete within the expected amount of time.

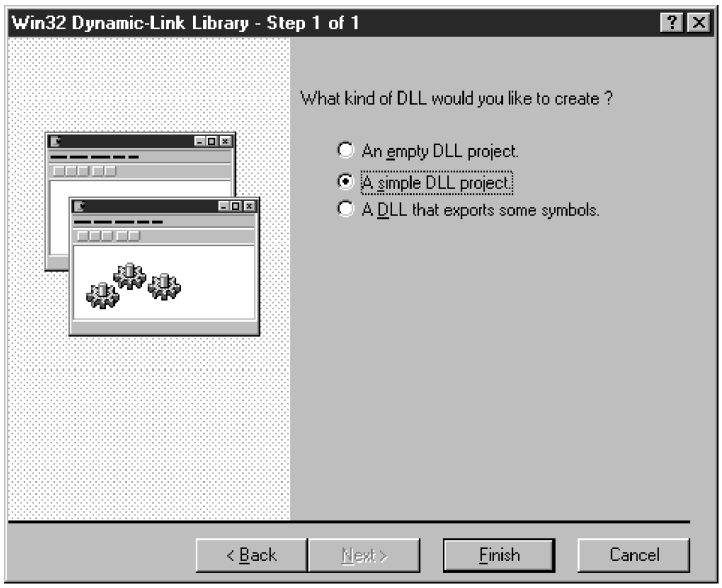
Create a Visual Studio project

When you create a new project in Visual Studio, select either a Win32 Dynamic-Link Library or MFC AppWizard (DLL).

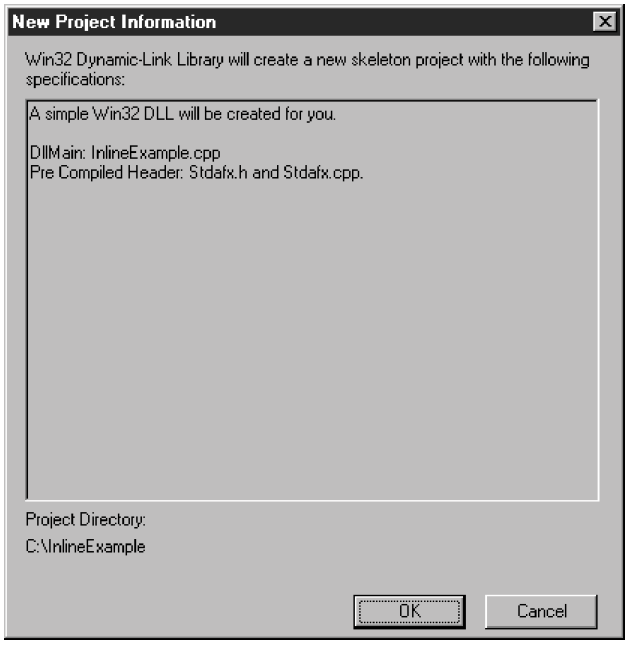
1. Create a new project.



2. Select the type of DLL project.



3. The software displays the type of files it will generate for the project.



Editing the Files in the Project

Add external routine code to the project files. All calls to external routines require that the user pass an external routine control structure as the first parameter in the call. The DLL developer must use the Rockwell supplied header file that describes the control structure. Below is the text of the header file along with a description of how various parts of the control structure should be used by the external routine DLL developer.

RA_ExternalRoutines.h

```
#ifndef __RA_EXTROUTINE_H__
#define __RA_EXTROUTINE_H__

#define MAX_PARAMS  10

/*           MSC assumes LSB first, 32 bit integers */

#pragma pack(push,1)

struct RoutineControlWord               // 4 bytes (32 bit word) total
{
       unsigned ErrorCode : 8;          // Error code if ER bit is set.
              // -- end byte 0 --
       unsigned NumParams : 8;          // From 0 to MAX_PARAMS,
              // -- end byte 1 --       excludes control structure
       unsigned ScanType : 2;           // 16-17 Normal, Pre, Post (0, 1, 2)
       unsigned ReservedA : 2;          // 18-19 Reserved Set A: DO NOT USE
       unsigned User : 2;               // 20-21 Defined by Ext Rtn developer
       unsigned ReservedC : 2;          // 22-23 Reserved Set C: DO NOT USE
              // -- end byte2 --
       unsigned EnableIn : 1;           // 24 Incoming rung status
       unsigned EnableOut : 1;          // 25 Returning rung status
       unsigned FirstScan : 1;          // 26 First Normal Scan occurring
       unsigned ER : 1;                 // 27 Control ERROR
       unsigned ReservedB : 1;          // 28 Reserved Set B: DO NOT USE
       unsigned DN : 1;                 // 29 Control DONE
       unsigned ReturnsValue : 1;       // 30 Indicates if routine returns anything
       unsigned EN : 1;                 // 31 Control ENABLE
              // -- end byte 3 --
};
#pragma pack(pop)
enum EXT_ROUTINE_PARAM_TYPE_E  // 4 bytes long

{
              FloatingPointValue = 0,                       // e.g., float p
              FloatingPointAddress,                         // e.g., float* p
              IntegerValue,                                 // e.g., short p
              IntegerAddress,                               // e.g., long* p
              ArrayAddress,                                 // e.g., int p[]
              StructureAddress,                             // e.g., MyStructT* p
              VoidAddress,                                  // e.g., void* p
              Void,                                         // e.g., "No return value"
              LastEntryInEnum
};
```

```
// Structure representing the type of the parameter defined
// for the External Routine.
struct EXT_ROUTINE_PARAMETERS           // 12 bytes long
{
       // Size of parameter/array element in bits
       unsigned long bitsPerElement;

       // If array, number of elements else 1.
       unsigned long numberOfElements;

       // Numeric representation and reference type.
       EXT_ROUTINE_PARAM_TYPE_E paramType;
};
```

The control structure for an external routine contains control, status, and meta information for that routine. The control structure is accurate at the time of its invocation and it enables the external routine to validate and influence its operation. Upon routine completion, the control structure contains status about the routine's execution as to its degree of success or failure. Other data areas are either reserved or defined by the user.

The meta information includes the definitions of the parameters passed to the external routine (paramDefs) and the parameter that is returned from the routine (returnDef). The meta information is read-only. It is derived at download time and then set upon every call to its corresponding external routine. These definitions let the external routine determine if it is being used in the proper context. The routine can check this information, which includes a parameter's type, the number of elements in the event of an array, and the number of bits in each of these elements.

The remaining meta information exists in the ctrlWord: the number of parameters (numParams), whether a return value (ReturnsValue) is expected by the caller, and whether it's executing during Normal, Pre, or Post scan (ScanType). All the meta information is set by the system and the external routine developer should treat it as read-only. Any modifications to this information is disregarded.

Control and status information exist in one location inside the RoutineControlWord structure (ctrlWord) within the external routine control. Control information consists of EnableIn and FirstScan. This information is set upon every invocation of the external routine. EnableIn reflects the rung status at the time of the call. FirstScan indicates whether this is the first scan after switching the controller to run mode. Use EnableIn to enable operation of the external routine and use FirstScan, or possibly Prescan, to perform any necessary initialization in the external routine.

Status information should be set by the external routine. The Enable (EN), Done (DN), and Error (ER) bits are used much like other ladder control structures. Set the EN bit when the external routine is enabled. Set the DN bit when the operation is complete. If an error occurs during execution, set the ER bit and store an error code in the ErrorCode member.

Another piece of status information is EnableOut. Set this bit if the external routine is used to compute rung status and the result indicates a TRUE condition. The system does not however update system rung status based on this information. It is used as an indication to the caller.

The controller does not modify or initialize the user defined bits (User). The use of these bits is up to the external routine developer.

```
// Fixed size structure defining JXR's signature and control.
struct EXT_ROUTINE_CONTROL              // 4 + 120 + 12 = 136 bytes long
{
             RoutineControlWord               ctrlWord;
             EXT_ROUTINE_PARAMETERS           paramDefs[MAX_PARAMS];
             EXT_ROUTINE_PARAMETERS           returnDef;
};
```

InlineExample.cpp

```
// InlineExample.cpp : Defines the entry point for the DLL application.

#include "stdafx.h"
//Include file for External Routine interface
#include "RA_ExternalRoutines.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
       {
             case DLL_PROCESS_ATTACH:
             case DLL_THREAD_ATTACH:
             case DLL_THREAD_DETACH:
             case DLL_PROCESS_DETACH:
             break;
       }
    return TRUE;
}
// This is an example of an exported function.
extern "C" __declspec(dllexport   int SumArray(
                                  EXT_ROUTINE_CONTROL* pERCtrl,
                                  int Val[])
{
       // Add all array elements provided, then return the Sum.
```

```
	pERCtrl->ctrlWord.EN = pERCtrl->ctrlWord.EnableIn;
	pERCtrl->ctrlWord.EnableOut = pERCtrl->ctrlWord.EnableIn;

	BOOL bFail = FALSE;

	// Number of parameters expected is 1 (exclude ctrl struct*), and
	// Check type of parameter against what is expected, and
	// Check size of array elements to make sure they agree with int type.
	if (pERCtrl->ctrlWord.NumParams != 1)
		bFail = TRUE;
	else if ( (pERCtrl->paramDefs[0].paramType != ArrayAddress) ||
		(pERCtrl->paramDefs[0].bitsPerElement != 8*sizeof(int)) )
		bFail = TRUE;

	// Check number of array elements
	int nNoElems = pERCtrl->paramDefs[0].numberOfElements;

	if (nNoElems == 0)
		bFail = TRUE;

	int itemp = 0;  // Initialize sum to zero

	if (pERCtrl->ctrlWord.EnableIn)
	{
		// Rung enabled, run the function's implementation
		if (!bFail)
		{
			// Sum all array elements
			for (int j = 0; j < nNoElems; j++)
				itemp += Val[j];

			// Set Error bit to zero if successful.
			pERCtrl->ctrlWord.ER = 0;
		}
		else
		{
			// Some error
			// Set Error bit to indicate error occurred
			pERCtrl->ctrlWord.ER = 1;
			pERCtrl->ctrlWord.ErrorCode = 1;  // Set ErrorCode
		}

		// Set Done bit before exit of this XR.
		pERCtrl->ctrlWord.DN = 1;
	}
	else
	{
		// Rung not enabled
		pERCtrl->ctrlWord.DN = 0;
	}

	return itemp;  // returns 0.0 if error
}
```

InlineExample.h

```
// Exported Functions:
extern "C" __declspec(dllimport   int SumArray(
                                  EXT_ROUTINE_CONTROL* pERCtrl,
                                  int Val[]);
```

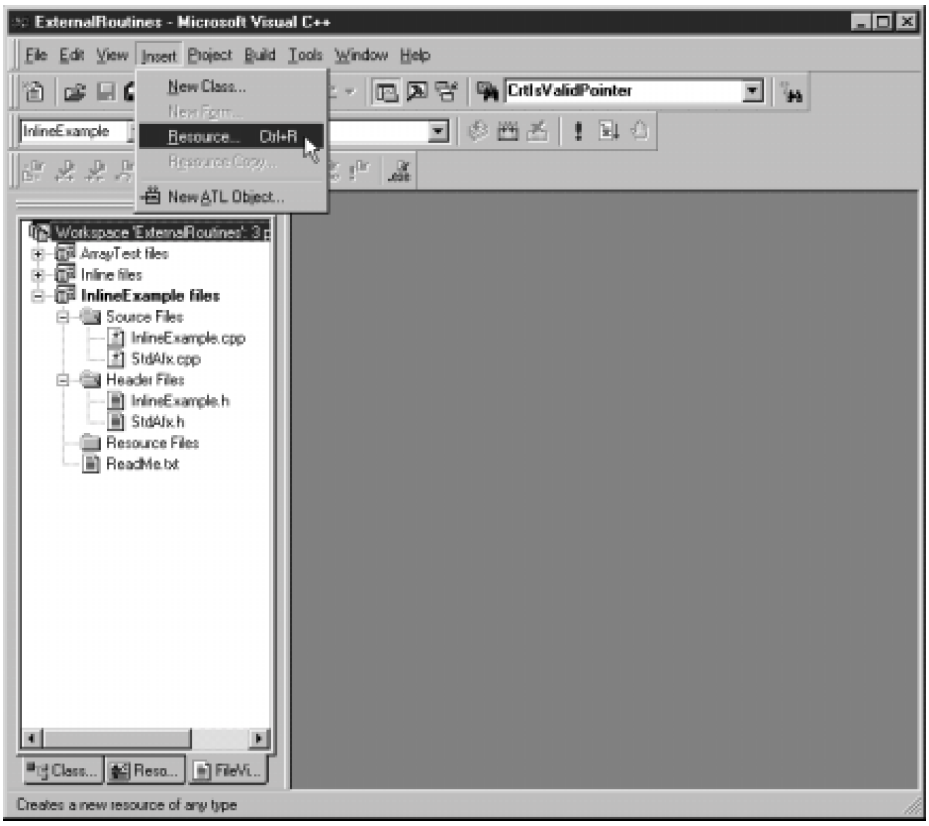
Creating an HTML Resource

The HTML resource:

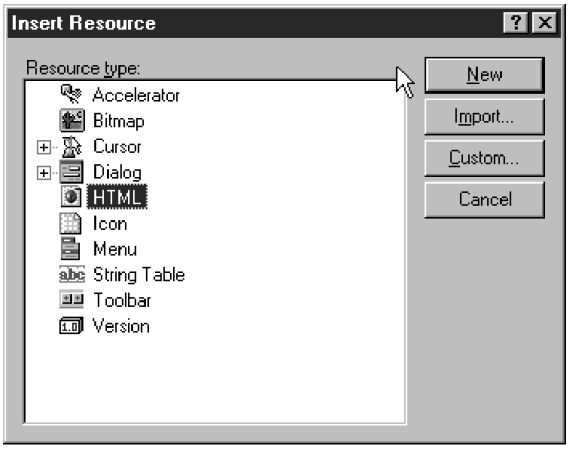
- describes the external routines which are contained in the DLL.
- provides descriptions that RSLogix 5000 software uses.
- provides type checking in the JXR instruction.
- gets the name of the routine to be used in the GetProcAddress call.

The information stored in the HTML resource is in XML. Use an HTML resource because Visual Studio does not support an XML resource.

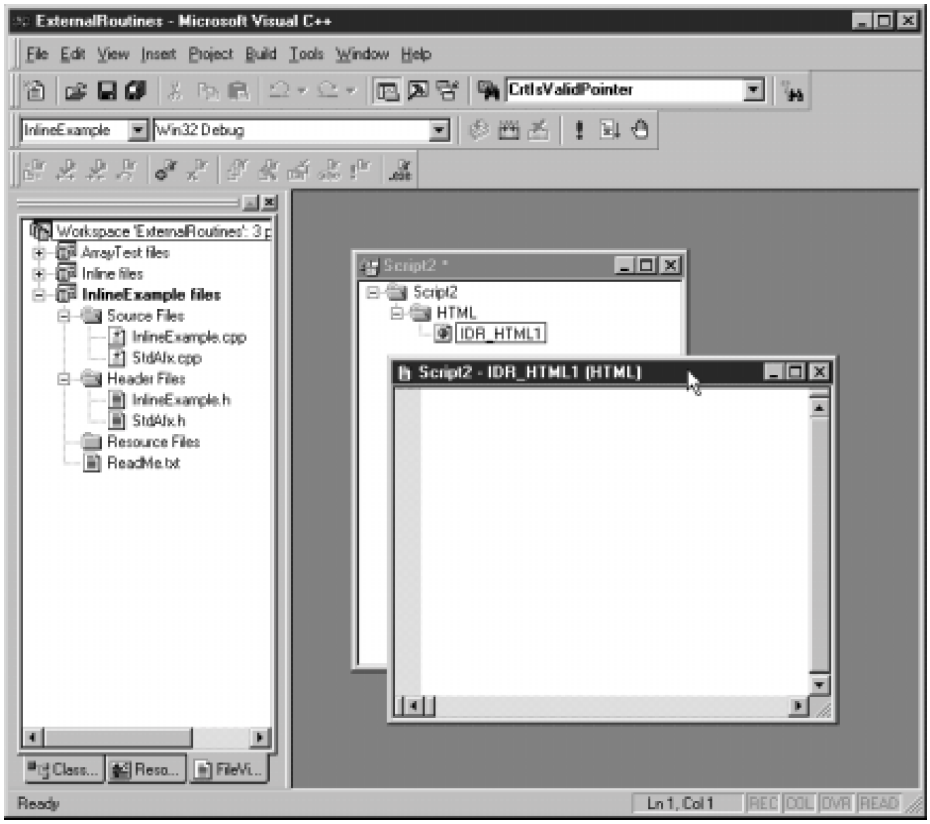
1. Insert a new resource.



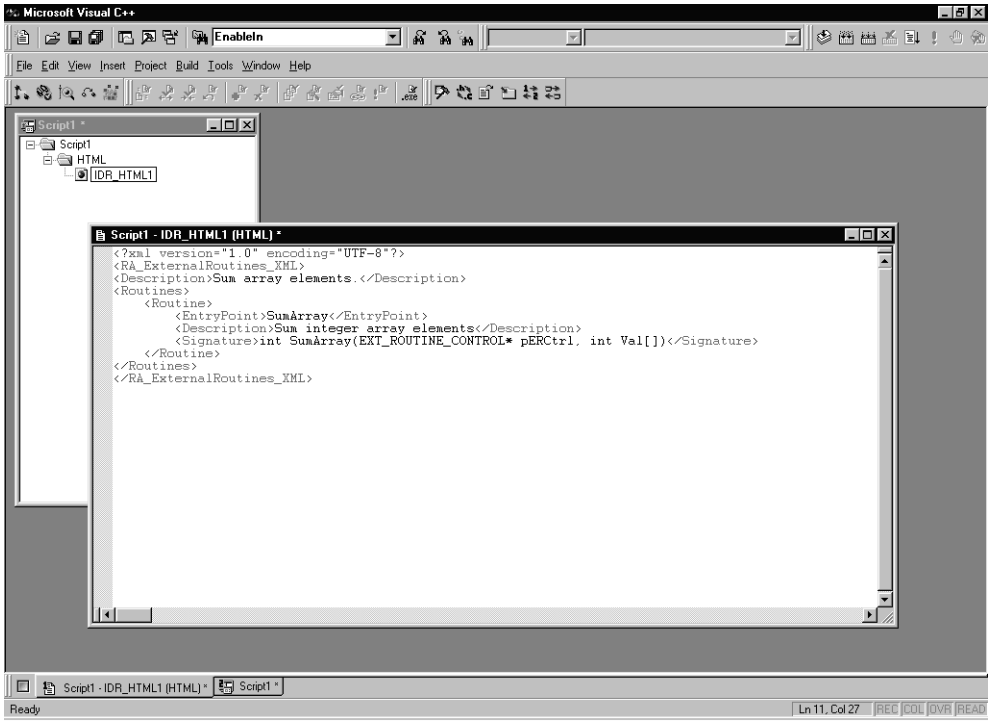
2. Select HTML as the resource type.



3. View the new resource.



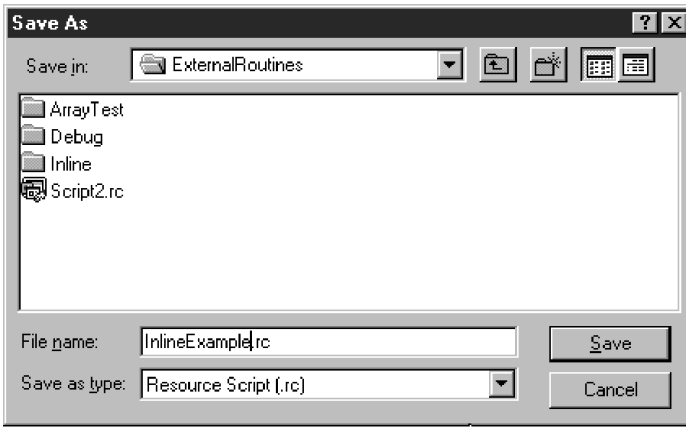
4. Edit the HTML file and put in the XML descriptions.



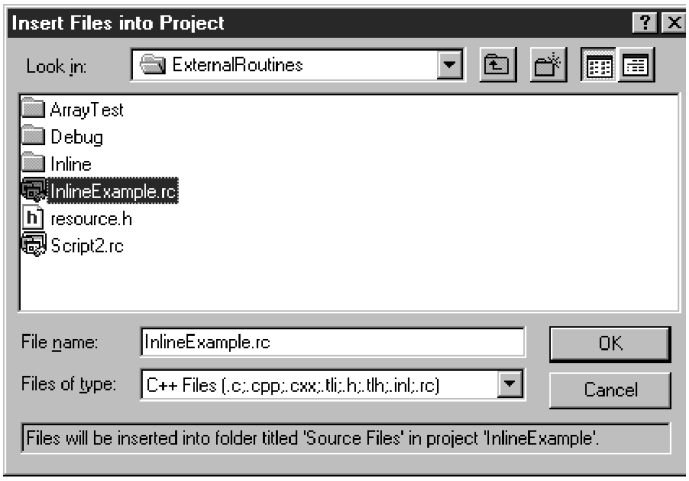
Edit the HTML file and put in the XML descriptions of the external routines. The following tags are required:

Tag:	Description:
<RA_ExternalRoutines_XML>	This tag indicates that the following information is related to external routines.
<Description>	This tag documents the type of routines that are contained in the DLL. The information provided here is completely up to the developer. This information is not used by RSLogix 5000 software and is not displayed to the user. This information is for internal documentation purposes only.
<Routines>	This tag indicates that the following information contains the description of the exported functions in the DLL.
<Routine>	This tag describes information that relates to one external routine. You can have more than one exported function per DLL, so you can have multiple <Routine> blocks. There is one <Routine> tag for each exported function in the DLL. Each <Routine> tag contains the following XML tags. <EntryPoint>, <Description>, and <Signature>.
<EntryPoint>	This tag contains the name of the function that is exported from the DLL. If the routine is exported using C style exporting, then the name is the same as the name of the function. If the routine is exported using C++ style exporting, then the name needs to match the C++ decorated name exported by the C++ compiler. Details on how to obtain the C++ decorated name are described in a later section of this document.
<Description>	This tag documents the functionality of an individual function. The information provided here is at the discretion of the developer. This description is displayed to the user by RSLogix 5000 software during the mapping procedure.
<Signature>	This tag describes the interface to the routine, its parameters, and its return value. This tag is used for two purposes: to display to the user during the mapping process and to verify parameters in the JXR instruction during verification of the project in RSLogix 5000 software. The number of parameters and parameter types must match exactly with what the routine requires.

Save the HTML resource. When saving the resource rename the resource from the default name to something more meaningful, like the name of the project.

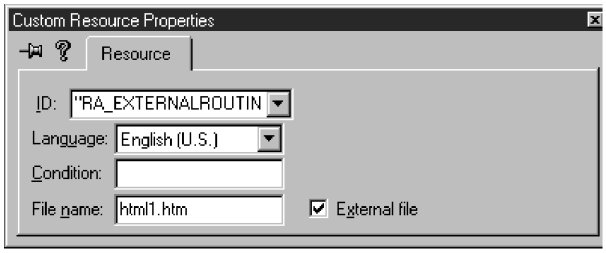


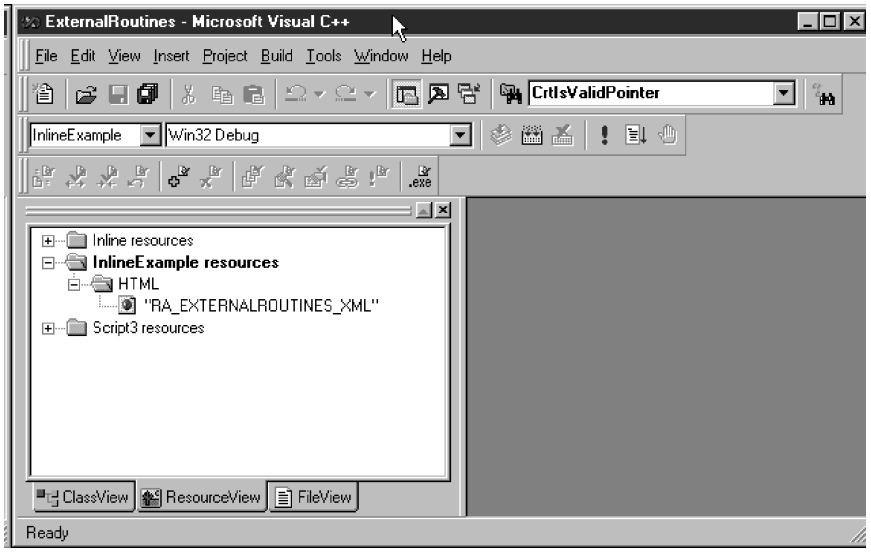
Insert the HTML resource into the project.



Change the name of the resource to “RA_EXTERNALROUTINES_XML”. Do this by editing the ID field of the resource properties box. Note that the quotes are required.

The ResourceView should look like this after you name the resource.

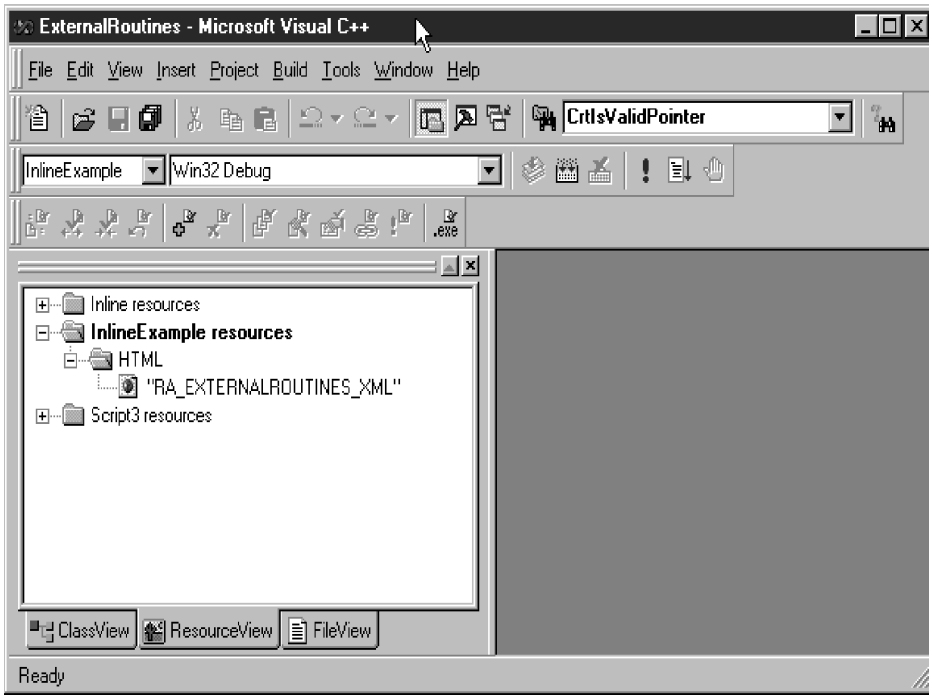




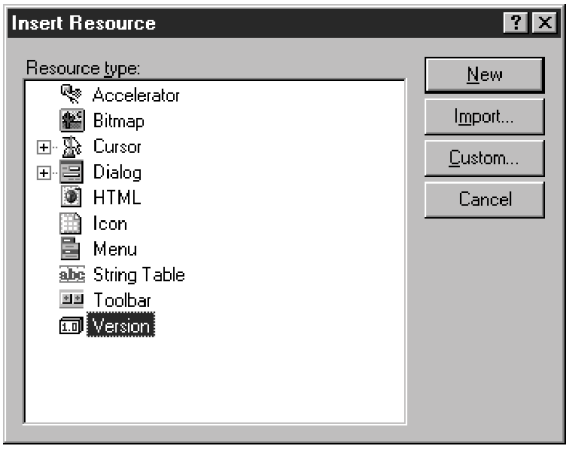
Adding Version Information to an External Routine DLL

Add version information to your DLL to keep track of your development changes.

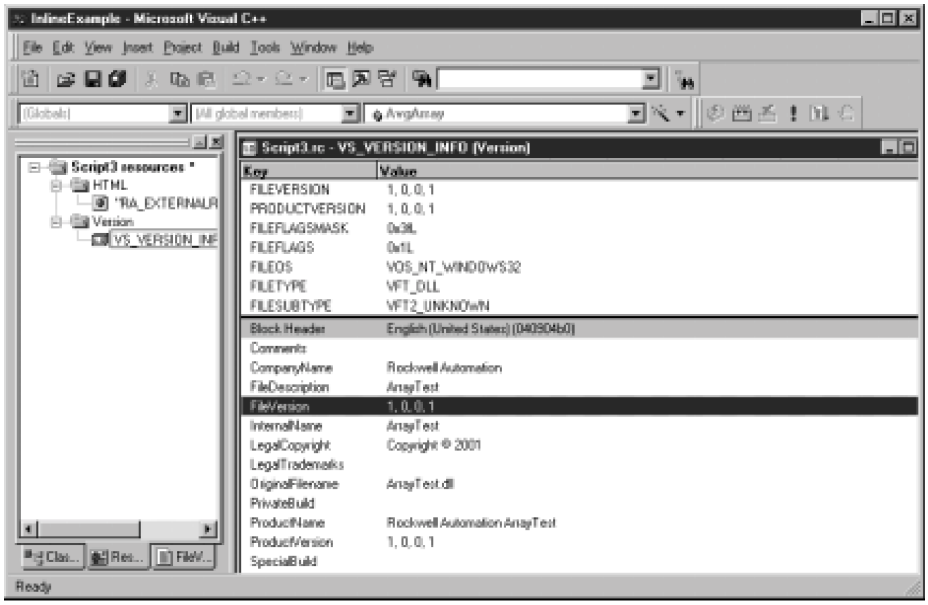
1. Select to insert information.



2. Select version.



3. Select the FileVersion field.



RSLogix 5000 software uses the FileVersion field under the Block Header English (United States) to display version information for the external routine DLL. The software only uses this field - not the FILEVERSION (all capital letters) or any other FileVersion field located in any other language sections.

The FileVersion field is a string and is completely under the control of the developer. Whatever you enter in this field is what RSLogix 5000 software displays.

RSLogix 5000 software displays this version information in two places:

- on the properties/configuration screen for the external routine.
- in the quick-view pane when the you places the cursor on an external routine within the Controller Organizer.

Building and Downloading External Routines

Before you build an external routine, make sure that RA_ExternalRoutines.h is in the include path for the project. Then:

1. Build the project.

2. Map the external routines into an RSLogix 5000 project.

3. Download the RSLogix 5000 project to a SoftLogix controller. The external routine DLL is downloaded as a part of the project download.

Updating an Existing External Routine

To update an existing external routine do the following:

1. Edit the source code in Visual Studio.
2. Rebuild the project.
3. Re-map the external routines into an RSLogix 5000 project.

   If you re-map one function from a DLL, all of the other functions that you use from the same DLL are also re-mapped. When a DLL is re-mapped, a re-verification is done on all of the routines that reference the DLL. Routines that reference a DLL different from the one re-mapped are unaffected by this process.

4. Re-download the RSLogix 5000 project to the SoftLogix controller.

The external routines must be re-mapped and the project must be re-downloaded to the controller in order for the changes to be made in the SoftLogix controller.

Creating Multithreaded External Routines

The following example shows an external routine which creates threads to do the majority of the processing. Consider using this approach when the amount of time required to execute is large enough that it could cause a watchdog fault in the controller. Applications which have disk I/O or, as in the following example, play *.wav files should be done in this manner.

The following example also shows how to synchronize the threads with the controller so that they can react properly to changes of state in the controller.

Sounds.cpp

```
// Sounds.cpp : Defines the entry point for the DLL application.

#include "stdafx.h"
#include "RA_ExternalRoutines.h"
#include <Mmsystem.h>
#include <process.h>

HANDLE       hTerminate = NULL;

// DllMain needs to create a global event which all threads need to check for.
// This event will be used to tell the threads that the DLL is being unloaded
// and that it is time to terminate.
//
// Not creating and using this event can lead to access violations in the threads which
// will cause the SoftLogix controller to terminate and display a red X across the module.

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
       hTerminate = CreateEvent (NULL, TRUE, FALSE, NULL);
       return (TRUE);

    case DLL_PROCESS_DETACH:
       SetEvent (hTerminate);
       Sleep (50); // give threads the chance to act on termination notice.
       return (TRUE);

    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
       return (TRUE);
    }

    return (FALSE);
}

typedef enum rungStates {
       FIRST_SCAN,
       RUNG_TRUE,
       RUNG_FALSE,
       INVALID_STATE
} RUNGSTATES;

RUNGSTATES rungState = INVALID_STATE;


HANDLE       hControllerState;
```

```
void PlaySound(char * Snd)
{
       HGLOBAL hResLoad;               // handle to loaded resource
       HRSRC hRes;                     // handle/ptr. to res. info. in hDLL
       LPTSTR lpResLock;               // pointer to resource data
       BOOL bRes = TRUE;

       // By building the sound resourced into the dll and doing a load
       // library on the dll we ensure that the resources are written
       // down to the controller with the dll. This means that we do not
       // have to worry about copying the resources to the controller on our own.
       HINSTANCE MyLib = LoadLibrary("Sounds.dll");
       if (MyLib == NULL)
       {
              return;
       }

       hRes = FindResource(MyLib, Snd, "SOUNDS");
       if (hRes == NULL)
       {
              return;
       }

       hResLoad = LoadResource(MyLib, hRes);
       if (hResLoad == NULL)
       {
              return;
       }

       lpResLock = (LPTSTR)LockResource(hResLoad);
       if (lpResLock == NULL)
       {
              return;
       }

       sndPlaySound(lpResLock, SND_SYNC | SND_MEMORY);

       FreeLibrary(MyLib);
}

void RungStateThread(void *p )
{

bool   exitThread = FALSE;
rungStates   oldRungState = rungState;
DWORD  status;
HANDLE       hController = p;
```

```
HANDLE       hArrayHandles [2];

       hArrayHandles[0] = hController;
       hArrayHandles[1] = hTerminate; // used to check if thread should terminate.

       while (!exitThread)
       {
              // This example uses an arbitrary timeout of 5 seconds
              // to determine whether the controller is still in run mode.
              // If this time expires, we can assume that the controller
              // is no longer in run mode, and we can terminate this thread.

              // Note:  The 5000 millisecond timeout value can be adjusted
              // to a value that fits the requirements for your specific
              // application.

              status = WaitForMultipleObjects (2, hArrayHandles, FALSE, 5000);
              switch (status)
              {
              case WAIT_OBJECT_0:
                     if (oldRungState != rungState)
                     {
                            oldRungState = rungState;
                            switch (oldRungState) {
                            case FIRST_SCAN:
                                   PlaySound("FIRSTSCAN");
                                   break;
                            case RUNG_TRUE:
                                   PlaySound("RUNGTRUE");
                                   break;
                            case RUNG_FALSE:
                                   PlaySound("RUNGFALSE");
                                   break;
                            default:
                                   exitThread = TRUE;
                            }
                     }
                     break;
              case WAIT_OBJECT_0 + 1:
                     exitThread = TRUE;
              case WAIT_TIMEOUT:
              case WAIT_ABANDONED:
              case WAIT_FAILED:
                     exitThread = TRUE;
              }
       }

       CloseHandle(hController);
       _endthread();
}
```

```
extern "C" __declspec(dllexport) void SayRungState(EXT_ROUTINE_CONTROL * pERCtrl)
{
       pERCtrl->ctrlWord.EN = pERCtrl->ctrlWord.EnableIn;

       rungState = INVALID_STATE;

       // Only create the thread on prescan.
       if (pERCtrl->ctrlWord.ScanType == 1)
       {
              hControllerState = CreateEvent(NULL, FALSE, TRUE, NULL);
              if (hControllerState)
              {
                     HANDLE hThread;

                     hThread = (HANDLE) _beginthread (RungStateThread,
                                   0,                          // stack size
                                   hControllerState);          // arglist
                     if (hThread != INVALID_HANDLE_VALUE)
                     {
                            // The following code will set the thread's priority to the
                            // same priority as the task that invoked the external routine.
                            // If the thread is not performing time-critical work, then
                            // it is recommended that you set its priority to
                            // THREAD_PRIORITY_IDLE, i.e.
                            // SetThreadPriority (hThread, THREAD_PRIORITY_IDLE);
                            SetThreadPriority (hThread,
                                          GetThreadPriority(GetCurrentThread()));
                     }
              }
       }
       else
       {
              if (pERCtrl->ctrlWord.FirstScan)
              {
                     rungState = FIRST_SCAN;
              }
              else if (pERCtrl->ctrlWord.EnableIn)
              {
                     rungState = RUNG_TRUE;
              }
              else
              {
                     rungState = RUNG_FALSE;
              }

              SetEvent(hControllerState);
       }

       return;
}
```

Thread priorities in a multithreaded external routine DLL

If you use Microsoft C or C++ to develop your external routine, only use the "_beginthread" C runtime function, as shown above in the example. Using the "_beginthread" and "_endthread" calls ensures that system resources are properly allocated by the Microsoft C runtime library.

When developing a multithreaded DLL, you must link with the multithreaded version of the C runtime library. Use the Multithreaded DLL option in your project settings for the run-time library option.

Keep in mind that the default priority assigned to a thread upon its creation is THREAD_PRIORITY_NORMAL. This priority level (priority level 2) is reserved for running the controller's periodic tasks. Set the priority of the newly created thread to something more appropriate so as not to interfere with the execution of periodic tasks in the controller. The example above shows how to set the thread priority to be equal to the priority of the parent thread, or THREAD_PRIORITY_IDLE if the thread does not perform any time-critical operations.

The following table shows the mapping between thread priorities and the controller's task priorities:

```
Continuous Task           THREAD_PRIORITY_LOWEST
Periodic Task (pri. 3)    THREAD_PRIORITY_BELOW_NORMAL
Periodic Task (pri. 2)    THREAD_PRIORITY_NORMAL
Periodic Task (pri. 1)    THREAD_PRIORITY_ABOVE_NORMAL
```

If you do not set the priority of a thread created via the "_beginthread" runtime function to the recommended values, periodic task overlap faults or watchdog faults can occur.

ATTENTION

Do not set the priority of any thread to a value greater than THREAD_PRIORITY_ABOVE_NORMAL because it will interfere with the operation of critical controller system threads and may result in unpredictable behavior of the controller.

Debugging External Routines

Setting up the debug session

If you built your DLL with Program Database Symbolic Information (PDB), you must copy the PDB file into the directory where the External Routine DLL is copied during a download to the controller. This file is not copied as a part of the normal download to the SoftLogix controller.

For example: If you have a controller in slot 3 that is using an External Routine DLL called InlineExample.dll, copy InlineExample.pdb to:

C:\Program Files\Rockwell Automation\SoftLogix5800\data\slot03.

The location "C:\Program Files\Rockwell Automation\SoftLogix5800" is where the SoftLogix controller is installed. The location "\data\slot03" is where data related to a particular instance of the controller resides. The slot directory is created as needed when a controller is inserted into the chassis monitor.

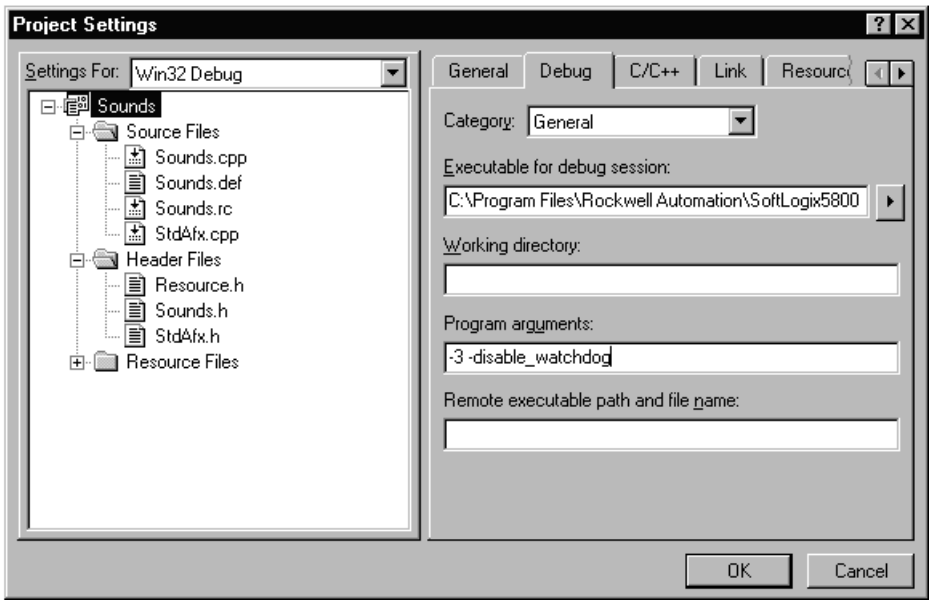
Starting a debug session

To debug an external routine, execute the SoftLogix controller (SoftLogix5800.exe) from the Visual Studio debugger. Create an empty project and edit the project settings.

ATTENTION

Do not attempt to control equipment while debugging external routines.

1. Set the debug dialog as follows.



Debug dialog item:	Description:
executable for debug session	C:\Program Files\Rockwell Automation\SoftLogix5800\ SoftLogix5800.exe
program arguments	*-n* -disable_watchdog Where *n* is a number that represents the slot number of the controller. Make sure to include a space between both arguments For example: -3 -disable_watchdog. The 3 flag indicates that the controller is in slot 3. The disable_watchdog flag lets you run the controller in the debugger and it disables the watchdog so you can step through your code without faulting the controller. This flag also forces the controller to run in Test mode (transition to Run mode is disabled), which causes output modules to be set to their Program mode.

Make sure you select the Softlogix5800.exe executable from the install path for Softlogix 5800 (by default, the Rockwell Automation folder)

Ensure you disable the watchdog timeout by entering -(slot#) -disable_watchdog (as shown above).

2. Perform a Build → Clean and then a Build → Batch Build. Verify that both Debug and Release are selected and then Rebuild All.

   This should produce debug information within the DLL.

3. Copy the *external_routine*.pdb file produced by visual studio to the Softlogix5800\data*slot#* folder.

4. Map over the DLL file produced into the RSLogix 5000 Project for all JXR instructions

5. Download the project to the controller and go offline

6. Remove the controller from the chassis.

7. Go back to Visual Studio and set your break points (see below).

8. Start the Visual Studio Project into Debug Mode by pressing F5.

9. If you watch the Virtual Chassis Monitor, you should see a SoftLogix controller be inserted into the chassis.

   Now you can step through your code in visual studio

Setting breakpoints in external routine code

When you download a project to the controller, this loads (or re-loads) the DLLs containing your external routines. Once the external routine DLLs are loaded, you can set breakpoints in any of the external routines.

Data Type Support

RSLogix 5000 Data Type:	C++ Data Type:	Passing Method:	Type Checking:
BOOL	bool	by reference by value	strong
INT	short	by reference by value	strong
DINT	long int	by reference by value	strong
SINT	char	by reference by value	strong
REAL	float	by reference by value	strong
UDT	structure	by reference	weak The type declaration must have the struct keyword.
others	void*	by reference	weak All nonliterals are accepted.

RSLogix 5000 arrays pass type checking if the external routine parameter is declared void * or uses “arrayType arrayName[]” convention. Arrays are passed by reference. Their element size and number are passed to an external routine via a control structure that can be checked at run time if desired by the external routine developer.

ARRAY example

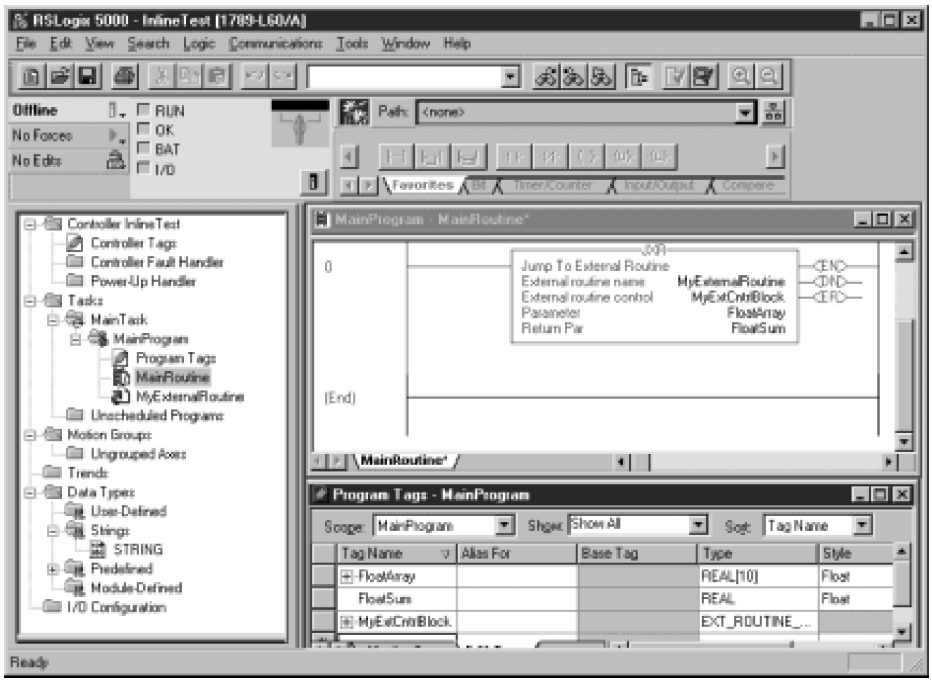
C declaration

```
extern "C" __declspec(dllexport) float SumArray(EXT_ROUTINE_CONTROL* pERCtrl,
                                                float Val[])
```

XML declaration

```
<Routine>
       <EntryPoint>SumArray</EntryPoint>
       <Description>Sum floating point array elements</Description>
       <Signature>float SumArray(EXT_ROUTINE_CONTROL* pERCtrl,
                                 float Val[])</Signature>
</Routine>
```

RSLogix 5000 declaration



INTEGER example

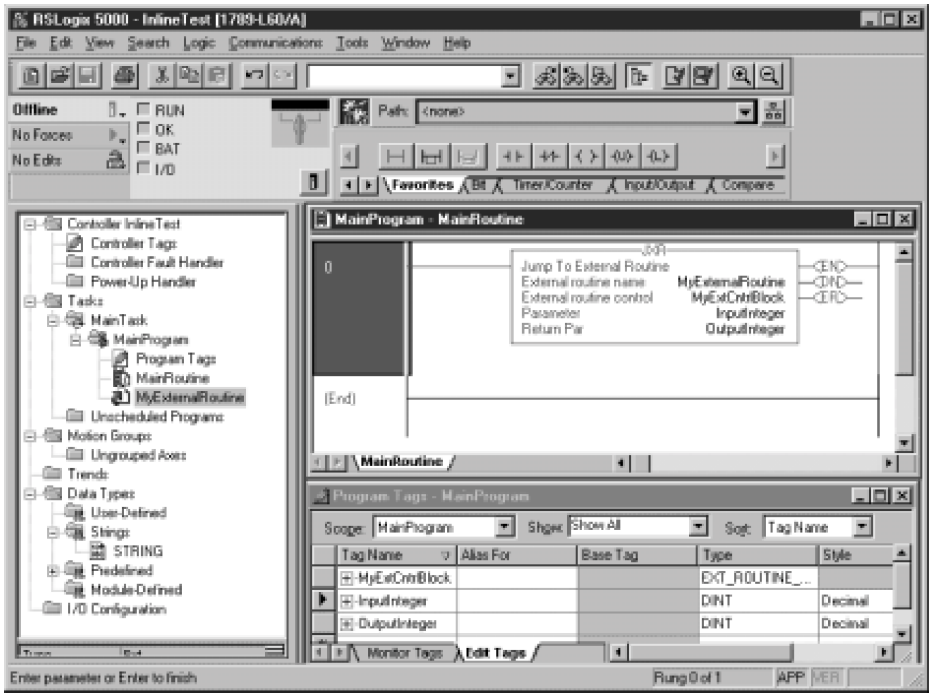
C declaration

```
extern "C" __declspec(dllexport) int SomeCalculation(EXT_ROUTINE_CONTROL* pERCtrl,
                                                     int Val)
```

XML declaration

```
<Routine>
       <EntryPoint> SomeCalculation </EntryPoint>
       <Description>Do an important calculation</Description>
       <Signature>int SomeCalculation (EXT_ROUTINE_CONTROL* pERCtrl,
                                       int Val)</Signature>
</Routine>
```

RSLogix 5000 declaration



STRUCTURE example

C declaration

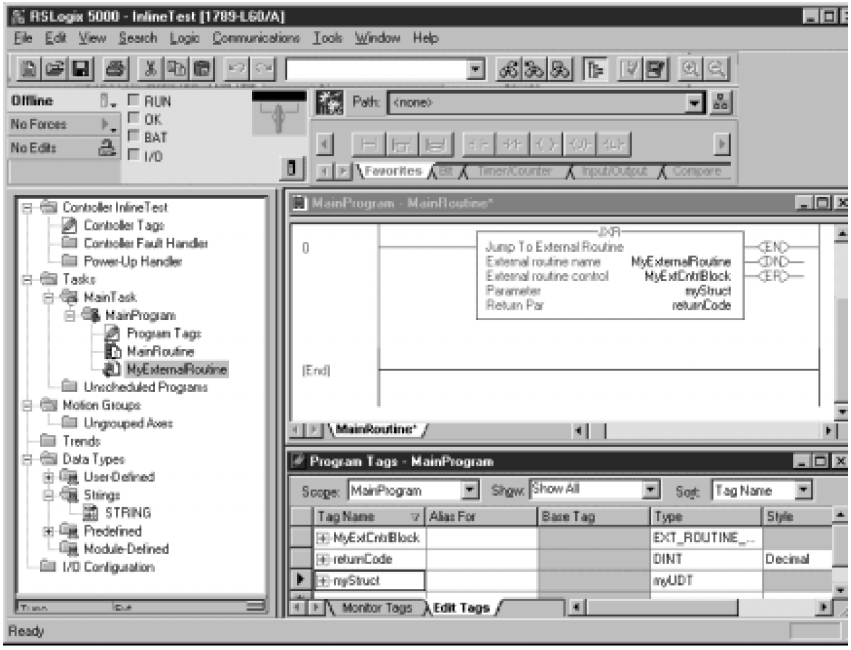
```
struct MyStruct
{
       // Structure with four integers
       int n1;
       int n2;
       int n3;
       int sum;
};

extern "C" __declspec(dllexport) int uvUDT(EXT_ROUTINE_CONTROL* pERCtrl,
                                           MyStruct* pMS)
```

XML declaration

```
<Routine>
       <EntryPoint>uvUDT</EntryPoint>
       <Description>This function accepts a pointer to a UDT</Description>
       <Signature>int uvUDT(EXT_ROUTINE_CONTROL* pERCtrl,
                            struct MyStruct* pMS)</Signature>
</Routine>
```

RSLogix 5000 declaration



STRING example

C declaration

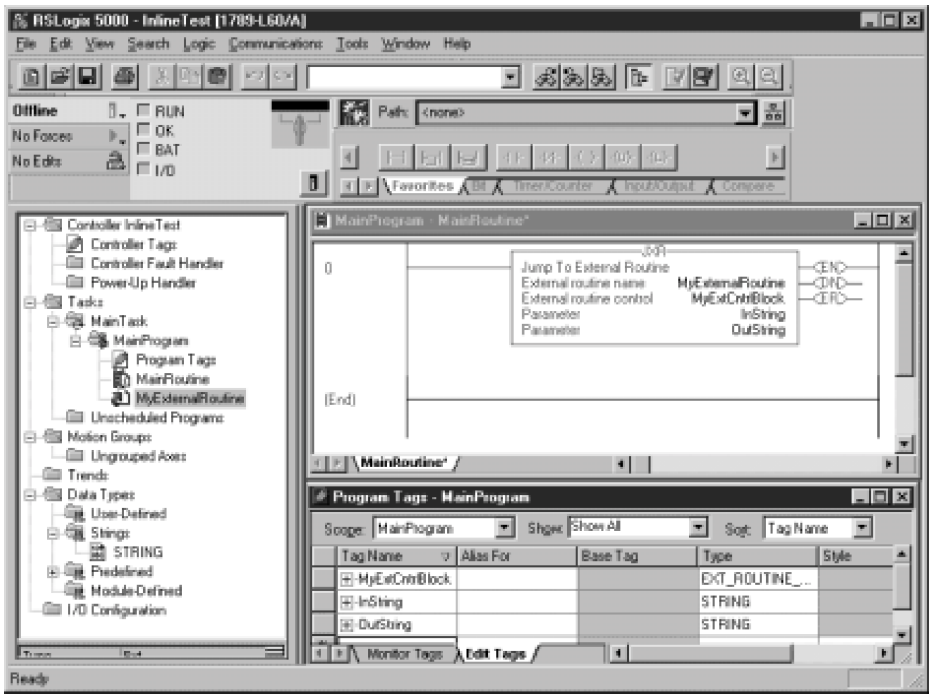
```
struct RA_String
{
       unsigned long Len;
       char  Data[82];
};

extern "C" __declspec(dllexport) void StringFunc(EXT_ROUTINE_CONTROL* pERCtrl,
                          RA_String* pInRA_String, RA_String* pOutRA_String)
```

XML declaration

```
<Routine>
       <EntryPoint>StringFunc</EntryPoint>
       <Description>This function accepts two strings. </Description>
       <Signature>void StringFunc(EXT_ROUTINE_CONTROL* pERCtrl,
                                  struct RA_String* pIn, struct RA_String* pOut)</Signature>
</Routine>
```

RSLogix 5000 declaration



Packing in structures

Take care when designing user defined structures that are shared between the SoftLogix controller and external routines. The packing mechanisms vary between the Visual Studio compiler and the RSLogix 5000 compiler. The following table and guidelines help illustrate this.

		RSLogix 5000 Structure Storage	
Member Data Type:	**Alignment:**	**Packing:**	**Storage:**
consecutive BOOLs	4 byte boundaries	bit packed across storage	((N-1 bits / 32) + 1) * 4 bytes
array <BOOL>	4 byte boundaries	bit packed across storage	((N-1 bits / 32) + 1) * 4 bytes
BOOL	1 byte boundaries	1 per byte of storage	1 byte
SINT	1 byte boundaries	1 per byte of storage	1 byte
INT	2 byte boundaries	1 per 2 bytes of storage	2 bytes
DINT	4 byte boundaries	1 per 4 bytes of storage	4 bytes
REAL	4 byte boundaries	1 per 4 bytes of storage	4 bytes
array <nonBOOL>	4 byte boundaries	1 per N bytes of storage	(eleSize * eleCount) bytes
<StructureT> (such as, UDT)	4 byte boundaries	1 per N bytes of storage	ceiling (sizeof(StructureT) / 4) * 4 bytes

- Aggregates like arrays and structures start on 4 byte boundaries.
- Consecutive BOOLs are bit packed, as are boolean arrays.
- SINTs and BOOLs are one byte aligned unless noted above.
- INT is two byte aligned.
- REAL and DINT are 4 byte aligned.
- Gaps of one or more bytes may exist between items.

The following table shows the Microsoft packing for Visual Studio version 6.0. If you are not using this version of Visual Studio, consult your documentation for appropriate packing information.

		Microsoft Win32 Structure Storage (default n=8)	
Member Data Type:	**Alignment:**	**Packing:**	**Storage:**
BOOL	min(1,n) byte boundaries	1 per byte of storage	1 byte
char	min(1,n) byte boundaries	1 per byte of storage	1 byte
short	min(2,n) byte boundaries	1 per 2 bytes of storage	2 bytes
int	min(4,n) byte boundaries	1 per 4 bytes of storage	4 bytes
long	min(4,n) byte boundaries	1 per 4 bytes of storage	4 bytes
float	min(4,n) byte boundaries	1 per 4 bytes of storage	4 bytes
array <AnyT>	min(eleAlignment,n) byte boundaries	1 per N bytes of storage	(eleSize * eleCount) bytes
<StructureT> (such as, struct)	min(largestOfMember,n) byte boundaries	1 per N bytes of storage	sizeof(StructureT) bytes

Parameter type checking

RSLogix 5000 Data Type:	Formal Parameter Enumeration:	Allowed C Language Types (XML):
literal integer value (such as 121)	IntegerValue	char, short, int, long, bool
DINT	IntegerAddress IntegerValue VoidAddress	int*, long* int, long void*
INT	IntegerAddress IntegerValue VoidAddress	short* short void*
SINT	IntegerAddress IntegerValue VoidAddress	char* char void*
literal floating point values (such as 12.345)	FloatingPointValue	float
REAL	FloatingPointAddress FloatingPointValue VoidAddress	float* float void*
BOOL	IntegerAddress IntegerValue VoidAddress	bool* bool void*
literal BOOL	IntegerValue	bool
UDT	StructureAddress VoidAddress	struct AnyStructT* s void*
arrays	ArrayAddress VoidAddress	char var[], short var[], int var[], long var[], bool var[], float var[], struct AnyStructT var[] void*

Return parameter

Only floating point and integer values can be returned from the external routine. You cannot pass a pointer to a memory location as the return parameter. This is enforced to maintain the integrity of the controller. Note that all memory which is referenced by both the controller and an external routine must have been allocated by the controller.

RSLogix 5000 Data Type:	Formal Parameter Enumeration:	Allowed C language Types (in XML):
DINT	IntegerValue	int long
INT	IntegerValue	short
SINT	IntegerValue	char
REAL	FloatingPointValue	float
BOOL	IntegerValue	bool

Exporting Functions Using C++ Export Style

If you export your external routine functions using the C++ export style, make sure that the EntryPoint value in your XML resource exactly matches the C++ decorated name that is exported by your C++ compiler. Visual Studio includes a tool (dumpbin.exe) that you can use to obtain the C++ decorated name from your DLL file. The dumpbin.exe tool is installed as part of Visual Studio product. The examples below show how to use this tool.

InlineExample.h

```
// Exported Functions:
__declspec(dllimport) int SumArray(EXT_ROUTINE_CONTROL* pERCtrl,
                                   int Val[]);
```

InlineExample.cpp

```
__declspec(dllexport) int SumArray(EXT_ROUTINE_CONTROL* pERCtrl,
                                   int Val[])
{
              // body of function.
}
```

Run dumpbin.exe

Run dumpbin.exe with the /exports flag set to display the decorated names for all of the exported routines. The following is the output running dumpbin /exports on a C++ DLL.

For example, entering this command:

```
Dumpbin.exe /exports InlineExample.dll
```

displays this output information.

```
Microsoft (R) COFF Binary File Dumper Version 6.20.8700
Copyright (C) Microsoft Corp 1992-2000. All rights reserved.

Dump of file InlineExample.dll

File Type: DLL

  Section contains the following exports for InlineExample.dll

           0 characteristics
    3BA9F7A0 time date stamp Thu Sep 20 10:05:20 2001
        0.00 version
           1 ordinal base
           1 number of functions
           1 number of names

    ordinal hint RVA      name

          1    0 0000100A ?SumArray@@YAHPAUEXT_ROUTINE_CONTROL@@QAH@Z

  Summary

        4000 .data
        1000 .idata
        2000 .rdata
        2000 .reloc
        1000 .rsrc
       28000 .text
```

Edit XML resource

Change the <EntryPoint> tag to be the decorated name (found when you ran dumpbin.exe). The XML EntryPoint name must EXACTLY match the decorated named displayed by the dumpbin.exe utility.

```
<?xml version="1.0" encoding="UTF-8"?>
<RA_ExternalRoutines_XML>
<Description>Sum array elements.</Description>
<Routines>
             <Routine>
                    <EntryPoint>?SumArray@@YAHPAUEXT_ROUTINE_CONTROL@@QAH@Z</EntryPoint>
                    <Description>Sum integer array elements</Description>
                    <Signature>int SumArray(EXT_ROUTINE_CONTROL* pERCtrl,
                                            int Val[])</Signature>
             </Routine>
</Routines>
</RA_ExternalRoutines_XML>
```

Other Considerations

Use care when passing tags by reference

You can pass tags by reference in a synchronous, single-threaded routine. You should not pass these memory addresses to another thread or process because it is possible for the originating tag to be deleted. Then, the reference to the originating tag in the thread or process becomes invalid and causes an access violation.

Using a external routine DLL that uses other DLLs

If you create an external routine DLL that uses other DLLs, make sure that the additional DLL files are accessible to the SoftLogix engine at runtime. For example, assume external routine MyER.DLL uses another file called MyAdditionalFile.DLL. RSLogix 5000 software copies MyER.DLL into the project area and downloads MyER.DLL to the appropriate controller slot during download. However, RSLogix 5000 software does not copy or download MyAdditionalFile.DLL.

To make MyAdditionalFile.DLL available on the target machine, put the MyAdditionalFile.DLL file into the Windows system32 directory on the target machine. This ensures that the file is available when needed. Otherwise, when the controller attempts to perform the Windows LoadLibrary call for MyER.DLL it will fail because the MyAdditionalFile.DLL cannot be found.

As a better solution, statically link any additional DLL files that are needed right into the external routine DLL. This leaves only one file and RSLogix 5000 software takes care of copying the DLL to the correct places.

Notes:

Programming Windows Events to Monitor and Change Controller Execution

Using This Chapter

There are different ways to programmatically use Windows events to monitor and change SoftLogix controller execution.:

If you want to:	Use:	See this page:
monitor SoftLogix events from an external routine	outbound events	10-1
configure a Windows event in the SoftLogix controller so that an application outside of the controller can initiate a task within the controller	windows events	10-5
from an external routine or application, programmatically save the current SoftLogix controller information (tag data values and controller configuration)	programmatic periodic save	10-9

Using Outbound Events

Use outbound events in asynchronous external routines to detect a change in the mode of a controller, allowing the external routine to start and stop asynchronous code appropriately.

Use the standard Windows "wait" functions to test or wait for these events. Replace the "xx" with the 2-digit slot number where the controller resides. For example, if you want to detect if the controller in slot 4 is in Run mode, check the SOFTLOGIX_04_RUN event.

Windows Event:	Description:
SOFTLOGIX_xx_STARTUP	This event is set after the controller in slot xx completes its power-up sequence. This event is reset when the controller is removed from the chassis or is shut down by the Windows operating system.
SOFTLOGIX_xx_SHUTDOWN	This event is set when the controller in slot xx is removed from the chassis or is shut down by the Windows operating system. This event is reset otherwise.
SOFTLOGIX_xx_MODE_CHANGE	This event is set whenever the controller in slot xx changes mode. This event is created as an automatic reset event.
SOFTLOGIX_xx_PROGRAM	This event is set when the controller in slot xx is in program mode. This event is reset when the controller in slot xx is not in program mode.
SOFTLOGIX_xx_RUN	This event is set when the controller in slot xx is in run mode. This event is reset when the controller in slot xx is not in run mode.
SOFTLOGIX_xx_TEST	This event is set when the controller in slot xx is in test mode. This event is reset when the controller in slot xx is not in test mode.
SOFTLOGIX_xx_FAULT	This event is set when the controller in slot xx is faulted. This event is reset when the controller in slot xx is not faulted.

Programming example: outbound events

The following example monitors a controller for mode changes and displays the controller mode whenever a change occurs.

```
/*****************************************************************************
**                                                                          **
** SOFTLOGIX 5800 OUTBOUND EVENTS EXAMPLE CODE                              **
** COPYRIGHT (c) 2003 ALLEN-BRADLEY COMPANY, L.L.C.                         **
**                                                                          **
** All rights reserved, except as specifically licensed in writing.         **
** The following work constitutes example program code and is intended      **
** merely to illustrate useful programming techniques. The user is          **
** responsible for applying the code correctly. The code is provided        **
** AS IS without warranty and is in no way guaranteed to be error-free.     **
*****************************************************************************/

/*****************************************************************************
*
* FILE: OutBoundEventExample.cpp
*
* FULL DESC:
*
* This is an example command mode application that will indicate when
* a SoftLogix5800 controller changes modes.  It uses the new Outbound
* Event support that is being introduced with version 12 of the
* SoftLogix5800 product.
*
* It should be compiled using the Microsoft Visual Studio compiler.
* You can use the following steps to compile and run this example.
*
* 1) Open a Windows command prompt window by choosing
*    Start->Run, and entering cmd for the command line.
* 2) Make sure that the Microsoft Visual Studio compiler bin
*    directory is in your path.
* 3) Execute the compiler using the following command
*    cl OutBoundEventExample.cpp
* 4) Run the resultant executable using the following command
*    OutBoundEventExample 3
*    The 3 in the example above indicates the slot number for
*    the controller for which you want to monitor mode changes.
*
* This example code waits for the controller mode change event,
* and after receiving the mode change event, checks the controller
* mode events to determine and display the new controller mode.
*
*       Copyright Allen-Bradley Company, Inc. 2003
*
*****************************************************************************/
#include <stdio.h>
#include <windows.h>
#include <tchar.h>

// Define event handle variables
HANDLE hModeChange = 0;
HANDLE hProgramMode = 0;
HANDLE hRunMode = 0;
```

```
HANDLE hTestMode = 0;
HANDLE hFaultMode = 0;

// Use strings to define the event names in which we are interested
// The user will pass in the slot number via the command line
TCHAR sa_mode_change_event_fmt[] = _T("SOFTLOGIX_%02d_MODE_CHANGE");
TCHAR sa_program_event_fmt[]     = _T("SOFTLOGIX_%02d_PROGRAM");
TCHAR sa_run_event_fmt[]         = _T("SOFTLOGIX_%02d_RUN");
TCHAR sa_test_event_fmt[]        = _T("SOFTLOGIX_%02d_TEST");
TCHAR sa_fault_event_fmt[]       = _T("SOFTLOGIX_%02d_FAULT");

int main(int argc, char *argv[])
{
    int slot = 0;
    DWORD status = 0;
    TCHAR eventname[_MAX_PATH];
    // We will keep the mode event handles in an array for the WaitForMultipleObjects call
    HANDLE EventArray[4];
    HANDLE *events = &EventArray[0];
    int numevents = 0;

    // Make sure a slot number is passed in on the command line
    argc--, argv++;
    if (argc != 1)
    {
        _tprintf(_T("You must enter a slot number on the command line.\n"));
        fflush(stdout);
        return 1;
    }

    slot = atoi(*argv);
    (void) GetLastError();

    // Create the mode change event.  Note: it must be created as auto reset.
    _stprintf(eventname, sa_mode_change_event_fmt, slot);
    hModeChange = CreateEvent (NULL, FALSE, FALSE, eventname);
    if (hModeChange == NULL)
    {
        _tprintf(_T("Bad mode change handle\n"));
        _tprintf(_T("GetLastError() = %d\n"), GetLastError());
        return 1;
    }

    // Create the program mode event.  Note: it must be created as manual reset.
    _stprintf(eventname, sa_program_event_fmt, slot);
    hProgramMode = CreateEvent (NULL, TRUE, FALSE, eventname);
    if (hProgramMode == NULL)
    {
        _tprintf(_T("Bad program mode event handle\n"));
        _tprintf(_T("GetLastError() = %d\n"), GetLastError());
        return 1;
    }
    else
    {
        // Add the program mode event to the event array
        EventArray[numevents] = hProgramMode;
        numevents++;
    }
```

```
// Create the run mode event.  Note: it must be created as manual reset.
_stprintf(eventname, sa_run_event_fmt, slot);
hRunMode = CreateEvent (NULL, TRUE, FALSE, eventname);
if (hRunMode == NULL)
{
    _tprintf(_T("Bad run mode event handle\n"));
    _tprintf(_T("GetLastError() = %d\n"), GetLastError());
    return 1;
}
else
{
    // Add the run mode event to the event array
    EventArray[numevents] = hRunMode;
    numevents++;
}

// Create the test mode event.  Note: it must be created as manual reset.
_stprintf(eventname, sa_test_event_fmt, slot);
hTestMode = CreateEvent (NULL, TRUE, FALSE, eventname);
if (hTestMode == NULL)
{
    _tprintf(_T("Bad test mode event handle\n"));
    _tprintf(_T("GetLastError() = %d\n"), GetLastError());
    return 1;
}
else
{
    // Add the test mode event to the event array
    EventArray[numevents] = hTestMode;
    numevents++;
}

// Create the fault mode event.  Note: it must be created as manual reset.
_stprintf(eventname, sa_fault_event_fmt, slot);
hFaultMode = CreateEvent (NULL, TRUE, FALSE, eventname);
if (hFaultMode == NULL)
{
    _tprintf(_T("Bad fault mode event handle\n"));
    _tprintf(_T("GetLastError() = %d\n"), GetLastError());
    return 1;
}
else
{
    // Add the fault mode event to the event array
    EventArray[numevents] = hFaultMode;
    numevents++;
}

// We are now ready to start waiting for mode changes!
_tprintf(_T("Ready\n"));
fflush(stdout);

// Loop forever, waiting for mode changes.  The user must perform a control-c to
// stop the application
while (1)
{
```

```
        // First we wait for the single mode change event.  The mode change event is an
        // automatic reset event that is set by the SoftLogix controller every time the
        // controller is changing modes.  Once the mode change event is set, we need to
        // check the state of the acutal mode events.
        status = WaitForSingleObject(hModeChange,INFINITE);
        switch (status)
        {
            case WAIT_OBJECT_0:
                _tprintf(_T("Mode change occured: "));
                fflush(stdout);
                break;
        }

        // A mode change occured.  Now we have to check to determine the new controller mode.
        // This is done by performing a WaitForMultipleObjects on the mode event handles.
        // These events are manual reset events that are controlled by the SoftLogix
        // controller.
        // Only one of these events can be set at any given time.
        // Log a message to the screen indicating the new controller mode, and then go back
        // to waiting for the mode change event above.
        status = WaitForMultipleObjects(numevents, EventArray, FALSE, INFINITE);
        switch (status)
        {
            case WAIT_OBJECT_0:
                        _tprintf(_T("now in program mode\n"));
                        fflush(stdout);
                        break;
            case WAIT_OBJECT_0+1:
                        _tprintf(_T("now in run mode\n"));
                        fflush(stdout);
                        break;
            case WAIT_OBJECT_0+2:
                        _tprintf(_T("now in test mode\n"));
                        fflush(stdout);
                        break;
            case WAIT_OBJECT_0+3:
                        _tprintf(_T("now in fault mode\n"));
                        fflush(stdout);
                        break;
        }
    }
    return 0;
}
```

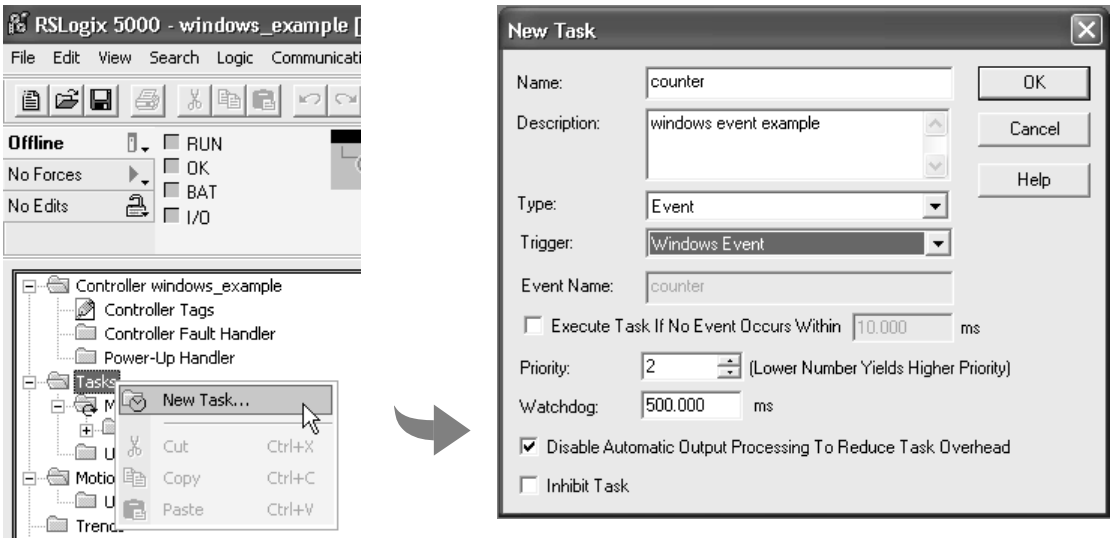
Configuring Windows Events to Launch Tasks within the SoftLogix Controller

Windows event tasks are functionality associated with Microsoft's Windows 2000 and XP operating systems. Applications outside of SoftLogix5800 (Visual Basic, RSView, Custom C applications, external routines, etc.) can cause a task within the SoftLogix5800 controller to execute.

Configuring a Windows-event task in the controller

In the SoftLogix controller, create a task and configure the trigger as a Windows event.

1. In RSLogix 5000 software, create an event task in a SoftLogix5800 controller project.



Click OK

Triggering a controller task from a Windows application

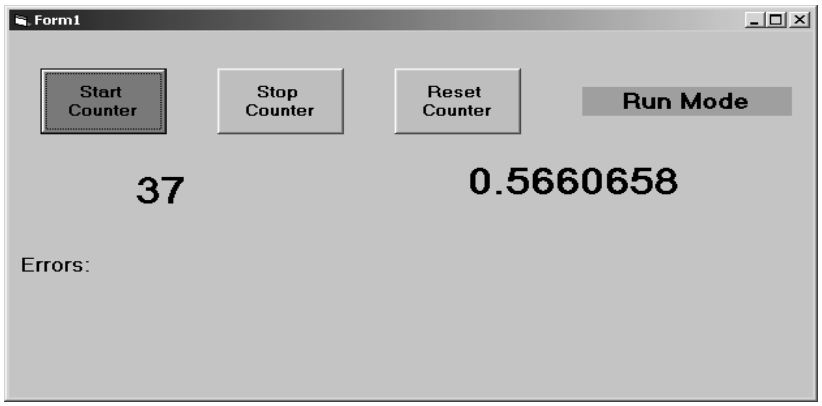
In your application, such as Visual Basic or RSView, use standard Windows API functions to open and set the Windows event in the controller.

Make sure you:

- name the Windows-event task in the controller with the same name as Windows event in the external application.
- open/create the Windows event in the external application as an “automatic reset” event, otherwise the task within the controller will execute continuously once the Windows event is set.
- use a CreateEvent call to create the Windows event; use a SetEvent call to signal the Windows event.

Programming example: Windows event

This example Visual Basic application uses Windows events to talk to a SoftLogix controller. The application passes data via a memory-mapped file using the Win32 standard library.



This example code shows how you can create the public declarations:

```
Public Declare Function CreateEvent Lib "kernel32" Alias "CreateEventA" _
(lpEventAttributes As Any, ByVal bManualReset As Long, _
ByVal bInitialState As Long, ByVal lpName As String) As Long

Public Declare Function SetEvent Lib "kernel32" (ByVal hEvent As Long) As Long
```

This example codes shows how you can use the CreateEvent and SetEvent calls. This example uses the shutdown, program, and run outbound events (see page 10-1) before executing the Windows event named “counter.” There is a corresponding Windows-event task in the controller named ”counter.”

```
Dim hCounter As Long
Dim hOutbound(3) As Long
Dim hCounter As Long

Private Sub Form_Load()
    Dim ErrorCodeEvent1 As Long

    On Error GoTo noHandle
    ErrorCodeEvent1 = 0

    hOutbound(0) = CreateEvent(ByVal 0&, 0, 0, "SOFTLOGIX_01_SHUTDOWN")
    ErrorCodeEvent1 = Err.LastDllError
    hOutbound(1) = CreateEvent(ByVal 0&, 0, 0, "SOFTLOGIX_01_PROGRAM")
    ErrorCodeEvent1 = Err.LastDllError
    hOutbound(2) = CreateEvent(ByVal 0&, 0, 0, "SOFTLOGIX_01_RUN")
    ErrorCodeEvent1 = Err.LastDllError
    hCounter = CreateEvent(ByVal 0&, 0, 0, "Counter")
    ErrorCodeEvent1 = Err.LastDllError
    If ErrorCodeEvent1 <> 183 Then '183 = Event already exists which is OK
       'handle error
    End If
```

```
noHandle:

End Sub

Private Sub Form_Unload(Cancel As Integer)
    CloseHandle (hCounter)
    CloseHandle (hOutbound(0))
    CloseHandle (hOutbound(1))
    CloseHandle (hOutbound(2))
End Sub

Private Sub pbCounter_Click(Index As Integer)
    Dim ErrorCodeEvent1 As Long

    SetEvent (hCounter)
    ErrorCodeEvent1 = Err.LastDllError
    If ErrorCodeEvent1 Then
       'Handle error
    End If
End Sub


Private Sub Timer1_Timer()
Dim inMode As Long
Dim ErrorDescription As String

    inMode = WaitForMultipleObjects(3&, hOutbound(0), False, 0)
    Select Case inMode
        Case WAIT_OBJECT_0 + 2
            Label3.BackColor = &HFF00&
            Label3.Caption = "Run Mode"
        Case WAIT_OBJECT_0 + 1
            Label3.BackColor = &HFF8080
            Label3.Caption = "Program Mode"
        Case WAIT_OBJECT_0
            Label3.BackColor = &HFF00FF
            Label3.Caption = "SHUTDOWN Mode"
        Case Else
            Label3.BackColor = &HFFFFFF
            Label3.Caption = "Other Mode"
    End Select

End Sub
```

Programmatically Saving the Controller

From an external routine or application, you can programmatically save the current controller information (tag data values and configuration information).

A programmatic save can use these pre-defined Windows events. Replace the "xx" with the 2-digit slot number where the controller resides.

Windows Event:	Description:
SOFTLOGIX_xx_KICKSAVE	This incoming event indicates that a save is to be forced in the controller in slot xx.
SOFTLOGIX_xx_SAVEPERMITTED	This outbound event indicates that a save is permitted in the controller in slot xx.
SOFTLOGIX_xx_SAVESTART	This outbound event indicates that a save has started in the controller in slot xx.
SOFTLOGIX_xx_SAVEDONE	This outbound event indicates that a save has completed in the controller in slot xx.
SOFTLOGIX_xx_SAVELOCK	This event is a handshake between multiple clients. Use this event to prevent multiple clients from attempting to start a save at the same time in the controller in slot xx.

Programming example: programmatic save of controller

The following example codes shows how you can use all of the above events to programmatically save the contents of a controller. This code uses printf statements to help track progress through the application.

```
int savenow(int slot)
{
        DWORD status;
        WCHAR eventname[_MAX_PATH];
        SECURITY_ATTRIBUTES sa;
        PSECURITY_DESCRIPTOR pSD;

        // Create a NULL DACL to allow other tasks to access the external events.
        pSD = (PSECURITY_DESCRIPTOR) LocalAlloc(LPTR, SECURITY_DESCRIPTOR_MIN_LENGTH);
        if (pSD == NULL)
            return 1;

        if (!InitializeSecurityDescriptor(pSD, SECURITY_DESCRIPTOR_REVISION))
            return 1;

        // Add a NULL DACL for full permission for all
        if (!SetSecurityDescriptorDacl(pSD, TRUE, (PACL) NULL, FALSE))
            return 1;
        sa.nLength = sizeof(sa);
        sa.lpSecurityDescriptor = pSD;
        sa.bInheritHandle = TRUE;

        // CreateEvent(security, manuallyReset, initialState, eventName);
        swprintf(eventname, sa_kicksave_event_fmt, slot);
        h_kicksaveEvent = CreateEvent (&sa, FALSE, FALSE, eventname);
        printf("h_kicksaveEvent = %d
```

```
        swprintf(eventname, sa_savepermitted_event_fmt, slot);
        h_savePermittedEvent = CreateEvent (&sa, TRUE, FALSE, eventname);
        printf("h_savePermittedEvent = %d

        swprintf(eventname, sa_savestart_event_fmt, slot);
        h_saveStartEvent = CreateEvent (&sa, TRUE, FALSE, eventname);
        printf("h_saveStartEvent = %d

        swprintf(eventname, sa_savedone_event_fmt, slot);
        h_saveDoneEvent = CreateEvent (&sa, TRUE, FALSE, eventname);
        printf("h_saveDoneEvent = %d

        swprintf(eventname, sa_savelock_event_fmt, slot);
        h_saveLockEvent = CreateEvent (&sa, TRUE, FALSE, eventname);
        printf("h_saveLockEvent = %d

              // Yield remainder of timeslice to make it less likely that a
              // context switch will occur between the wait for lock event and
              // corresponding SetEvent().
              // If saves not permitted (download in progress) OR
              // a save is in progress
              //   Explain
              // Else
              //   Set lock event
              //   Kick save event
              // Endif
              // Wait for save started (if desired)
              // Wait for save done (if desired)
              SwitchToThread();
              if ((WaitForSingleObject(h_savePermittedEvent, 0) != WAIT_OBJECT_0) ||
              WaitForSingleObject(h_saveLockEvent, 0) == WAIT_OBJECT_0)
              {
                           printf("Save not permitted or save already running

                           return 1;
              }
              else
              {
                           SetEvent(h_saveLockEvent); // Save in progress
                           printf("Kicking save

                           SetEvent(h_kicksaveEvent);
              }

              WaitForSingleObject(h_saveStartEvent, INFINITE);
              printf("Save started

              WaitForSingleObject(h_saveDoneEvent, INFINITE);
              printf("Save complete


              return 0;
}
```

Windows Considerations

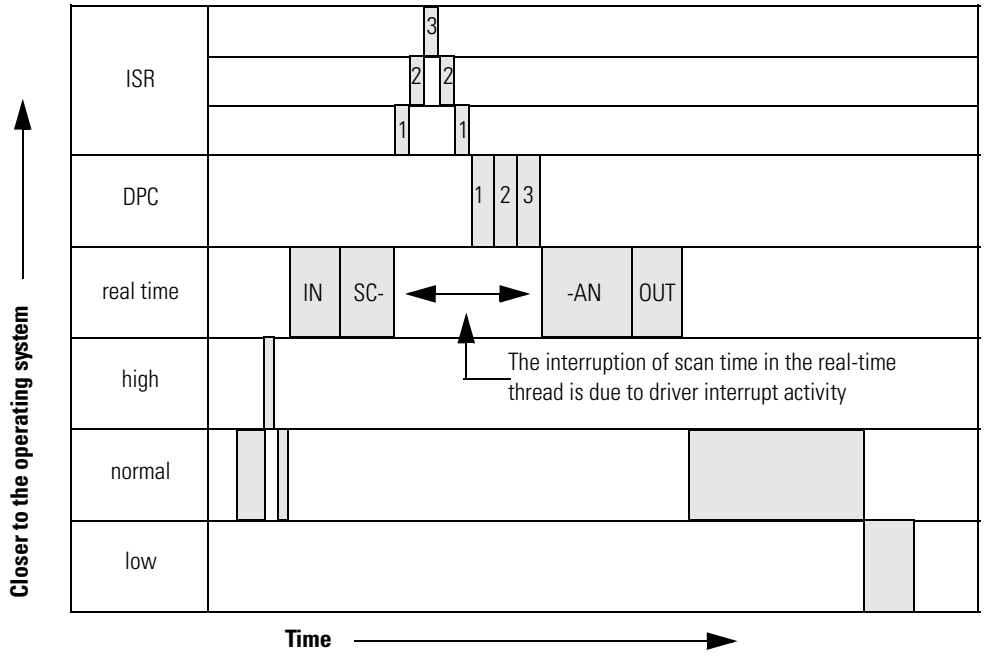
Using This Appendix

Windows Objects

There are three objects that execute within Windows that get CPU resources based on Window's multitasking/multithreading algorithms.

Windows Object	Description:
interrupt service routine (ISR)	An interrupt service routine is a software routine that is primarily executed in response to hardware and software interrupts. ISRs always execute immediately and run in the kernel-mode layer of Windows. Each ISR executes at one of 32 levels. The hardware interrupts that occur in a computer get mapped to 16 of the 32 possible levels. The important point about ISRs is that they are written by the vendors of hardware and software products and not Microsoft. Microsoft can recommend how to write ISRs, but there is no guarantee about the how well other vendors write them. For example, once an ISR starts to execute, it can raise its level to a higher priority so that the Windows scheduler won't swap it out, or it can even make a function call to mask all other interrupts until it is finished. Due to the variations in code writing, ISRs can cause swings in system responsiveness and determinism in a soft controller.
deferred procedure call (DPC)	A deferred procedure call is a software routine that is queued by an ISR to perform less time-critical processing. DPCs also execute in the kernel-mode layer of Windows, and therefore can prevent user mode applications from running. DPCs do not have priority levels, but execute in a FIFO (first-in-first-out) order as queued by their associated ISRs. Poorly written DPCs could unnecessarily keep the SoftLogix controller from running. Disk drivers, network drivers, and video drivers can be major sources of long DPCs, causing variation in SoftLogix performance. Do not use CD writing (recording) devices and fancy screen savers on the SoftLogix controller because they have shown significant jumps in CPU utilization.
dispatched threads of execution	Threads are the primary execution pieces of software that Windows switches between. Some are associated with larger application programs and background processes, while others are created by the kernel and device drivers. Threads are individual code segments spawned from processes that can be in one of four priority levels; low, medium, high and real-time. Threads that are spawned from a real-time process, like the SoftLogix controller, execute to the point of blocking, yielding, or completing. The dwell component of the SoftLogix controller allows the continuous task of the controller to give time to other lower priority threads that need to execute.

The following diagram shows the relationship between these objects and shows how one object has to stop running if another with higher priority wants to execute. The SoftLogix controller executes as a real-time priority process, and thus waits for all ISRs and DPCs to complete before executing.



Other Considerations

Consideration:	Description:
Multiple CPUs in one computer	Multiple CPUs in the PC can greatly improve performance. The Windows scheduler algorithm automatically uses both CPUs to execute whatever needs to be executed. Any code that needs to execute will move to whichever CPU is available, unless the current process specifically requests a certain CPU. The CPU Affinity parameter of the SoftLogix controller has you specify a which CPU to execute on, allowing you to customize your system for more determinism.
Blue screen events	Blue screen events are the result of the Pentium processor generating self-diagnostic events that fall in the category of fault or abort. Usually there is code to recover from fault interrupts, such as Page Faults, and these do not cause Windows to stop. But there are occasions in the case of hardware failures that generate a NMI (non-maskable interrupt) or a parity error that are considered faults that cannot be ignored and therefore Windows does the proper thing and immediately stops. This is similar to a PLC-5 processor 'red lighting' when it detects an internal memory or hardware error. Blue screens that occur during system integration of new third-party hardware indicate a poorly written driver that is corrupting Windows kernel memory. After a blue screen event, the I/O modules controlled by a SoftLogix controller go to their fail safe modes of operation (as specified when the I/O was configured). A motion card controlled by the SoftLogix controller sees a blue screen event as a communication fault and takes shut down action (motion stops).

Consideration:	Description:
Microsoft service pack	Microsoft service pack is the name Microsoft gives to an operating system upgrade. It is always recommended to apply the latest service pack after installing third party software, especially networking drivers and the addition of network protocols. Whenever you receive errors that seem low-level or don't make sense, reapply the latest Microsoft service pack and reboot.
Bus mastering	The hard drive must support bus mastering. You might also need bus-mastering drivers for the PC chip set. For example, Intel motherboards call this software "Application Accelerator." If you have to restore the operating system to the hard drive, the bus mastering software which might have been pre-loaded at the factory, might not get restored during the recovery process and you might have to manually install the software yourself.
Third party peripheral devices	Third party peripheral devices, such as network cards and IDE devices should be installed directly in the computer's primary PCI bus. IDE devices should use PCI bus mastering. Bus mastering is the capability of writing directly to the computer's memory without having to use the Pentium chip to move the data. You can use Microsoft's DMACHECK.EXE utility to see whether a third-party card truly uses PCI bus mastering. You might also need to turn bus mastering on within the BIOS setup of your computer. See the documentation for your computer for more information about bus mastering.
TestTime utility	The SoftLogix controller ships with a TestTime utility that you can use to determine the responsiveness of your system to CPU interrupts. The utility measures how long it takes to respond to a software interrupt that is generated every 2 ms. It measures the average, max, number of occurrences and standard deviation of how quickly your PC responds. If you find significant delays, focus on any peripheral devices on the computer and their associated drivers. The best way to use the utility is to run it on a new system with no software or third-party hardware installed to get a baseline measurement. Then rerun the TestTime utility each time you install a piece of hardware or a software package to determine if there is a problem.
Screen savers	Disable screen savers on your computer. OpenGL screen savers have been known to generate excessive loading on a PC. This can cause fluctuations in SoftLogix control because video-driver operations interrupt the SoftLogix real-time execution priority under Windows. OpenGL is a standard for generating 2-dimensional and 3-dimensional graphics in current screen savers and animated games.

Running a SoftLogix Controller on Windows

The SoftLogix controller executes as a service (background program) that starts when Windows boots and then runs at real-time priority within Windows. Most other applications, such as word processors and spreadsheets, run at normal priority. Because SoftLogix runs at a real-time level, it is guaranteed to get as many CPU cycles as it needs before allowing the CPU to execute other application programs. Only DPCs and ISRs run before a SoftLogix controller.

Selecting a dwell time setting

Every SoftLogix controller has a main task that can be configured to run continuously or periodically. If set for continuous, the main task would use all of the Windows CPU cycles, if it were able, running as a real-time priority process. But the dwell time configuration of the SoftLogix controller is a value in milliseconds which is directly added to the end of every scan of a continuous program task. The dwell time is a period of time that counts off in real-time after the SoftLogix controller's continuous task. This time is like a sleep time for the SoftLogix controller so that Windows can execute lower priority threads.

If a SoftLogix controller's periodic task is set to run, it runs during the dwell time, but the time spent executing the periodic task is not added to the dwell time. The dwell time counts in the background in real-time and the end of the dwell marks the continuous task as ready to run. The continuous task will run as long as no other periodic tasks are already executing or are ready to execute.

A dwell time of 0 ms does leave some dynamic amount of time of dwell that is less than 1 ms to prevent you from completely using all CPU cycles, and thus locking up your computer. When the continuous task enters the dwell time, it makes a function call to Windows to "SwitchToThread," which is a function that lets the next thread that needs to run go ahead and execute.

If multiple SoftLogix controllers in the same virtual chassis are set for a dwell time of 0 ms, the controllers will starve other applications that are running at normal priority. The effect is sluggish mouse control and slow response time by other Windows applications. And if you run this configuration on a slower computer, you may even lock yourself out of being able to do anything in Windows.

IMPORTANT It is possible to lock yourself out of your computer if you have multiple controllers installed in the virtual chassis and

- each controller is set for a dwell of 0
- periodic tasks are set for very low settings (short time periods)

In this state, the keyboard and mouse are not recognized by Windows because Windows is spending all of its time executing the real-time tasks of the SoftLogix controllers. If the controllers are set to start in "last state", you will never be able to move the mouse to put them in Program mode to free up CPU resources.

It is recommended that during development, set the controller to start in the Remote Program mode. This way, if you ever have controllers in Run mode and the PC locks up, you can cycle power and have the controllers come up in Program mode, giving you enough CPU time to make changes to your application to correct the problem. Then after development is complete, you can change the startup mode to start in "last state".

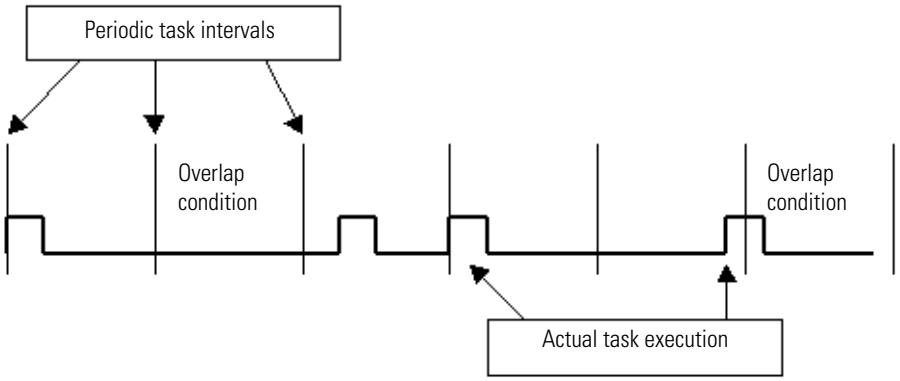
There is no window in RSLogix 5000 programming software that shows overall scan time including the dwell time component. The scan times reported in each task are values that indicate the time to scan a particular program and do not include dwell time. Use the Task Manager's Performance Monitor to gauge the effect of dwell time settings.

Using periodic tasks

Periodic tasks always attempt to execute according to their setting, and they always interrupt the continuous task. If the controller is running it's dwell time, a periodic task still interrupts the dwell time to run. If two periodic tasks attempt to run at the same time, the task that has the higher priority executes first. Be careful not to execute too many periodic tasks with short intervals as you can start to use all the bandwidth of the computer without leaving CPU cycles to operate the mouse and keyboard.

A periodic task pauses if an ISR or DPC routine needs to be executed by Windows, and then the periodic task continues when the interrupt is complete. The periodic task executes again in real time at the next preset interval. The time spent in the ISR or DPC does not get added to the time counted between periodic tasks.

A periodic task detects an overlap and sets the Overlap fault bit in the controller if a periodic tasks fails to run at all during its assigned time slot or if a periodic tasks starts later than scheduled and cannot complete before the start of the next period. The following diagram shows periodic task intervals, when a task actually starts, and what is considered an overlap condition.

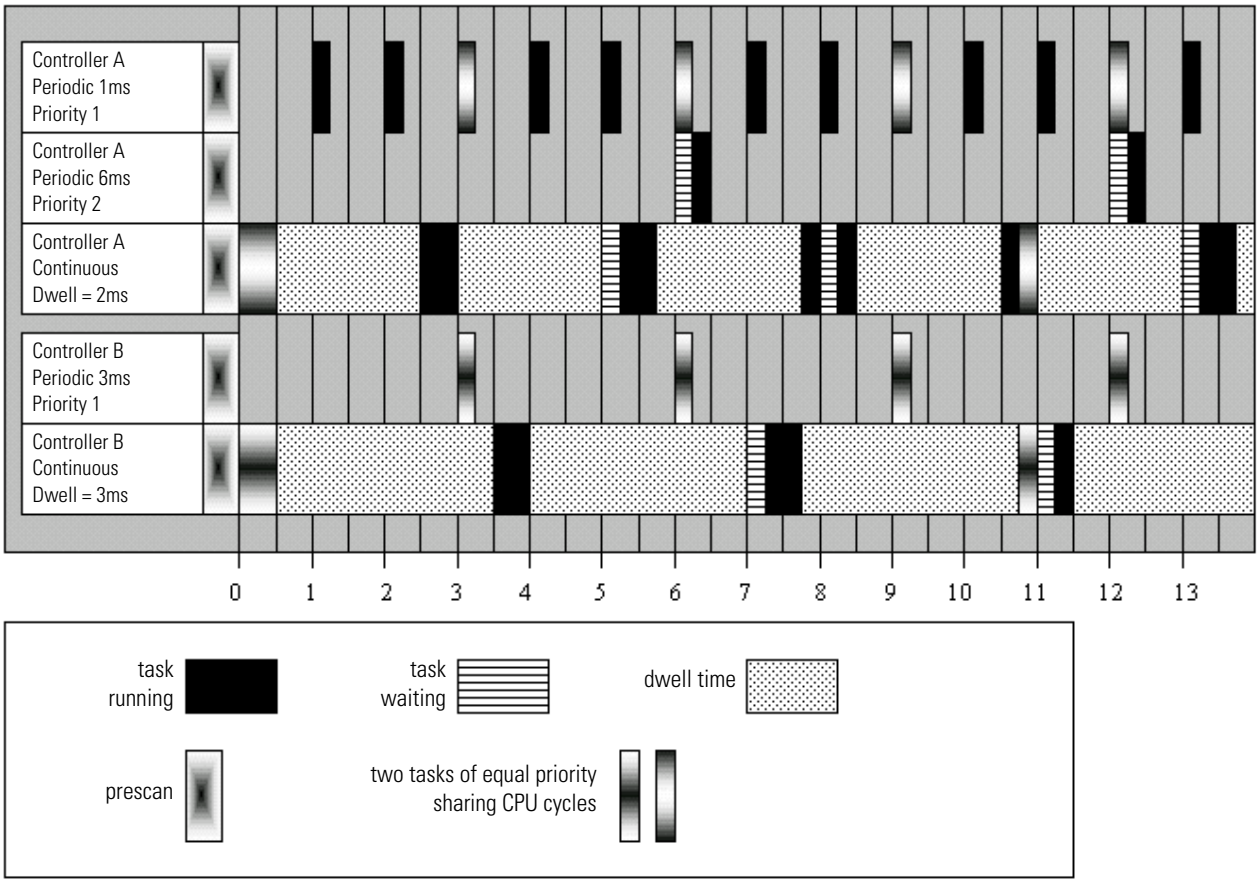


If two controllers in the same virtual chassis each have high priority periodic tasks and the tasks become active at the same time, Windows tries to switch between the tasks at whatever quantam is set within Windows. The quantam varies based on the performance boost setting for the process. With no performance boost, the quantam is 20 ms for the Windows workstation. Typically a SoftLogix controller finishes the entire scan of a periodic task before using a whole quantam.

To use the whole quantam, a thread has to be ready to execute the whole time. If a thread stops and makes any type of I/O call, (such as disk drive, DRAM memory, etc.), the thread gets switched by Windows and the CPU executes the next thread that is ready to run. This applies to the SoftLogix controller because the controller references different tags in a program scan, which are DRAM I/O operations. Therefore, Windows switches back and forth many times between two periodic tasks that are executing at the same time and at the same priority level, with the switching happening in the microsecond range.

The following diagram shows the timing of task execution between two SoftLogix controllers in the same virtual chassis. Each controller has periodic tasks and a continuous task. The example periodic tasks are short and only take 0.25 ms to execute. The example continuous tasks take 0.5 ms to execute. Anytime two periodic tasks need to execute at the same time and they each have the same priority, they share CPU cycles as Windows constantly switches between.

The beginning of the diagram shows what happens when the controller goes from Program mode to Run mode, which involves a prescan of all tasks. Then Run mode begins. The real time starts counting as shown at the bottom of the diagram.



Selecting the system overhead time slice

All Logix-based controllers have a configuration setting for the system overhead time slice. This function lets the controller take care of communication requests that occur from other controllers or from queued requests from within the controller's application program. The time slice switches the priority level of the continuous task with that of the background communication task, which is always running at a lower level than the continuous task.

The time slice setting is a percent value that is applied to a 100 ms background timing window. With a setting of 10% (the default), for every 100 ms of real time, there is 10 ms of time when the communications task priority is higher than the continuous task. If there is communication activity to perform, the controller does it and when completed, the controller lets the continuous task run again during that 10 ms window. For the next 90 ms, the continuous task is at its normal priority and the communications task is lower. During dwell time, if there are communication tasks ready to run, they will run during the dwell even though the communication task is not switched to a higher priority. And any periodic task that needs to run overrides both the continuous task and the communications task.

Using multiple SoftLogix controllers

Multiple controllers in the virtual chassis, executing on a computer with only one CPU, is less efficient than one controller. With multiple controllers, Windows has to take time to swap threads in a round-robin fashion, assuming all the controllers have a continuous task and a very small dwell. If your computer has multiple CPUs, then assign multiple controllers across the multiple CPUs.

PC Hardware Considerations

The PC hardware you chose for the SoftLogix controller will have a dramatic impact on the performance of the SoftLogix control system. The recommended platform at the time of release of the SoftLogix controller is a Pentium III (450 Mhz minimum) processor with 128 Mbytes of RAM. As mentioned earlier, a dual CPU system will provide greater stability in ladder scan time and motion updates. And if a graphical MMI package is running on the same computer, 256 Mbytes of RAM is recommended.

Other considerations include:

Consideration:	Description:
hard drives	The hard drive should be capable of bus mastering in order to reduce loading on the Pentium processor. Bus mastering allows the hard drive to initiate data transfers without using Pentium CPU cycles. To accomplish this the PC must have a motherboard that supports this technology as well as a BIOS that supports it. Then the drive itself should be capable of bus mastering. Most PC vendors will fully integrate for you this IDE bus mastering capability.
CD-ROM drives	Verify that the hard drive for your PC is on a designated IDE channel and that the CD-ROM drive is on another (secondary). Some PC vendors attempt to put the CD -ROM as slave off of the primary IDE channel and this causes performance problems for the hard drive.

Consideration:	Description:
redundant array of disks (RAID)	This technology uses multiple hard drives in a PC, so that any one hard drive can fail without causing Windows to crash. There are 5 different versions of RAID, each with its own method of error correction and recovery. The SoftLogix controller supports the RAID environment, which is recommended for critical applications that can't afford a crash. Sensitivity to hard drive crashes is common among PC users, but over the last 5 years, the reliability of hard drives has greatly increased. RAID technology is expensive and can be hard to implement and support. A more inexpensive option is to have another hard drive with a copy of the original hard drive image available. You can even mount the duplicate in the same PC without power or IDE connections, so that it is ready to connect if the original hard drive ever fails.
uninterruptable power supplies (UPS)	Uninterruptable power is an excellent accessory for a SoftLogix system as it prevents disruptions to the SoftLogix controller due to brown outs and power outages, which are the most common interferences to PCs. There are many UPS systems available, including some with digital outputs that can be interfaced back into the SoftLogix controller using discrete inputs so that the controller can detect a power outage and prepare for an orderly shutdown after a designated amount of time.

Notes:

System Performance Tuning Guidelines

Introduction

The amount of PC processing power required for your application depends on your control application configuration, e.g. number of tasks, periodic rate settings, number of data tags, and whether other windows applications are running. This appendix describes the effect of these items and provides some guidelines on how to maximize system performance.

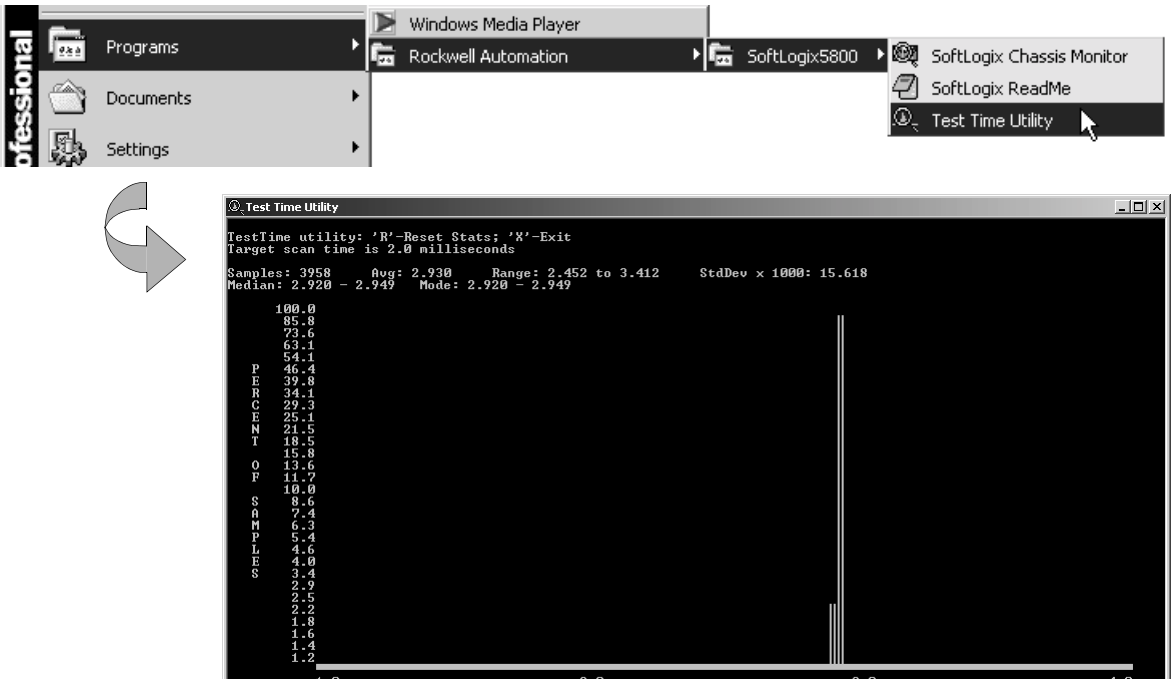
The CPU speed of the PC and whether the system is a single or dual Pentium are also important. The developer of the soft control system must partition and scale the application correctly for the capabilities of the PC.

Pre-Qualifying your PC for Soft Control

After installing SoftLogix5800 on your PC, verify that the system performance is appropriate for soft control. This is not usually an issue on PC's pre-configured and supplied by major vendors.

Run the Test Time utility that is installed with SoftLogix5800. Let this application run for a period of time while you perform tasks that you normally run on the computer. When minimized, the utility monitors your system in the background.

When you open the window, the utility displays this information:



The Test Time utility monitors your system's responsiveness to a repetitive 2 ms timer. It displays both a textual and a graphical display of the timing response of your computer. If your system is set up and operating properly the maximum value in the range should be no greater than 3 or 4 ms. If the value is significantly higher, your system is not appropriate for running SoftLogix and performing machine control.

Some reasons why your system might not behave as expected are:

- The system does not have bus mastering enabled. For information on bus mastering, see the PC hardware considerations in Appendix A.
- The system has an "ill-behaved" device driver that violates the Microsoft Windows Hardware Quality Lab (WHQL) guidelines. Verify that you have the correct vendor-specific device driver loaded for your video, Ethernet, sound, SCSI, IDE, etc. devices. These drivers should be "signed" by Microsoft, or if provided by another vendor, the driver should be certified by the Microsoft Windows Hardware Quality lab. Most vendors will indicate driver that have been WHQL certified on their driver download web page.

- Your system has an ISA Ethernet or other expansion card that is not bus-mastering capable. You can verify this by removing the ISA card(s) and running Test Time again.

- There is another real-time priority application running on the PC. Typically, the only commercially available software that can cause this type of problem is CD writer software.

- Check that your PC BIOS is compatible with the operating system. Windows 2000 and Windows XP power management features and other aspects of the operating system require support in the PC vendor's supplied BIOS. Check your PC vendor's web site for updated BIOS versions and for details on which operating systems are supported.

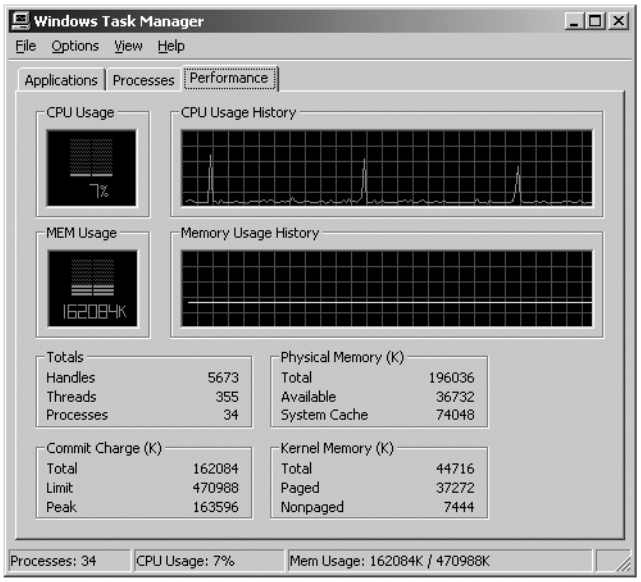
- Your system is running Windows 2000 or Windows XP and is subject to the "System Performance Counter Unexpectedly Leaps forward problem" as described in the Microsoft Knowledgebase. Many systems will not demonstrate this problem unless actually running the SoftLogix5800 controller. This problem causes unexpected watchdog timeouts and I/O connection timeouts. There is no feasible workaround on these systems. The alternatives are to run Windows NT 4.0 or upgrade your PC such that it does not contain the affected South Bridge chipset.

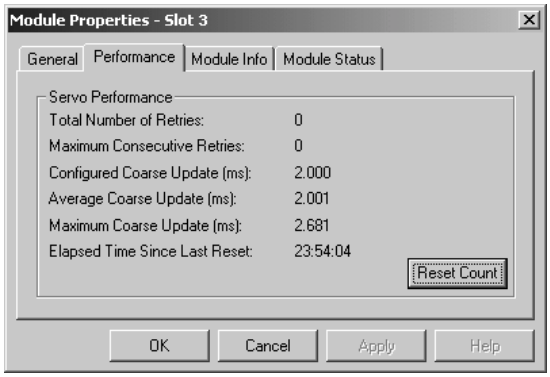
- You are trying to run SoftLogix5800 on a platform that does not meet the minimum system requirements of a Pentium 4 1.6 GHz or greater.

- There is not enough RAM or disk space. Sufficient RAM keeps memory swapping by the operating system to a minimum. Ample disk space and a defragmented drive prevent excessive hard disk accesses.

Tuning the System Performance with SoftLogix5800

There are a number of configurable parameters in the SoftLogix5800 controller that affect the overall system performance of your PC. These parameters are:

SoftLogix Parameter:	Description:
continuous task dwell time	By default this value is 10 ms. Your application should be structured such that the continuous task contains programs that are not time critical so that you can adjust the dwell time to a value as large as possible. For time critical tasks use Periodic type tasks and be prudent in the number of periodic tasks and the configured period.
periodic save interval	This parameter should also be set as large as possible to a value that is appropriate for your application. Every time the periodic save runs, SoftLogix high priority tasks consume CPU time to perform the data tag value save to hard disk. This setting does not affect the storing of online edits to the hard disk. Any edits are immediately saved when they are entered. This is true even when the periodic save is disabled. Only data tag values are saved when the periodic save executes. You can also observe the execution of the periodic save in the Windows Task Manager. It is possible that if you set the time interval short and have a large number of tags that the periodic save runs continuously. This in itself is not a problem, however as a goal, keep the overall CPU use below 80% on a continuous basis. This leaves headroom for other Windows applications to run properly. If your system cannot be tuned to achieve this, then using a dual Pentium PC or upgrading to a faster Pentium processor is recommended.  The spikes show when the periodic save is running.
CPU affinity	Use the CPU affinity setting on a dual Pentium processor PC to balance the overall system performance. You can: • run control on one CPU and leave the second CPU for other Windows applications. • install two SoftLogix controllers in the virtual chassis and set the affinity of each to a different CPU. For the best overall performance, especially when using integrated motion, use one SoftLogix controller in the virtual chassis on a single CPU system.

SoftLogix Parameter:	Description:
motion	Use the Module Properties performance dialog in the SoftLogix5800 Chassis Monitor to verify whether the controller is reliably exchanging data with a motion card at the scheduled coarse update interval. The number of retires should be zero, which indicates that motion coarse updates are occurring at the scheduled interval.  If you see excessive retries, follow the recommendations outlined in the Pre-Qualifying your PC for Soft Control section above. **Note:** • The Total Number of Retries field indicates the number of times that the controller did not communicate with the motion card at the scheduled coarse update interval since the motion counters were reset. • The Maximum Consecutive Retries field shows the highest number of consecutive coarse updates that were not completed successfully since the motion counters were reset. • The Configured Coarse Update field indicates the programmed coarse update interval. • The Average Coarse Update field indicates the average, actual coarse update interval since the motion counters were reset. • The Maximum Coarse Update field indicates the maximum length of an actual coarse update interval. It is typical and acceptable for this value to be up to 0.8 milliseconds greater than the programmed coarse update value. When the value of the Maximum Coarse Update field approaches two times the Configured Coarse Update value, retries are likely to occur. **Important:** Communication retries between a motion module and the SoftLogix controller can affect motion performance. Delays in the coarse rate exchange of data can cause increased errors in axis position or velocity. Consecutive retries (whose sum approach 70 ms) result in a ModuleSyncFault.
EtherNet/IP connections	The total number of connections from an EtherNet/IP port in a SoftLogix controller depends on the performance of the computer running the controller. As you increase the number of connections, the performance of the computer decreases.

System Startup

There are much greater CPU demands on the PC during system boot up. This is due to the fact that the SoftLogix controller restores its application program so that it can restart in the same state as it was shutdown. This restore is done at a high priority level so that the time from “power button press” to running the control application is as short as possible. While the restore is occurring, other Windows applications are also putting demands on the PC system during their startup phase.

If the system CPU load is too great during startup, Windows displays the Server Busy dialog box. If this condition occurs for other applications, such as SQL server, Microsoft Internet Information Server, Virus scanners, or disk defraggers, you should delay starting other Rockwell applications, such as RSSQL and RSView, until after the controller has completed its restore process and begins executing its application.

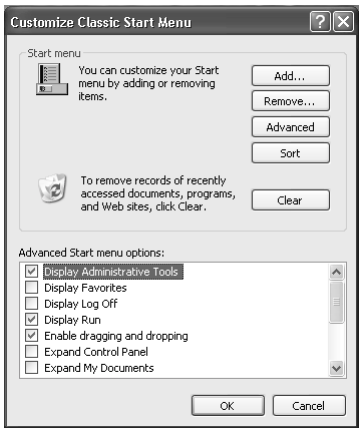
A SoftLogix controller with a typical application on a contemporary PC takes about one to two minutes to complete power-up and restore of the application. See A2048/A9662 TechNote on how to delay the startup of other applications on the PC.

Monitoring PC Performance

Standard installations of the Windows XP operating system include a PC performance utility that is useful for monitoring send and receive parameters. This utility is available as part of Administrative Tools.

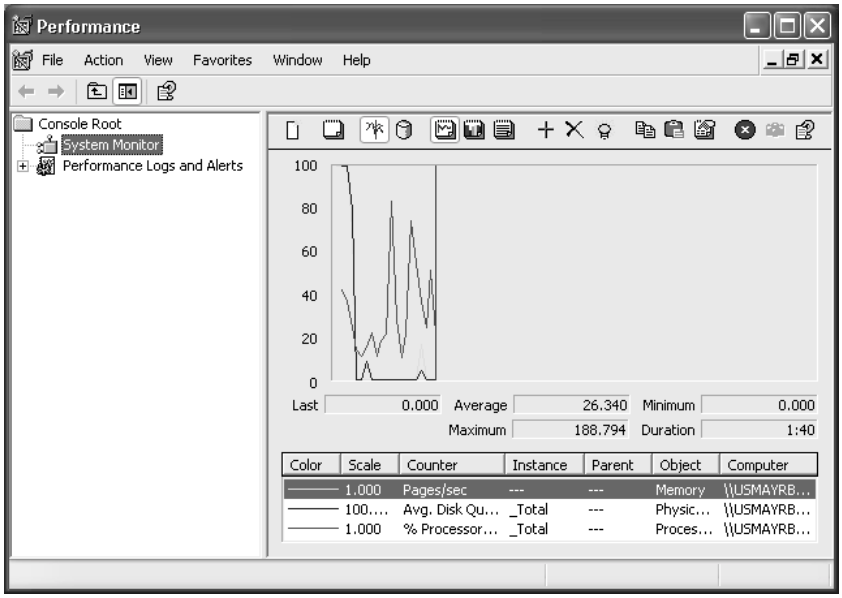
You might have to customize the Start menu to display the Administrative Tools option.

1. Right-click on the Start menu and select Properties. From the Properties menu, customize the Start menu to display the Administrative Tools option.



Click OK and then click Apply

2. From the Start menu, select Programs → Administrative Tools → Performance.



Notes:

Monitoring Controller LEDs

Indicator:	Color:	Description:
RUN	off	The controller is in Program or Test mode.
	solid green	The controller is in Run mode.
I/O	off	Either: • There are *no* devices in the I/O configuration of the controller. • The controller does *not* contain a project (controller memory is empty).
	solid green	The controller is communicating with all the devices in its I/O configuration.
	flashing green	One or more devices in the I/O configuration of the controller are *not* responding.
	flashing red	A virtual chassis error was detected. Contact your Rockwell Automation representative or local distributor.
FRC	off	No tags contain I/O force values. I/O forces are inactive (disabled).
	flashing green	At least one tag contains an I/O force value. I/O force values are inactive (disabled).
	solid green	I/O forces are active (enabled). I/O force values may or may not exist.
RS232[1]	off	No COM port was selected.
	solid green	The selected COM port was successfully assigned to channel 0 of the controller.
	solid red	There is a COM port conflict or you selected an invalid COM port number.
BAT[1]	off	Normal operation.
	flashing amber	The controller is in power-up mode.
	solid red	Persistent storage for the controller has failed.
OK	flashing red	**If the controller is:** / **Then:** a new controller / the controller requires a firmware update not a new controller / A major fault occurred. To clear the fault, either: - Turn the keyswitch from PROG to RUN to PROG - Go online with RSLogix 5000 software
	solid red	The controller detected a non-recoverable fault, so it cleared the project from memory. To recover: 1. Cycle power to the chassis. 2. Download the project. 3. Change to Run mode. If the OK LED remains solid red, contact your Rockwell Automation representative or local distributor.
	solid green	The controller is OK.

(1) Note that these LEDs function slightly different than the same LEDs on a ControlLogix controller.

SoftLogix EtherNet/IP Module LEDs

Link Status (LINK) indicator

Condition:	Status:	Indicates:	Recommended Action:
off	no link	The module is not connected to a powered Ethernet device. The module cannot communicate on Ethernet.	Verify that all Ethernet cables are connected. Verify that the Ethernet switch is powered.
flashing green	data transmission	The module is communicating on Ethernet.	Normal operation. No action required.
solid green	link OK	The module is connected to a powered Ethernet device. The module can communicate on Ethernet.	Normal operation. No action required.

Network Status (NET) indicator

Condition:	Status:	Indicates:	Recommended Action:
solid green	CIP connections established	The module has an IP address and CIP connections (Class 1 or Class 3) are established.	Normal operation. No action required.
flashing green	no CIP connections established	The module has an IP address, but no CIP connections are established.	Normal operation if no connections are configured. No action required. If connections are configured, check connection originator for connection error code.
flashing red	Lost network connection	The module detected that the network connection has been lost.	Verify that all Ethernet cables are connected. Verify that the Ethernet switch is powered.

Module Status (OK) indicator

Condition:	Status:	Indicates:	Recommended Action:
solid green	OK	The module is operating correctly.	Normal operation. No action required.
flashing green	standby	The module is not configured correctly.	Verify the module's configuration.
solid red	major fault	An unrecoverable fault has occurred.	Cycle power to the controller. Correct the fault.
flashing red	minor fault	A recoverable fault has occurred.	Correct the fault.

Numerics

A

B

C

D

E

O

P

R

S

T

U

V

W

X

Notes:

How Are We Doing?

Your comments on our technical publications will help us serve you better in the future. Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at www.ab.com/manuals, or email us at RADocumentComments@ra.rockwell.com

Pub. Title/Type SoftLogix5800 System User Manual

Cat. No. 1789-L10, -L30, -L60 Pub. No. 1789-UM002F-EN-P Pub. Date March 2004 Part No. 957867-11

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness 1 2 3	How can we make this publication more useful for you?	
Completeness (all necessary information is provided) 1 2 3	Can we add more information to help you? ☐ procedure/step ☐ illustration ☐ feature ☐ example ☐ guideline ☐ other ☐ explanation ☐ definition	
Technical Accuracy (all provided information is correct) 1 2 3	Can we be more accurate? ☐ text ☐ illustration	
Clarity (all provided information is easy to understand) 1 2 3	How can we make things clearer?	
Other Comments	You can add additional comments on the back of this form.	

Your Name _____ Location/Phone _____

Your Title/Function _____ Would you like us to contact you regarding your comments?

___No, there is no need to contact me

___Yes, please call me

___Yes, please email me at _____

___Yes, please contact me via _____

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Phone: 440-646-3176 Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Allen-Bradley
RELIANCE ELECTRIC
DODGE
ROCKWELL SOFTWARE
Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

Rockwell Automation Support

Rockwell Automation provides technical information on the web to assist you in using our 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 with a hardware module 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 module up and running:

United States	1.440.646.3223 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 tests all of our 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:

United States	Contact your distributor. You must provide a Customer Support case number (see 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 return procedure.

Reach us now at www.rockwellautomation.com

Wherever you need us, Rockwell Automation brings together leading brands in industrial automation including Allen-Bradley controls, Reliance Electric power transmission products, Dodge mechanical power transmission components, and Rockwell Software. Rockwell Automation's unique, flexible approach to helping customers achieve a competitive advantage is supported by thousands of authorized partners, distributors and system integrators around the world.

Americas Headquarters, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
European Headquarters SA/NV, avenue Herrmann Debroux, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1789-UM002F-EN-P - March 2004
Supersedes Publication 1789-UM002E-EN-P - June 2003
PN 957867-11

Allen-Bradley

SoftLogix5800 System

User Manual