

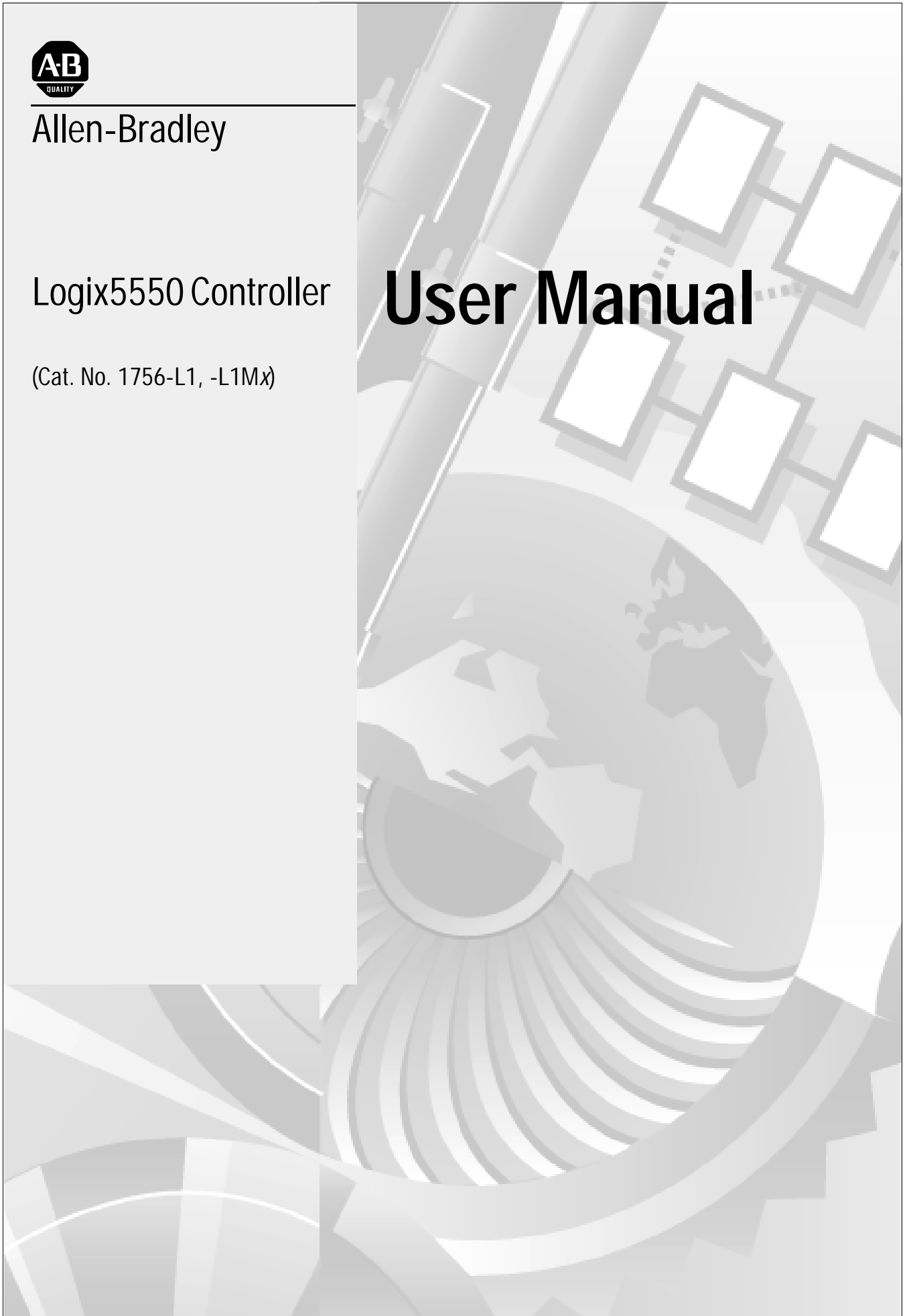


Allen-Bradley

Logix5550 Controller

(Cat. No. 1756-L1, -L1Mx)

User Manual



Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. *Safety Guidelines for the Application, Installation, and Maintenance of Solid State Controls*, publication SGI-1.1 describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will the Allen-Bradley Company be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, the Allen-Bradley Company cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Allen-Bradley Company with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of the Allen-Bradley Company is prohibited.

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.

Attentions help you:

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

Important: Identifies information that is especially important for successful application and understanding of the product.

Introduction

This release of this document contains new and updated information. To help you find the new and updated information, look for change bars, as shown next to this paragraph.

Updated Information

This document has been updated throughout. The most significant changes are:


For this new/updated information:	See chapter:
Upload/download changes	2
Forcing	11

Notes:

Using This Manual

Introduction

This manual is one of several ControlLogix documents.

Task/Goal:	Documents:
Installing the controller and its components	<i>Logix5550 Controller Quick Start</i> , publication 1756-10.1 <i>Logix5550 Memory Board Installation Instructions</i> , publication 1756-5.33
Using the controller You are here 	<i>Logix5550 Controller User Manual</i> , publication 1756-6.5.12
Programming the controller	<i>Logix5550 Controller Instruction Set Reference Manual</i> , publication 1756-6.4.1
Configuring and communicating with digital I/O modules	<i>Digital Modules User Manual</i> , publication 1756-6.5.8
Configuring and communicating with analog I/O modules	<i>Analog Modules User Manual</i> , publication 1756-6.5.9
Selecting and installing a chassis	<i>ControlLogix Chassis Installation Instructions</i> , publication 1756-5.69
Selecting and installing a power supply	<i>ControlLogix Power Supply Installation Instructions</i> , publication 1756-5.1

Who Should Use This Manual

This document provides a programmer with information about how the Logix5550 controller:

- stores and processes data
- operates
- communicates with other modules
- processes and handles fault information

Purpose of This Manual

This manual is intended to help you design and operate a system using a Logix5550 controller. The first chapter in this manual provides the steps and information you need to get started.

Use the remainder of this manual to help you:

- work with controller projects
- configure I/O modules
- organize data
- develop programs
- configure produced and consumed data
- account for communication connections
- communication over a serial network
- communicate over other networks
- identify and process controller faults

Conventions and Related Terms

This manual includes a glossary to define common terms.

Getting Started

Chapter 1

Using This Chapter	1-1
Installing the Controller	1-2
Prepare the controller	1-3
Install the controller	1-3
Creating and Downloading a Project	1-4
Create a project	1-5
Changing project properties	1-6
Adding a local input module	1-7
Adding a local output module	1-9
Changing module properties	1-11
Viewing I/O tags	1-12
Creating other tags	1-13
Documenting I/O with alias tags	1-14
Enter logic	1-16
Download a project	1-18
Viewing program scan time	1-21
Viewing controller memory usage	1-22
What To Do Next	1-22

Working with Projects

Chapter 2

Using This Chapter	2-1
Creating a Project	2-1
Naming controllers	2-2
Changing Project Properties	2-2
Working with the Controller Organizer	2-3
Saving Your Project	2-4
Uploading From the Controller	2-4
Using Coordinated System Time	2-5

Configuring I/O Modules

Chapter 3

Using This Chapter	3-1
Introduction	3-1
Logic Scanning	3-2
Defining I/O Updates	3-2
How an I/O module uses COS	3-2
How an I/O module uses RPI	3-3
When an analog module uses RTS	3-3
How I/O Modules Operate	3-3

Configuring Local I/O	3-4
Naming modules	3-5
Electronic keying	3-6
Configuring communication format	3-7
Selecting controller ownership	3-8
Inhibiting module operation	3-9
Configuring I/O in a Remote Chassis	3-11
Changing Configuration Information	3-15
Accessing I/O	3-16
Example of local addressing	3-17
Example of remote addressing	3-18
Defining aliases	3-19
Viewing Module Fault Information	3-19
Using the programming software to view I/O faults	3-21
Using logic to monitor I/O faults	3-22

Organizing Data

Chapter 4

Using This Chapter	4-1
How the Controller Stores Data	4-1
Creating Tags	4-2
Data types	4-3
Naming tags	4-4
Entering tags	4-4
Using Base Tags	4-6
Memory allocation for base tags	4-6
Data type conversions	4-8
Specifying bits	4-8
Using Structures	4-9
Predefined structures	4-10
Module-defined structure	4-10
User-defined structure	4-10
Memory allocation for user-defined structures	4-11
Referencing members within a structure	4-12
Viewing an Array as a Collection of Elements	4-13
Indexing through arrays	4-14
Specifying Bits Within Arrays	4-15
Viewing an Array as a Block of Memory	4-15
How the controller stores array data	4-16
Varying a dimension	4-17
Memory Allocation for Arrays	4-17
Aliasing Tags	4-19
Scoping Tags	4-20
Scoping tags local to a program	4-21
Scoping tags global to a controller	4-21

Developing Programs

Chapter 5

Using This Chapter	5-1
Organizing Projects	5-1
Defining Tasks	5-2
Using a continuous task	5-3
Using a periodic task	5-3
Creating tasks	5-5
Naming tasks	5-6
Configuring tasks	5-6
Setting the task watchdog	5-8
Avoiding periodic task overlap	5-8
Defining Programs	5-8
Creating programs	5-9
Naming programs	5-9
Configuring programs	5-10
Defining Routines	5-11
Creating routines	5-11
Naming routines	5-12
Configuring routines	5-12
Entering Ladder Logic	5-13
Entering branches	5-14
Scheduling System Overhead	5-15
Downloading a Project	5-16

Communicating with Other Controllers

Chapter 6

Using This Chapter	6-1
Using MSG Instructions	6-1
Communicating with another Logix5550 controller ...	6-1
Communicating with other processors	6-2
Mapping addresses	6-4
Using Produced and Consumed Tags	6-6
Processing produced and consumed tags	6-7
Maximum number of produced and consumed tags ...	6-8
Planning to Support Produced and Consumed Tags	6-9
Identifying another local controller	6-10
Identifying a remote controller	6-10
Producing a Tag	6-12
Consuming a Tag	6-14
Sending Large Arrays of Data	6-17

Allocating Communication Connections

Chapter 7

Using This Chapter	7-1
How the ControlLogix System Uses Connections	7-1
Determining Connections for I/O Modules	7-2
Direct connections for I/O modules	7-2
Rack optimized connections for I/O modules	7-4
Combining direct and rack optimized connections	7-5
Determining Connections for Produced/Consumed Tags	7-6
Connections for produced tags	7-6
Optimizing produced tags	7-7
Connections for consumed tags	7-7
Determining Connections for Messaging	7-7
Determining Total Connection Requirements	7-8

Communicating with Devices on a Serial Link

Chapter 8

Using This Chapter	8-1
Using RS-232	8-1
Connecting to the Serial Port	8-2
Configuring the controller to use the serial port	8-3
Using the DF1 Serial Protocol	8-4
Master/slave communication methods	8-5
Configuring Serial Communications	8-5
Configuring a DF1 point-to-point station	8-6
Configuring a DF1 slave station	8-7
Configuring a DF1 master station	8-8
If you choose one of the standard polling modes	8-9

Communicating with a Workstation

Chapter 9

Using This Chapter	9-1
Configuring Communications to the Controller	9-1
Defining Connection Paths	9-2
Connection path examples	9-4

Integrating Motion

Chapter 10

Using This Chapter	10-1
Introduction.	10-1
Developing a Motion Control Application Program	10-2
Selecting the master controller	10-2
Adding a 1756-M02AE module	10-3
Naming an axis	10-4
Configuring a servo axis	10-5
Running hookup diagnostics and auto tuning.	10-11
Writing a Motion Application Program	10-12
Understanding the MOTION_INSTRUCTION tag	10-13
Using motion status and configuration parameters	10-13
Modifying motion configuration parameters.	10-14
Handling motion faults.	10-14
Understanding errors.	10-14
Understanding minor/major faults	10-14
Understanding a programming example	10-15

Forcing

Chapter 11

Using This Chapter	11-1
Forcing	11-1
Entering Forces	11-2
Entering forces from the data monitor	11-2
Entering forces from the ladder editor	11-3
Enabling Forces.	11-4
Disabling Forces	11-5
Removing Forces	11-5
Monitoring Forces	11-6

Handling Controller Faults

Chapter 12

Using This Chapter	12-1
Understanding Controller Faults.	12-1
Viewing Controller Faults.	12-2
Monitoring I/O Faults.	12-2
Handling Hardware Faults	12-3
Processing Minor Faults	12-3
Processing instruction-execution minor faults	12-4
Writing logic for instruction-execution minor faults	12-5
Processing other minor faults	12-6
Writing logic for other minor faults.	12-7
Minor Fault Types and Codes	12-8
Processing Major Faults	12-9
Writing logic for a major fault	12-12
Major Fault Types and Codes	12-14
Creating a Program Fault Routine	12-16
Creating the Controller Fault Handler.	12-16

Creating a program for the controller fault handler . . .	12-17
Naming programs	12-17
Selecting an unscheduled program	12-17
Configuring programs	12-18
Creating routines	12-19
Naming routines	12-19
Accessing the FAULTLOG	12-20
MajorFaultBits structure	12-20
MinorFaultBits structure	12-20

Preparing a Power-Up Program

Chapter 13

Using This Chapter	13-1
How the Controller Powers Up in Run Mode.	13-1
Processing the power-up handler	13-2
Creating the Power-Up Handler	13-3
Creating a program for the power-up handler	13-3
Naming programs	13-3
Selecting an unscheduled program	13-4
Configuring programs	13-4
Creating routines	13-5
Naming routines	13-6
Clearing the Major Fault	13-6

Troubleshooting

Appendix A

Using This Appendix	A-1
Identifying Controller Components	A-1
Monitoring Controller Status LEDs	A-2
Determining which modules are not responding	A-3
Monitoring Controller Status	A-5
Viewing status through the programming software	A-5
Monitoring status flags	A-6
Using GSV/SSV instructions	A-6
Changing Controller Mode	A-8
Examining Controller Prescan Operations	A-9
Instructions with unique prescan operations	A-9
Recovering from prescan errors	A-10

IEC1131-3 Compliance**Appendix B**

Using This Appendix	B-1
Introduction.	B-1
Operating System	B-2
Data Definitions.	B-2
Programming Languages	B-3
Instruction Set.	B-3
IEC1131-3 Program Portability	B-4
IEC Compliance Tables	B-4

Specifications**Appendix C**

Logix5550 Controller	C-1
Logix5550 Memory Board	C-2
1756-CP3 Serial Cable Pinouts	C-3
1756-BA1 Battery	C-3
1756-M0A2E Motion Module.	C-4

Glossary

Notes:

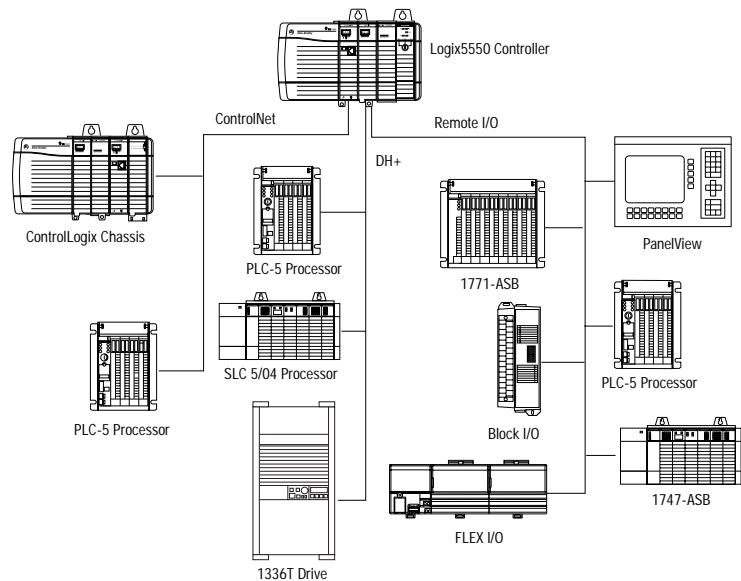
Getting Started

Using This Chapter

This chapter introduces the Logix5550 controller and provides a quick overview on installing the controller and on creating and downloading a project. The steps in this chapter introduce the basic aspects of the Logix5550 controller and refer you to later chapters in this manual for more details.

The Logix5550 controller suits a wide range of control applications by supporting:

- multiple controllers in one ControlLogix™ chassis
- controllers distributed across multiple chassis
- scheduled processor-to-processor communications
- multiple controllers that share the same I/O modules and communications modules



30169

Installing the Controller

The following directions summarize the procedure for installing a Logix5550 controller. For details, see the *Logix5550 Controller Quick Start*, publication 1756-10.1, which ships with the controller.

Take these precautions to guard against ESD damage:



ATTENTION: Electrostatic discharge can damage the components. Follow these guidelines:

- touch a grounded object to discharge potential static
 - wear an approved grounding wriststrap
 - do not touch connectors or connector on component boards
 - do not touch circuit components inside the controller
 - if available, use a static-safe work station
 - when not in use, store each component in the anti-static packaging in which it was shipped
-

You can install or remove ControlLogix system components while chassis power is applied and the system is operating. If you remove the controller, all the devices owned by the controller go to their configured faulted state.



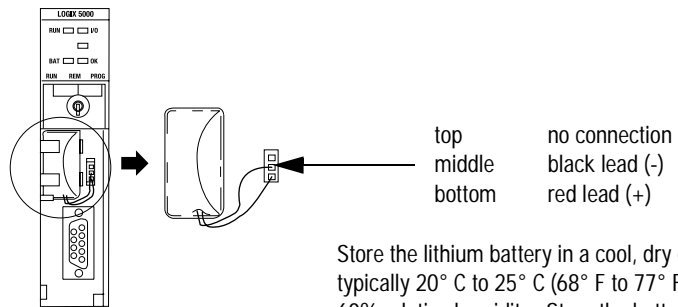
ATTENTION: When you insert or remove a module while backplane power is on, an electrical arc may occur. An electrical arc can cause personal injury or property damage by:

- sending an erroneous signal to your system's actuators causing unintended machine motion or loss of process control
- causing an explosion in a hazardous environment

Repeated electrical arcing causes excessive wear to contacts on both the module and its mating connector. Worn contacts may create electrical resistance that can affect module operation.

Prepare the controller

1. Install the battery.



Store the lithium battery in a cool, dry environment, typically 20° C to 25° C (68° F to 77° F) and 40% to 60% relative humidity. Store the batteries in the original container, away from flammable materials.

30167



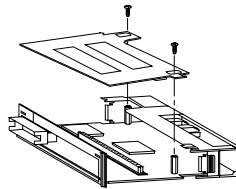
Attention: Only install a 1756-BA1 battery.

For more information, see *Guidelines for Handling Lithium Batteries*, publication 1756-5.68.

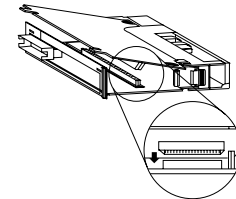
2. Install the memory expansion board, if any.

a. Remove the side plate.

b. Attach the memory board.



40017



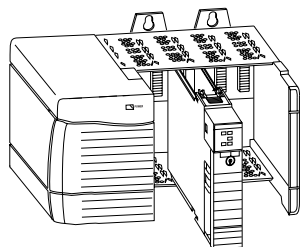
40018

The Logix5550 controller can be purchased with a memory expansion board already installed (catalog numbers 1756-L1M1, -L1M2, or -L1M3).

For more information, see the *Logix5550 Memory Board Installation Instructions*, publication 1756-5.33.

Install the controller

You can place the Logix5550 controller in any slot. You can use multiple Logix5550 controllers in the same chassis. The total number of modules in a chassis depends on power supply capacity.



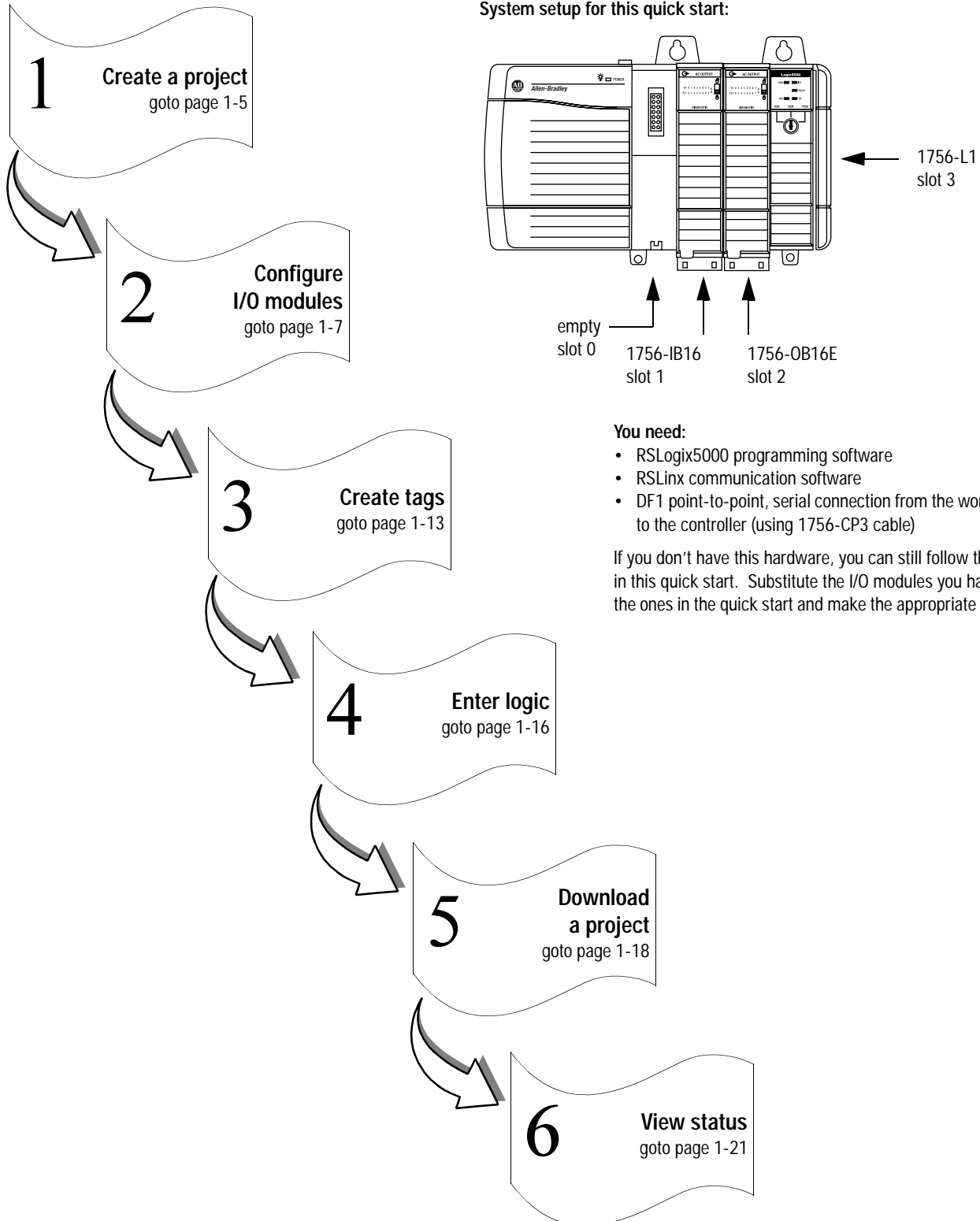
20880

1. Align the circuit board with the top and bottom guides in the chassis.
2. Slide the module into the chassis.
3. Make sure the module properly connects to the chassis backplane.

The controller is fully installed when it is flush with the power supply or other fully-installed modules and the top and bottom latches are engaged.

Creating and Downloading a Project

The following diagram illustrates the steps you follow to create and download a project. The remainder of this quick start provides examples of each step, with references to other chapters in this manual for more details.



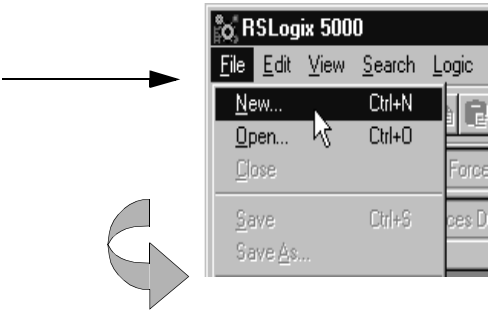
Create a project

To follow the steps in this quick start, RSLogix5000 programming software must already be installed and running.

1 Create a project

see chapter 2

1. Select File → New to create a project.



2. Define the project.

The software uses the project name you enter with an .ACD extension to store your project.

You must enter a name. →

Select the chassis type and specify the slot number of the controller. (You will have to change the default values.) →

Describe the project (optional). →

Select where to store the project (typically use the default directory). →

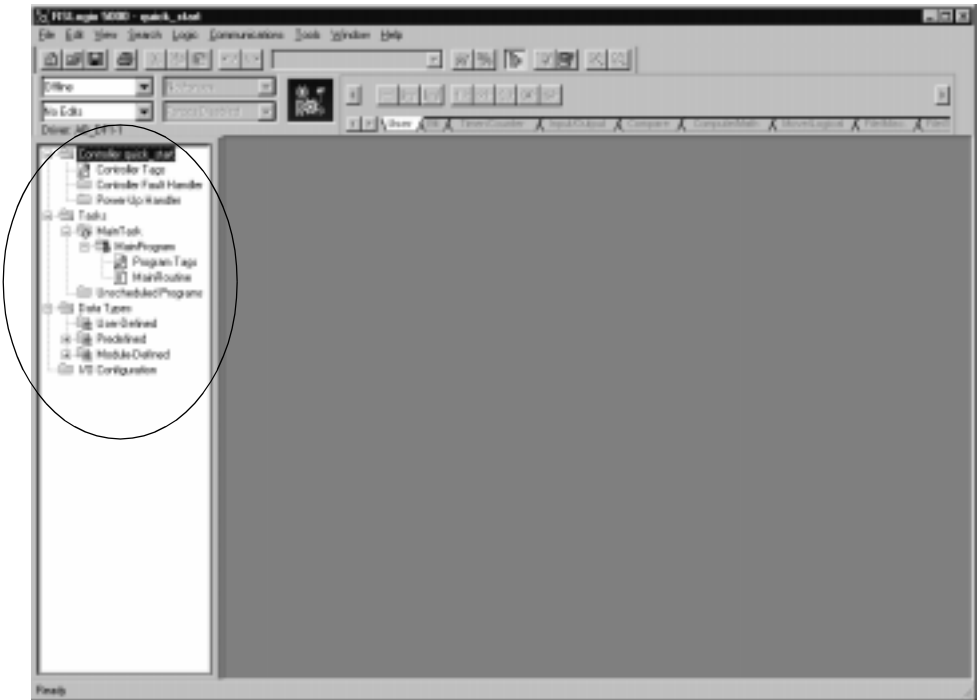
New Controller

Click OK. →

A screenshot of the 'New Controller' dialog box in RSLogix 5000. The dialog has several fields: 'Name' (containing 'quick_start'), 'Chassis Type' (a dropdown menu showing '1756-A4A 4-Slot Chassis'), 'Slot Number' (a numeric field with '3'), 'Description' (a text area containing 'This is a sample control system for the quick start'), and 'Create In' (a text field showing 'C:\RSLogix 5000\Projects' with a 'Browse...' button). At the bottom are 'OK', 'Cancel', and 'Help' buttons. Arrows from the text annotations point to the corresponding fields in the dialog.

The software displays:

controller organizer →



Changing project properties

1. View properties for Controller quick_start.

A. Place the cursor over the Controller quick_start folder. →

B. Click the right mouse button and select Properties.



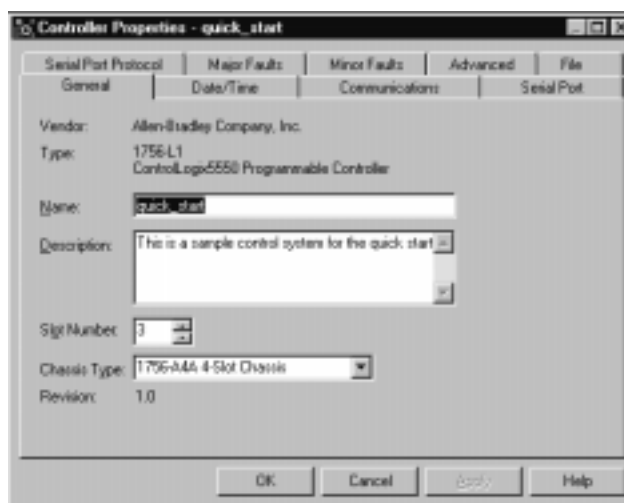
1 Create a project

see chapter 2

2. View the General tab.

The screen defaults to the General tab.

Verify that the controller settings are correct. Make changes if necessary. →



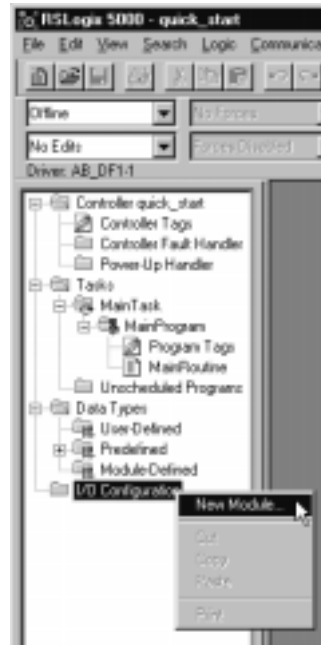
Click OK.

From this tab:	You can:
General	modify the controller name, description, and controller properties for the current project
Date/Time	<i>online only</i> view and edit the controller's wall clock time and the coordinated system time status.
Communications	configure communication information that is stored with the project file
Serial Port	view and configure the serial port on the controller
Serial Port Protocol	configure the serial port for: <ul style="list-style-type: none"> • DF1 point-to-point • DF1 slave • DF1 master
Major Faults	<i>online only</i> view any major faults that have occurred on the controller
Minor Faults	<i>online only</i> view any minor faults that have occurred on the controller
Advanced	<i>some features are online only</i> view and edit advanced controller properties, which include the system fault program, the power loss program, and system overhead time slice
File	view information about the project file

Adding a local input module

1. Create a new module.

- A. Place the cursor over the I/O Configuration folder.
- B. Click the right mouse button and select New Module

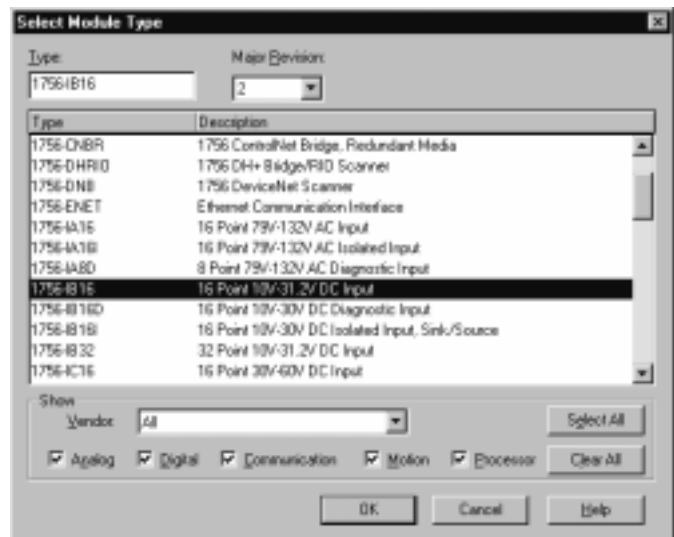


2 Configure I/O modules

see chapter 3

2. Select an input module to add.

Select a catalog number.
For this quick start example, select 1756-IB16.



Click OK.

continued

Adding a local input module (*continued*)

3. Identify the input module. These screens are specific to the 1756-IB16 input module.

2 Configure I/O modules

see chapter 3

- You should enter a name.
Verify the slot number.
Describe the module (optional).
Select the communication format.
Specify electronic keying.

Click Next.

4. Use the Create wizard to configure the input module.

Use default values for this quick start example.

If you do not want to page through each screen in the Create wizard, click Finish to create the module using default values.

Click Next.

Click Next.

Click Next.

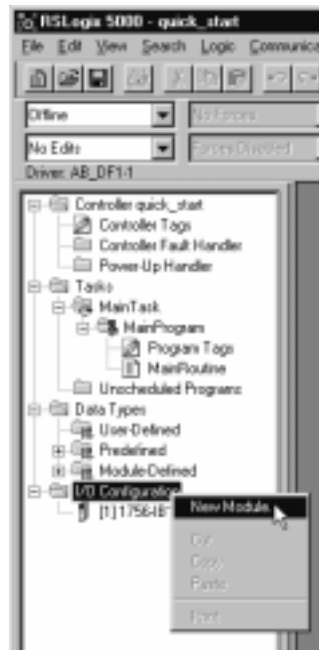
Click Finish.

continued

Adding a local output module

1. Create a new module.

- A. Place the cursor over the I/O Configuration folder.
- B. Click the right mouse button and select New Module

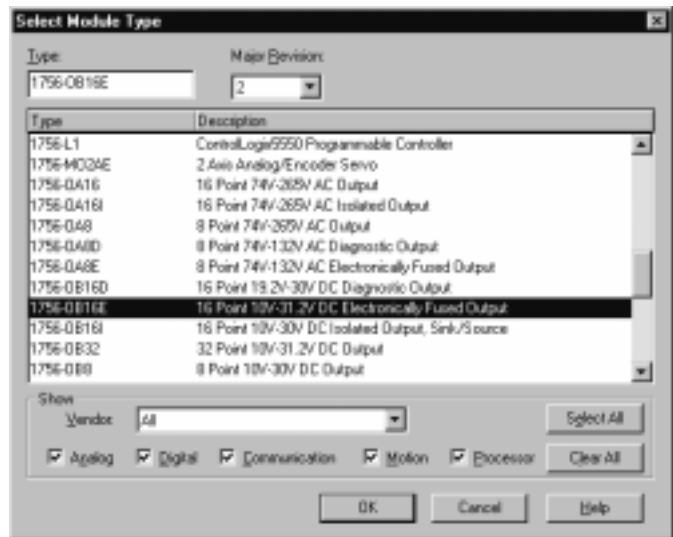


2 Configure I/O modules

see chapter 3

2. Select an output module to add.

- Select a catalog number.
For this quick start example, select 1756-OB16E.



Click OK.

continued

Adding a local output module (*continued*)

3. Identify the output module. These screens are specific to the 1756-OB16E output module.

You should enter a name.
Verify the slot number.
Describe the module (optional).
Select the communication format.
Specify electronic keying.

2 Configure I/O modules
see chapter 3

Click Next.

4. Use the Create wizard to configure the output module.

Use default values for this quick start example.

If you do not want to page through each screen in the Create wizard, click Finish to create the module using default values.

Click Next.

Click Next.

Click Next.

Click Next.

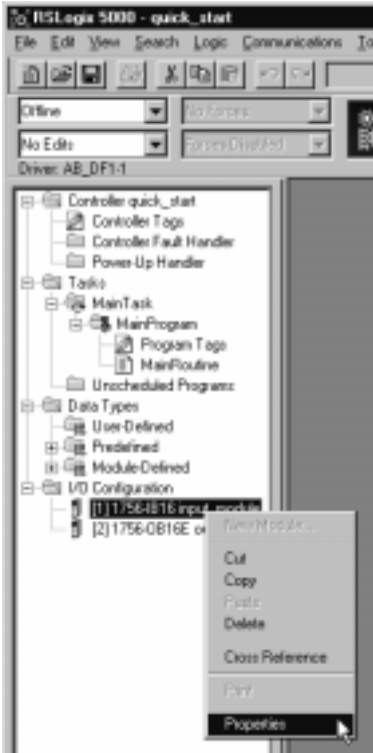
Click Finish.

continued

Changing module properties

1. View properties for the module.

- A. Place the cursor over the 1756-IB16 module.
- B. Click the right mouse button and select Properties



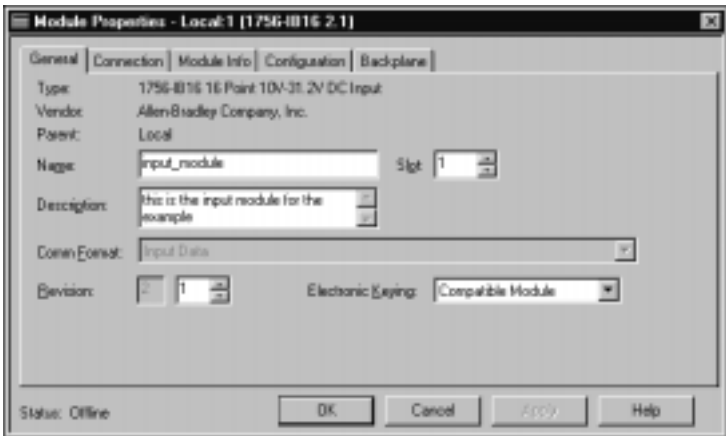
2Configure I/O modules

see chapter 3

2. View the General tab.

The screen defaults to the General tab.

Verify that the module settings are correct. Make changes if necessary.



Click OK.

The tabs that appear depend on the type of module.

From this tab:	You can:
General	modify the properties for the current module
Connection	define controller to module behavior: <ul style="list-style-type: none">select requested packet intervalchoose to inhibit the connection to the moduleconfigure the controller so loss of connection generates a major faultview module faults (online only)
Module Info	<i>online only</i> view module identification and status information reset module to power-up state
Configuration	configure the module
Backplane	<i>online only</i> view information about module's communication over the backplane clear module faults set transmit retry limit

Viewing I/O tags

1. View the module-defined tags.

Place the cursor on the Controller Tags folder and double-click.



2

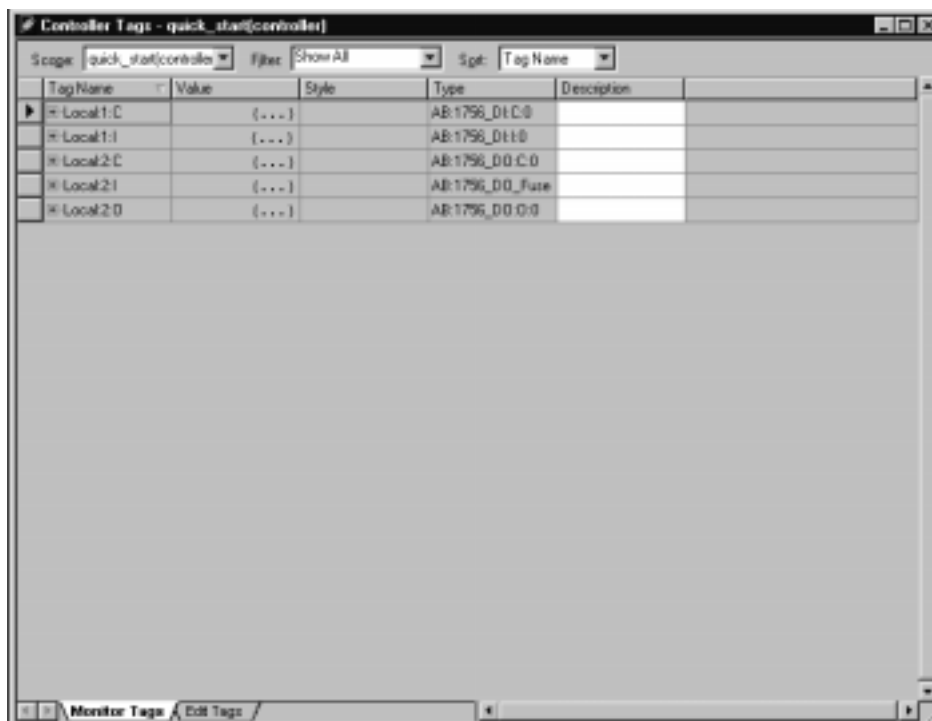
Configure
I/O modules

see chapter 3

The software displays the module-defined tags for the I/O modules you created.

The 1756-IB16 input
module is in slot 1.

The 1756-OB16E output
module is in slot 2.

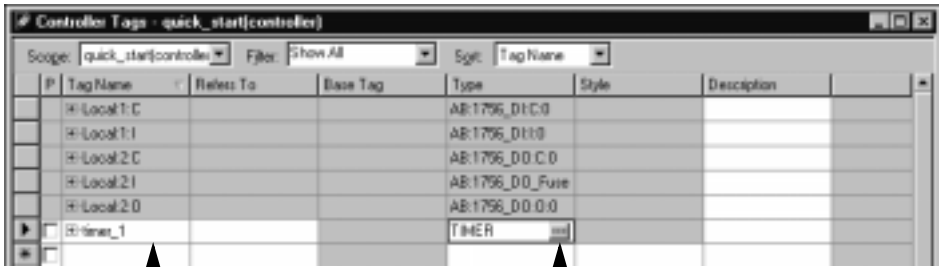


Click the Edit Tags tab.

continued

Creating other tags

1. Create a tag.

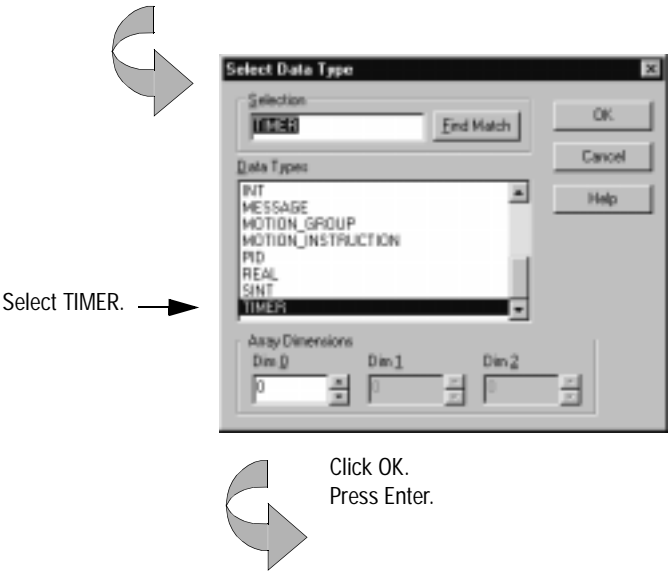


Enter the name of the new tag.

Tab to this column and select the data type.

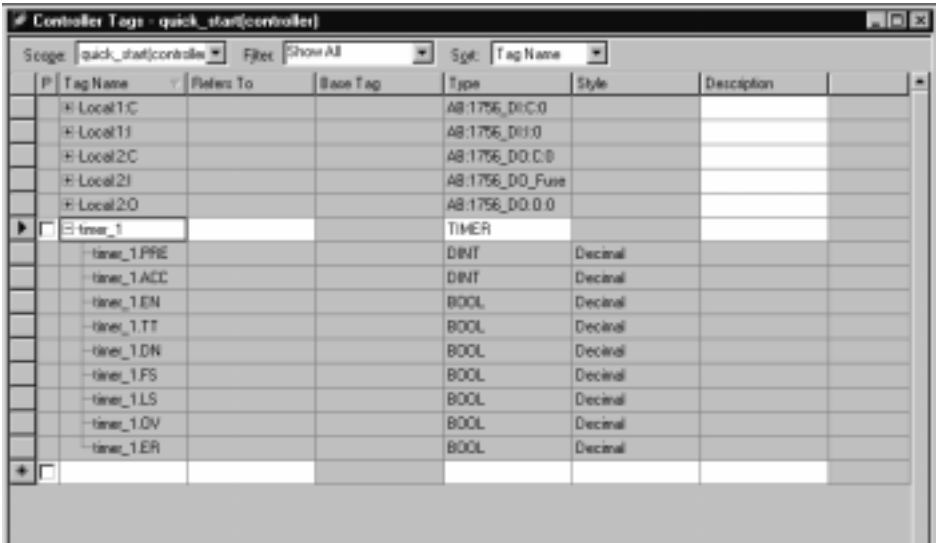
3 Create tags
see chapter 4

2. Select the data type.



The software displays the tag.

Click + to display the members of the TIMER structure.



You might have to resize the column to see the tag extensions.

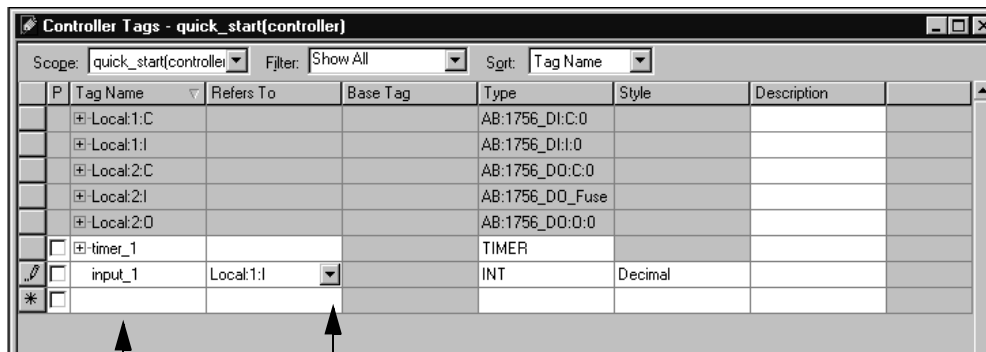
continued

Documenting I/O with alias tags

1. Create an alias tag *input_1* for Local:1:I.Data.1.

3 Create tags

see chapter 4

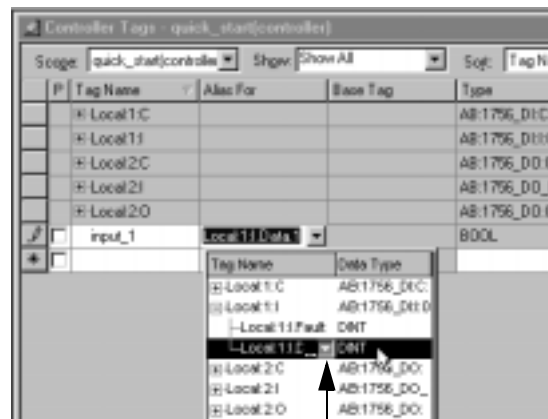


Enter the name of the tag.

Tab here or click in the box.
Click here to select tag to reference.

2. Select an input data word.

- A. Click on the input data structure.
- B. Click + to display the members of the structure.
- C. Click the input data word



Click here to select a bit.

2. Select a specific bit.

Click on the bit.



Press Enter.

continued

Documenting I/O with alias tags *(continued)*

4. Repeat steps 1 and 2 above to create an alias tag *output_1* for Local:2:0.Data.1

The software displays the alias tags.

3

Create tags

see chapter 4

Controller Tags - quick_start(controller)							
Scope: quick_start(controller)		Filter: Show All		Sort: Tag Name			
P	Tag Name	Refers To	Base Tag	Type	Style	Description	
	Local 1: C			AB 1756-DI C:0			
	Local 1: I			AB 1756-DI I:0			
	Local 2: C			AB 1756-DO C:0			
	Local 2: I			AB 1756-DO Fuse			
	Local 2: O			AB 1756-DO O:0			
	timer_1			TIMER			
	input_1	Local 1: I Data 1	Local 1: I Data 1	BOOL	Decimal		
	output_1	Local 2: O Data 1	Local 2: O Data 1	BOOL	Decimal		

Enter logic

1. Use default task, program, and routine.

When you created the project, the software automatically created a MainTask, MainProgram, and MainRoutine. Use these defaults for the quick start.

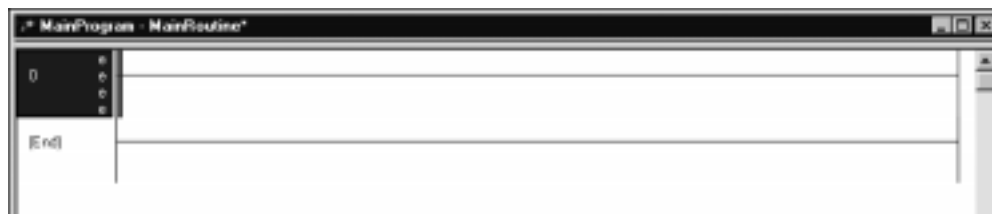
4 Enter logic

see chapter 5

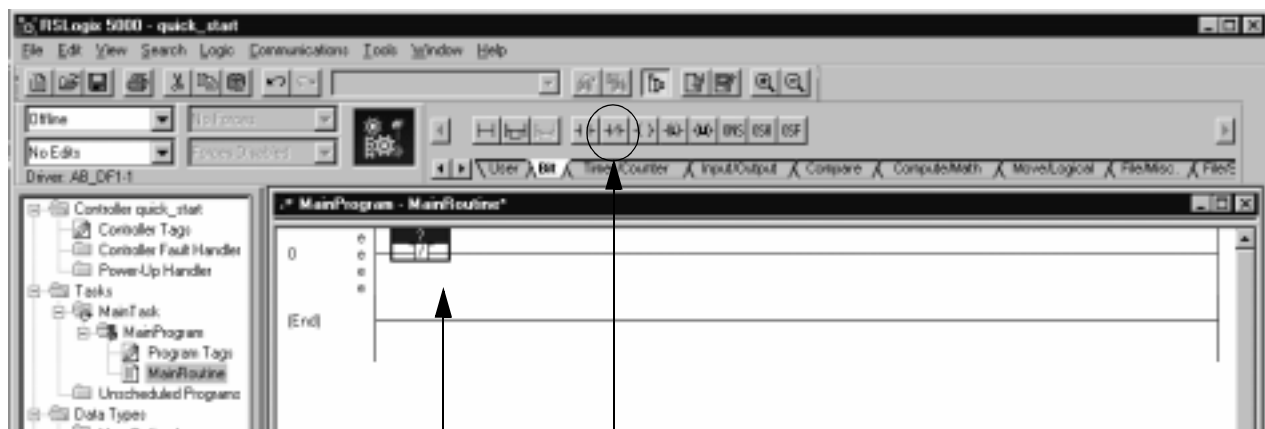
Double-click MainRoutine.



The software displays an empty routine.



2. Enter an XIO instruction.

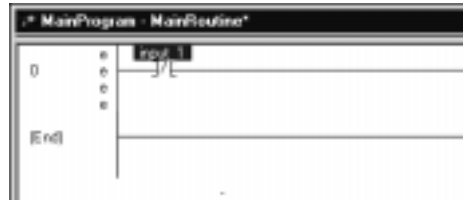


Drag and drop the XIO instruction on an empty rung.

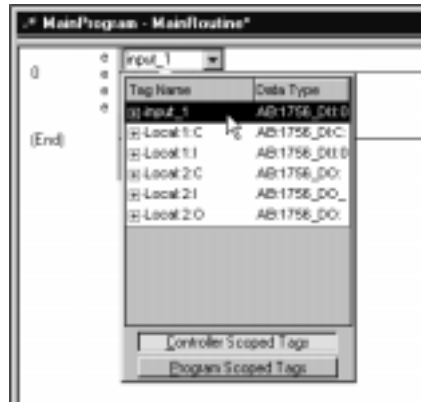
Entering logic (*continued*)

3. Assign a tag to the XIC instruction.

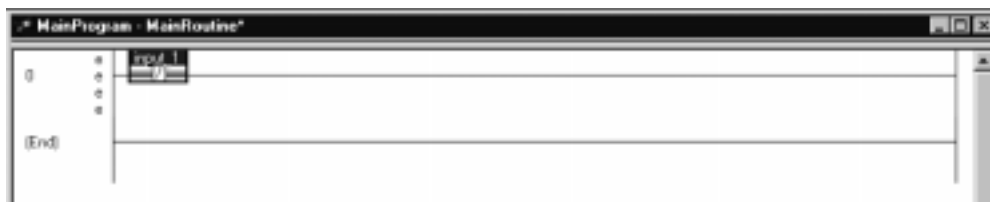
Double-click the tag area of the instruction.



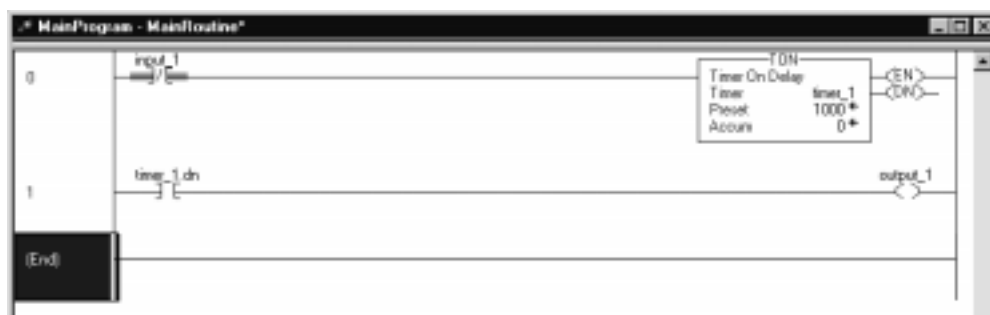
Use the drop-down menu to select the alias tag *input_1*.



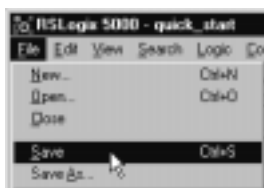
The software displays an incomplete rung.



4. Enter this logic.



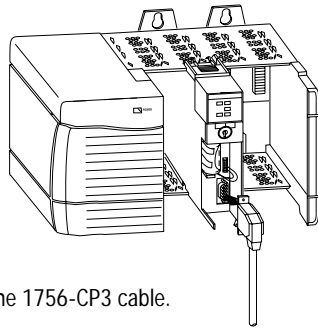
4. Select File → Save to save the project.



see chapter 5

Download a project

1. Make a serial connection from the workstation to the controller.



Use the 1756-CP3 cable.



see chapter 5 and
chapter 8

2. Configure the controller's serial port for DF1 point-to-point.

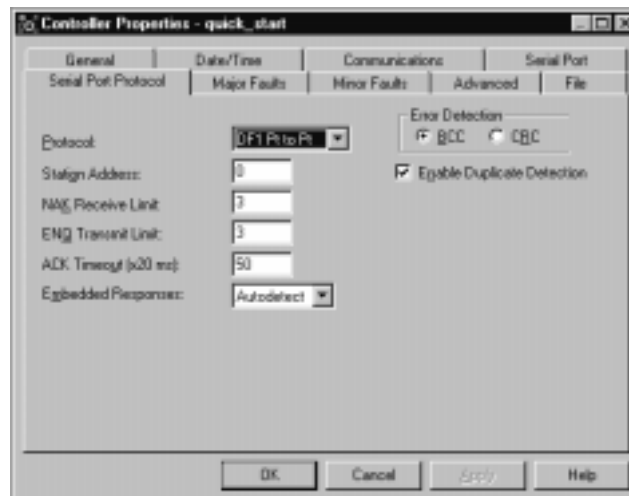
A. Place the cursor over the Controller quick_start folder.

B. Click the right mouse button and select Properties.



A. View Serial Port Protocol.

B. Select DF1 Pt. to Pt.



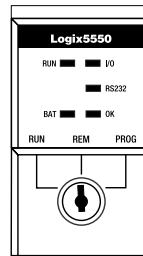
continued

Download a project (*continued*)

3. Turn the controller's keyswitch to PROG and then back to REM.

Make sure the keyswitch is in the REM position.

This places the controller in Remote Program mode.

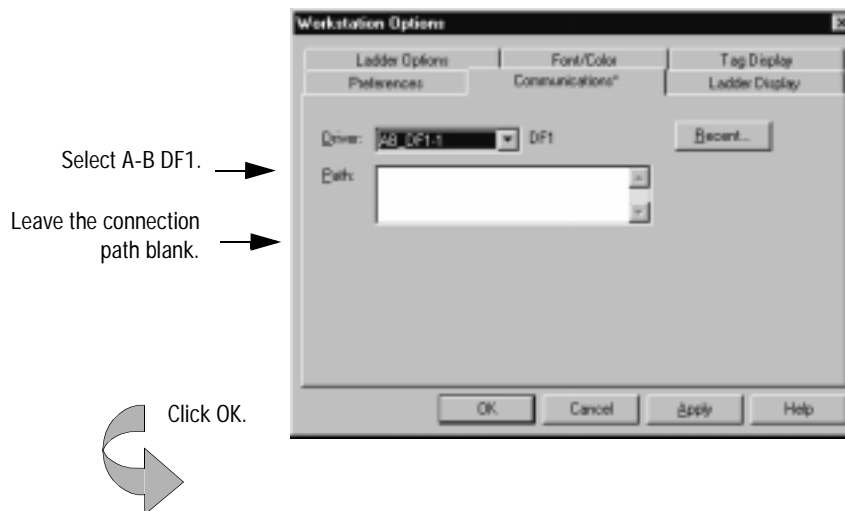


see chapter 5 and
chapter 8

4. Select Communications → Configure.



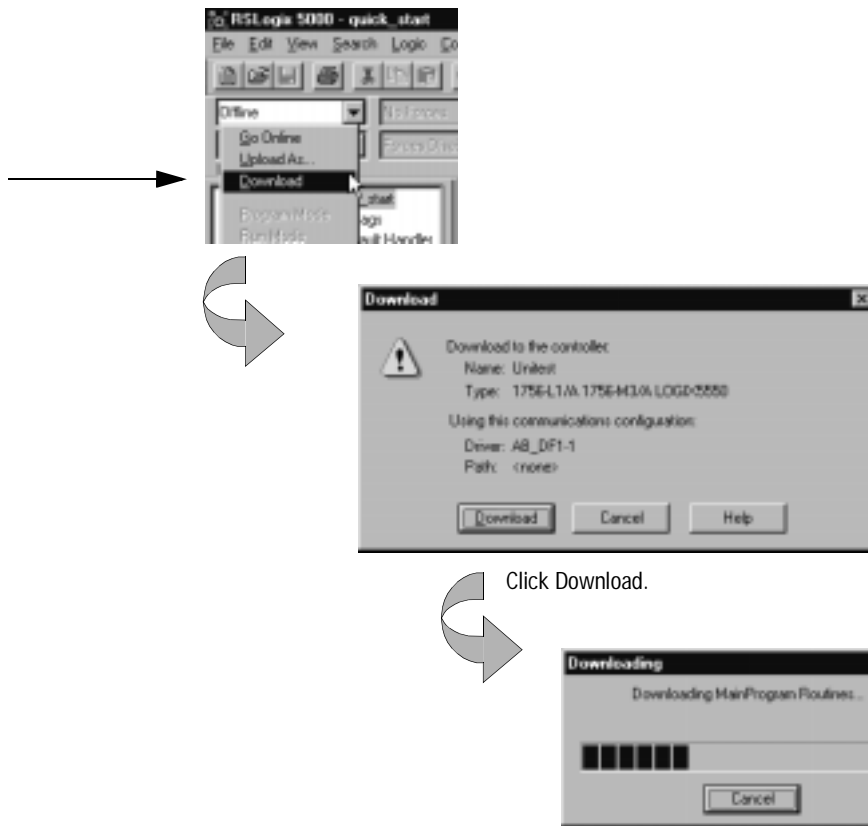
5. Select the DF1 communication protocol.



Important: The DF1 driver only shows as a communication choice if you have already configured a DF1 driver using RSLinx communication software.

Download a project (*continued*)

6. Select Download.



see chapter 5 and
chapter 8

7. Put the controller in Run mode.

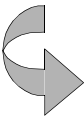
Viewing program scan time

1. View properties for the MainProgram.

- A. Place the cursor over the MainProgram folder.
B. Click the right mouse button and select Properties.



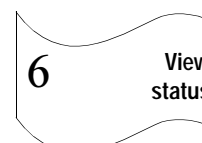
2. Select the Configuration tab.



This tab displays the maximum and last scan times for the program.

Viewing controller memory usage

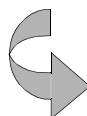
1. View properties for Controller quick_start.



see chapter 5

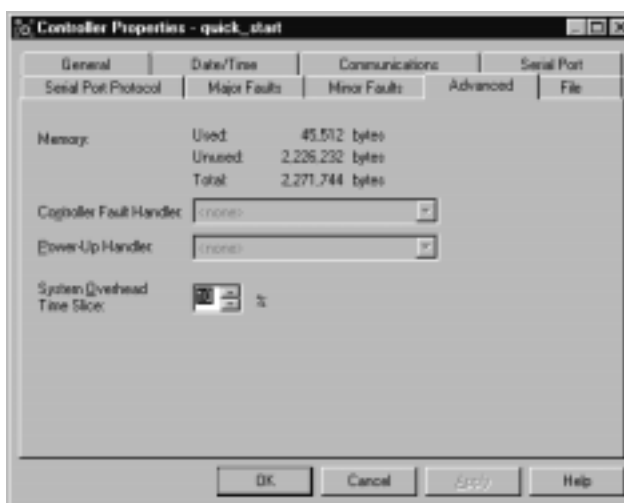
A. Place the cursor over the Controller quick_start folder.

B. Click the right mouse button and select Properties



2. Select the Advanced tab.

In addition to other information, this tab displays controller memory usage.



What To Do Next

Once your controller is installed and operating, you can begin developing and testing your control application. Use RSLogix5000 programming software.

Use the remaining chapters in this manual as reference material for developing and testing your control application. The remaining chapters provide detailed information about how the controller operates.

Working with Projects

Using This Chapter

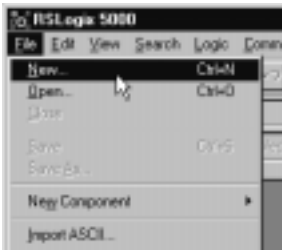
For information about:	See page:
Creating a project	2-1
Changing project properties	2-2
Working with the controller organizer	2-3
Saving your work	2-4
Uploading from the controller	2-4
Using coordinated system time (CST)	2-5

Creating a Project

Before you can begin programming or configuring the controller, you must create a project file. The project file is the file on the hard drive of your workstation that stores logic and configuration information. The project file has an .ACD extension.

To create a project, specify this information:

1. Select File → New.



In this field:	Enter:
Name	Enter the name of the controller for this application. This name is also used for the project file (with an .ACD extension). The name is required.
Chassis Type	Select the type of chassis that contains the controller. Use the pull-down menu to select from the available types.
Slot Number	Select the slot number where the controller is installed.
Description	Enter a description of the controller (optional).
Create In	Select where to store the project file on the hard drive of your workstation. You can use the default (which was configured when the software was installed) or specify a different location.

Naming controllers

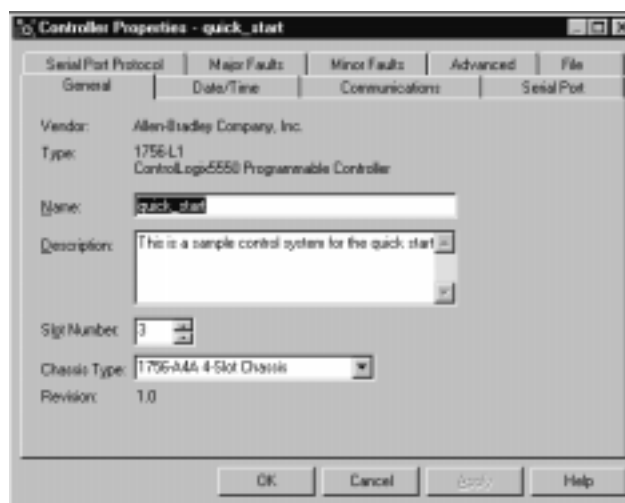
Controller names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

You can also add a description. Descriptions can have as many as 128 characters. You can use any printable character.

Changing Project Properties

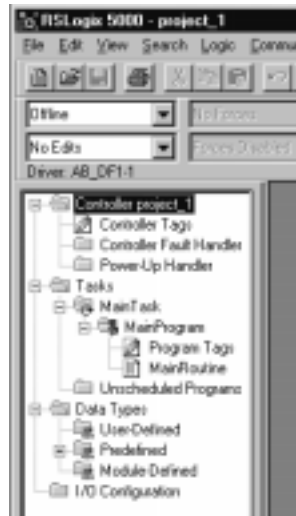
1. Place the cursor over the Controller folder.
2. Click the right mouse button and select Properties.



From this tab:	You can:
General	modify the controller name, description, and controller properties for the current project
Date/Time	<i>online only</i> view and edit the controller's wall clock time and the coordinated system time status.
Communications	configure communication information that is stored with the project file
Serial Port	view and configure the serial port on the controller
Serial Port Protocol	configure the serial port for: <ul style="list-style-type: none"> • DF1 point-to-point • DF1 slave • DF1 master
Major Faults	<i>online only</i> view any major faults that have occurred on the controller
Minor Faults	<i>online only</i> view any minor faults that have occurred on the controller
Advanced	<i>some features are online only</i> view and edit advanced controller properties, which include the system fault program, the power loss program, and system overhead time slice
File	view information about the project file

Working with the Controller Organizer

The controller organizer is a graphical representation of the contents of a project. The display uses folders and files to group information about logic and configuration.



In front of each folder, there is an icon with a + sign or a – sign. The + sign indicates that the folder is closed. Click on it to display the files in the folder. The – sign indicates that the folder is already open and its contents are visible.

Click the right mouse button on any item in the controller organizer to display a context-sensitive menu for that item. These pop-up menus are often shortcuts to using options from the menu bar. The examples in this manual most often use right-click actions on items in the controller organizer.

Saving Your Project

As you create logic and make configuration changes, save your work to the project file.

If you:	This is what happens:
Save	The programming software saves programming and configuration changes to the current project file. The title bar of the programming software displays the name of the current project file.
Save As	<p>The programming software creates a new project file using the current project file and the name you specify.</p> <p>The controller name is independent of the project file name. If you save a current project file as another name, the controller name is unchanged. Use controller properties to change the controller name to match the project name.</p>

If you are programming online when you save your project, data values are uploaded from the controller and saved as well.

Important: If you do not want the data values uploaded from the controller, go offline before saving the project.

Uploading From the Controller

If you do not have the project file for a controller, you can upload from the controller and create a project file. However, not everything that is stored in a project file is available from the controller. If you upload from a controller, the new project file will not contain:

- rung comments
- descriptions for tags, tasks, programs, routines, modules, or user-defined structures
- chains of aliases (aliases pointing to other aliases)

Alias chains are not completely reconstructed from the controller. If there are several possible names for a data item, the firmware and software choose a best-fit alias that may not reflect how the alias was specified in the original project.

1. Select Upload.



If you upload a project from a controller and there is not a matching project on the workstation with the same name, use Select File to enter a name. This process saves the project to the workstation using the name you enter. The project will not have any comments and descriptions, because this information is not stored in the controller.

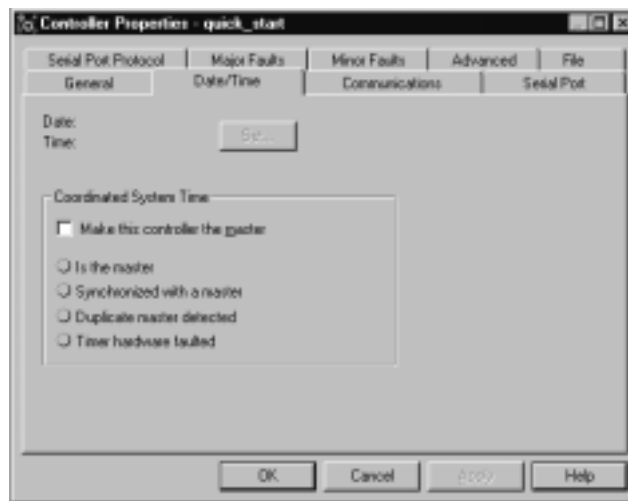
If you upload a project from a controller and a matching project file with the same name already exists on the hard drive of the workstation, the upload process offers two choices. If you use Select File and enter a new name, the process saves the project to the workstation under a different name. If you select Upload Merge, the process merges the project image in the controller with the comments and descriptions in the project file on the workstation.

Using Coordinated System Time

1. Place the cursor over the Controller folder.
2. Click the right mouse button and select Properties.



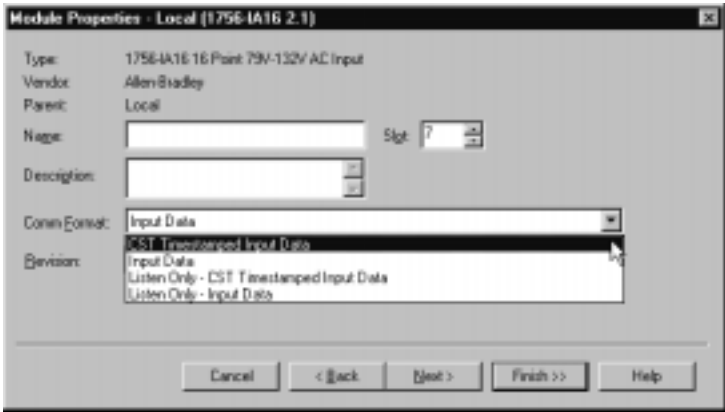
Only one controller in a chassis
can be the CST master.



The CST value is stored as an array of two DINT elements. The `TIMESTAMP[0]` element stores the lower 32 bits; the `TIMESTAMP[1]` element stores the upper 32 bits.

You can compare the CST clocks of different modules in the same chassis for timekeeping purposes. For example, knowing when an input bit changed by checking the CST timestamp from the input module, you can schedule an output bit to change 4.736 seconds later according to the CST clock in the output module. For an example of using timestamped inputs to schedule outputs, see the *ControlLogix Digital I/O Modules User Manual*, publication 1756-6.5.

Not all I/O modules support the CST communication format. You select CST when you specify the communication format as you add the I/O module to the controller organizer.



The controller also has a WALLCLOCKTIME object that is similar to the CST timestamp. The WALLCLOCKTIME object has a DateTime attribute that provides the time that has elapsed since 12:00 am 1 January 1972.

Use a GSV instruction to capture the DateTime attribute of the WALLCLOCKTIME object into a DINT[7] array.

This element:	Contains:
DINT[0]	year
DINT[1]	integer representation of month (1-12)
DINT[2]	integer representation of day (1-31)
DINT[3]	hour (0-23)
DINT[4]	minutes (0-59)
DINT[5]	seconds (0-59)
DINT[6]	microseconds (0-999,999)

You could also use a GSV instruction to capture the CurrentValue attribute of the WALCLOCKTIME object into a DINT[2]. This provides the number of microseconds that have elapsed since 12:00 am 1 January 1972.

This element:	Contains:
DINT[0]	lower 32 bits of value
DINT[1]	upper 32 bits of value

Configuring I/O Modules

Using This Chapter

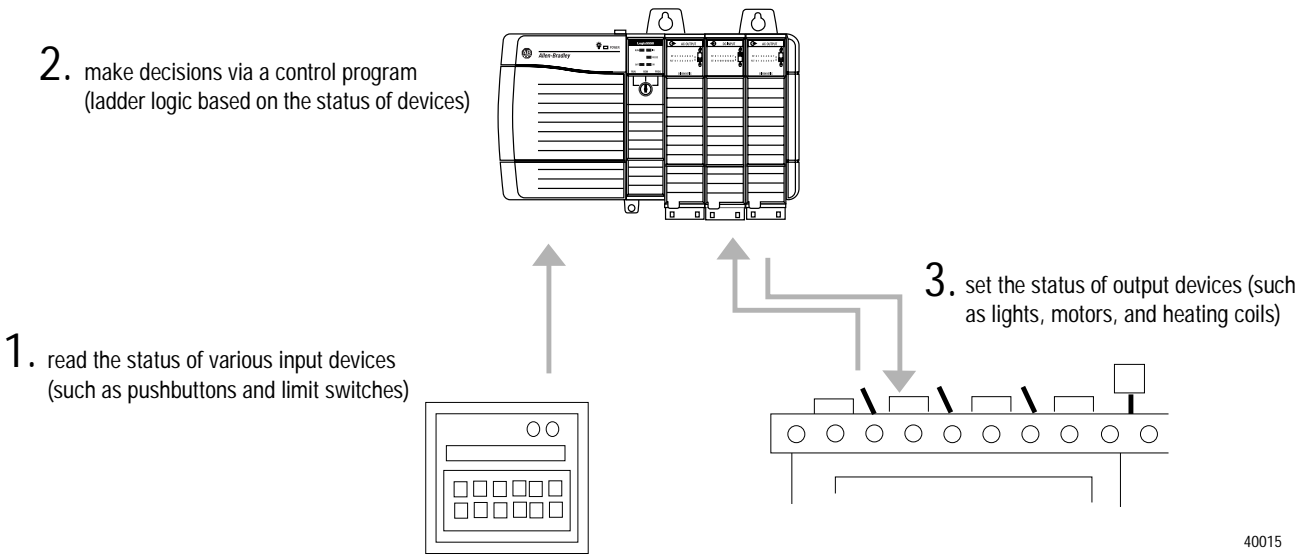
For information about:	See page:
How the controller scans I/O	3-1
Defining I/O updates	3-2
How I/O modules operate	3-3
Configuring local I/O	3-4
Configuring remote I/O	3-11
Accessing I/O	3-16
Viewing module fault records	3-19

The configuration information for the module depends on the module you selected. For more information, see the user documentation for the specific module.

This document:	Has this publication number:
Digital Modules User Manual	1756-6.5.8
Analog Modules User Manual	1756-6.5.9

Introduction

The basic function of a programmable controller is to:



40015

The controller performs two primary functions:

- executes logic
- reads input data and sends output data

Logic Scanning

The controller continually scans the control logic. One scan is the time it takes the controller to execute the logic once. Input data transfers to the controller asynchronous to the logic scan. The controller transfers output data at the end of each and every program scan.

If you want input data to remain constant throughout one scan, make a copy of the input data at the beginning of the scan and use the copy throughout the scan.

Defining I/O Updates

The ControlLogix system follows a producer/consumer model. Input modules produce data for the system. Output modules, controllers, and intelligent modules produce and consume data.

The producer/consumer model multicasts messages. This means that multiple nodes can consume the same data at the same time from a single device. Where you place I/O modules in the control system determines how the modules exchange data.

If the I/O module is:	And you place the module here:	The data exchange method is based on:
digital	local chassis	change of state and requested packet interval
	remote chassis	requested packet interval
analog	local chassis	real time sample and requested packet interval
	remote chassis	real time sample and requested packet interval

How an I/O module uses change-of-state (COS)

Digital input modules in the local chassis use the change-of-state method to transfer data. This method transfers data whenever an input point changes from ON to OFF or OFF to ON.

Use change-of-state data exchange in projects where:

- data changes rapidly, such as counting, timing, and position referencing applications
- data is digitally-intensive, such as packaging and material-handling applications

You must specify an RPI regardless of whether you enable COS. If a change does not occur within the RPI timeframe, the module multicasts data at the rate specified by the RPI.

How an I/O module uses the requested packet interval (RPI)

The requested packet interval is a cyclic data exchange that specifies the rate at which a module multicasts its data. Data is updated at a rate that is appropriate to the module and your project. You can reserve bandwidth for rapidly-changing modules. Data updated at precise intervals provides for better determinism.

Use cyclic data exchange in projects where:

- data changes slowly, such as measuring temperature or flow
- data exchange must be predictable and repeatable
- you need precision sampling for closed-loop control (PID)
- data is needed for trending, data logging, etc.

When an analog module uses real-time sampling (RTS)

Analog input modules use real-time sampling (RTS). The analog module scans all the input channels but multicasts only the channel data that changed.

How I/O Modules Operate

The type of module and where you place the module determines how the module operates:

Module Type:	Placement:	Operation:
digital input	local chassis	<p>The RPI specifies the rate at which a module multicasts its data. The time ranges from 200 microseconds to 750 milliseconds. When the specified time frame elapses, the module will multicast data.</p> <p>If a change of state (COS) does not occur within the RPI timeframe, the module multicasts data at the rate specified by the RPI.</p> <p>Because the RPI and COS functions are asynchronous to the logic scan, it is possible for an input to change state during program scan execution. Buffer input data so your logic has a stable copy of data during its scan. Copy the input data from your input tags to another structure and use the data from there.</p>
	remote chassis	<p>The RPI and COS values still define when the module multicasts data within its own chassis, but only the value of the RPI determines when the owner controller receives the data over the network.</p> <p>When an RPI value is specified for an input module in a remote chassis, in addition to instructing the module to multicast data within its own chassis, the RPI also “reserves” a spot in the stream of data flowing across the ControlNet network. The timing of this “reserved” spot may or may not coincide with the exact value of the RPI, but the owner-controller will receive data at least as often as the specified RPI.</p>
digital output	local chassis	<p>If the module resides in the same chassis as the owner-controller, the module receives the data almost immediately after the owner-controller sends it.</p>
	remote chassis	<p>If an output module resides in a chassis other than that of the owner-controller (i.e. a remote chassis connected via ControlNet), the owner-controller sends data to the output module only at the RPI rate.</p> <p>The RPI also “reserves” a spot in the stream of data flowing across the ControlNet network. The timing of this “reserved” spot may or may not coincide with the exact value of the RPI, but the output module receives data at least as often as the specified RPI.</p>

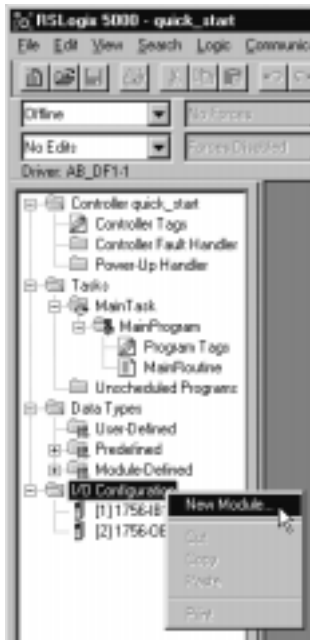
Module Type:	Placement:	Operation:
analog input	local chassis	<p>The RTS value specifies when to multicast updated channel data. The RPI value specifies when to multicast all its current channel data.</p> <p>The module resets the RPI timer each time an RTS transfer occurs. If the RTS value is less than or equal to the RPI value, each multicast of data from the module has newly updated channel data. The module only multicasts at the RTS rate.</p> <p>If the RTS value is greater than the RPI, the module multicasts at both the RTS rate and the RPI rate.</p>
	remote chassis	<p>The RPI and RTS rates still define when the module multicasts data within its own chassis, but only the RPI value determines when the owner-controller receives the data over the network.</p> <p>The RPI also “reserves” a spot in the stream of data flowing across the ControlNet network. The timing of this “reserved” spot may or may not coincide with the exact value of the RPI, but the controller receives data at least as often as the specified RPI.</p>
analog output	local chassis	<p>The RPI value specifies when the owner-controller broadcasts output data to the module. If the module resides in the same chassis as the owner-controller, the module receives the data almost immediately after the owner-controller sends it.</p>
	remote chassis	<p>If an output module resides in a chassis other than that of the owner-controller (i.e. a remote chassis connected via ControlNet), the owner-controller sends data to the output module only at the RPI rate.</p> <p>The RPI also “reserves” a spot in the stream of data flowing across the ControlNet network. The timing of this “reserved” spot may or may not coincide with the exact value of the RPI, but the output module receives data at least as often as the specified RPI.</p>

Configuring Local I/O

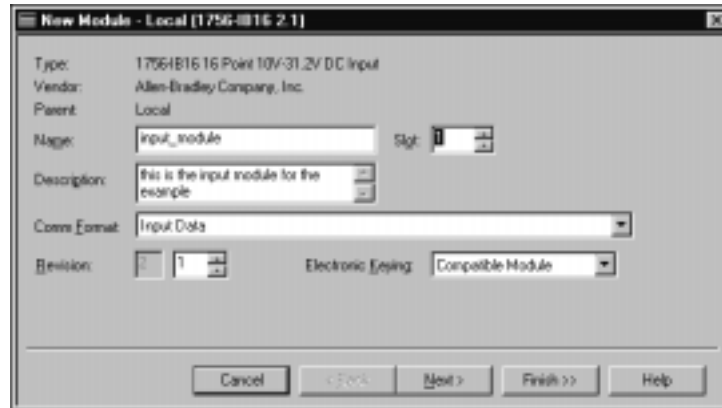
You use your programming software to configure the I/O modules for the controller.

When you configure an I/O module, you specify characteristics specific to that module. The programming software automatically adds the module-defined tags for the module as controller-scoped tags.

1. Select I/O Configuration.
2. Click the right mouse button and select New Module.



To configure an I/O module, select which module to install. Then specify this information:



In this field:	Enter:
Name	Enter a name for the module (optional).
Description	Enter a description for the module (optional).
Slot Number	Enter the slot number where the module is installed.
Communication Format	Select one of the communication formats supported by the module. Some formats specify controller ownership of the module. The communication format can also define the data structure the module uses.
Electronic Keying	Select an electronic keying method.

After you identify the I/O module, the programming software displays additional configuration screens, which depend on the type of module. Once you finish the configuration, the I/O module appears in the controller organizer.

Naming modules

Module names follow IEC 1331-3 identifier rules and:

- must begin with an alphabetic character or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

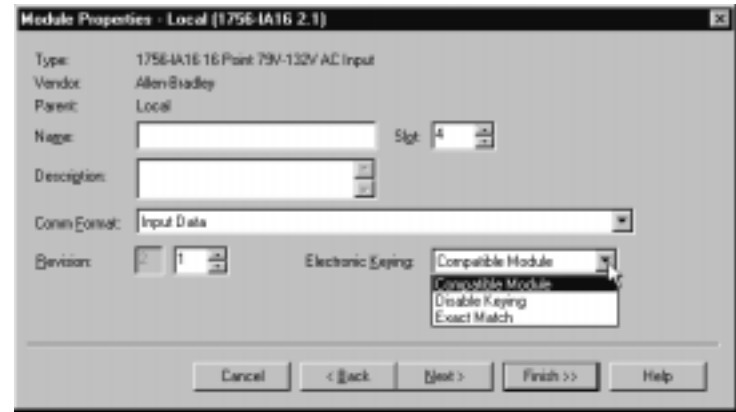
You can also add descriptions to modules. Descriptions can have as many as 128 characters. You can use any printable character.

Electronic keying



ATTENTION: Be careful when you disable electronic keying. If used incorrectly, this option can lead to personal injury or death, property damage, or economic loss.

Specify electronic keying to ensure that a module being inserted or configured is the proper revision.



Keying:	Description:
compatible module	<p>The module must be compatible with the software configuration. These characteristics must match:</p> <ul style="list-style-type: none">• module type• catalog number• major revision <p>The minor revision must be equal to or greater than the one specified in the software.</p>
disable keying	<p>No attributes of the software or hardware are required to match.</p>
exact match	<p>The module must match the software configuration exactly. These characteristics must match:</p> <ul style="list-style-type: none">• module type• catalog number• major revision• minor revision



ATTENTION: Changing the RPI and electronic keying selections may cause the connection to the module to be broken and may result in loss of data.

ATTENTION: Be extremely cautious when using the disable keying option. If used incorrectly, this option can lead to personal injury, death, property damage, or economic loss.

Configuring communication format

The communication format determines the data structure the I/O module uses, as well as the type of connection made to the module and the controller ownership of the module. Many I/O modules support different formats. Each format supports a different data structure.

Use the documentation for the I/O module to determine what data format to use. The larger data formats use more controller memory and use more bandwidth on the communication network.

For example, the following structures are available for a 1756-IB16 module. The communication format determines the predefined tags.

communication format: input data

Local:1:C	{...}		AB:1756_DI:C:0
Local:1:C.DiagCOSDisable	0	Decimal	BOOL
Local:1:C.FilterOffOn_0_7	1	Decimal	SINT
Local:1:C.FilterOnOff_0_7	1	Decimal	SINT
Local:1:C.FilterOnOff_8_15	1	Decimal	SINT
Local:1:C.FilterOnOff_8_15	1	Decimal	SINT
Local:1:C.FilterOnOff_16_23	0	Decimal	SINT
Local:1:C.FilterOnOff_16_23	0	Decimal	SINT
Local:1:C.FilterOnOff_24_31	0	Decimal	SINT
Local:1:C.FilterOnOff_24_31	0	Decimal	SINT
Local:1:C.COSOnOffEn	2#0000_0000_0000_0000_1111_1111_1111_1111	Binary	DINT
Local:1:C.COSOffOnEn	2#0000_0000_0000_0000_1111_1111_1111_1111	Binary	DINT
Local:1:I	{...}		AB:1756_DI:I:0
Local:1:I.Fault	2#0000_0000_0000_0000_0000_0000_0000_0000	Binary	DINT
Local:1:I.Data	2#0000_0000_0000_0000_0000_0000_0000_0000	Binary	DINT

communication format: listen only

Local:7:I	{...}		AB:1756_DI:I:0
Local:7:I.Fault	2#0000_0000_0000_0000_0000_0000_0000_0000	Binary	DINT
Local:7:I.Data	2#0000_0000_0000_0000_0000_0000_0000_0000	Binary	DINT

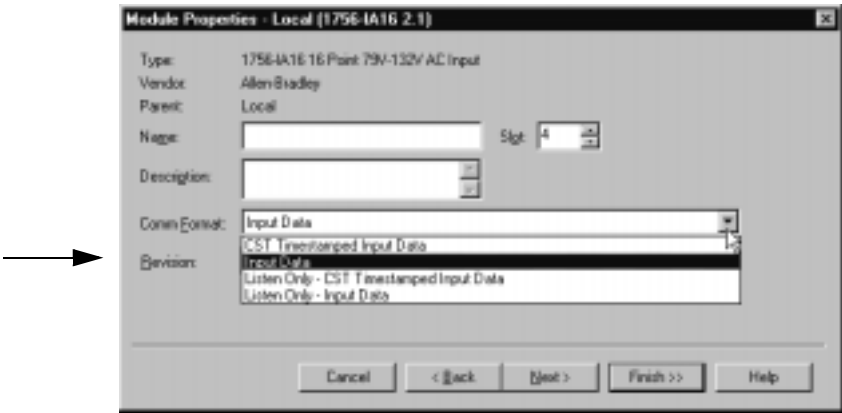
Selecting controller ownership

The ControlLogix architecture makes it possible for more than one controller to communicate with (own) one I/O module. Multiple controllers can own an input module; only one controller can own an output module.

There is a noted difference in controlling input modules versus controlling output modules.

Controlling:	This ownership:	Description:
input modules	owner	An input module is configured by a controller that establishes a connection as an owner. This configuring controller is the first controller to establish an owner connection. Once an input module has been configured (and owned by a controller), other controllers can establish owner connections to that module. This allows additional owners to continue to receive multicast data if the original owner controller breaks its connection to the module. All other additional owners must have the identical configuration data and identical communications format that the original owner controller has, otherwise the connection attempt is rejected.
	listen-only	Once an input module has been configured (and owned by a controller), other controllers can establish a listen-only connection to that module. These controllers can receive multicast data while another controller owns the module. If all owner controllers break their connections to the input module, all controllers with listen-only connections no longer receive multicast data.
output modules	owner	An output module is configured by a controller that establishes a connection as an owner. Only one owner connection is allowed for an output module. If another controller attempts to establish an owner connection, the connection attempt is rejected.
	listen-only	Once an output module has been configured (and owned by one controller), other controllers can establish listen-only connections to that module. These controllers can receive multicast data while another controller owns the module. If the owner controller breaks its connection to the output module, all controllers with listen-only connections no longer receive multicast data.

You specify ownership by selecting the communications format when you configure the I/O module.



Inhibiting module operation

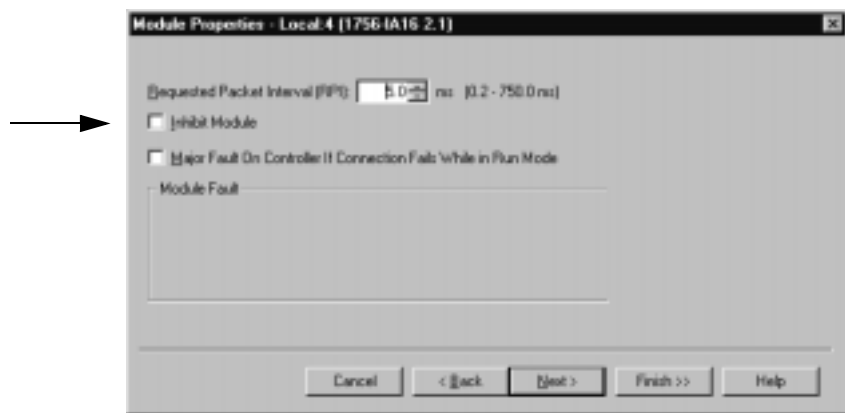
In some situations, such as when initially commissioning a system, it is useful to disable portions of a control system and enable them as you wire up the control system. The controller lets you inhibit individual modules or groups of modules, which prevents the controller from trying to communicate with the modules.

When you configure an I/O module, it defaults to being not inhibited. You can change an individual module's properties to inhibit a module.




ATTENTION: Inhibiting a module causes the connection to the module to be broken and prevents communication of I/O data.

On the Connection tab of the module properties in the programming software, you can select to inhibit that specific module.

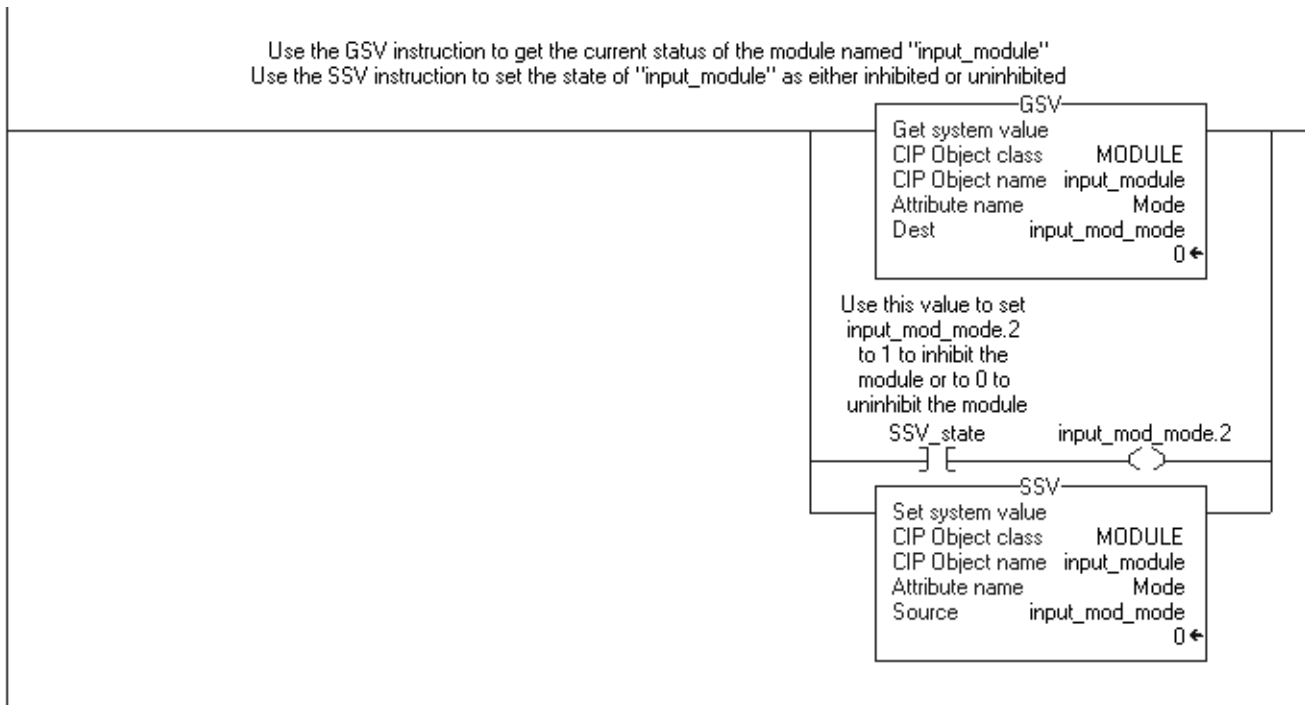


When you inhibit a communication bridge module, such as a 1756-CNB or 1756-DHRIO module, the controller shuts down the connections to the bridge module and to all the modules that depend on that bridge module. Inhibiting a communication bridge module lets you disable an entire branch of the I/O network.

When you select to inhibit the module, the controller organizer displays a yellow attention symbol () over the module.

If you are:	Inhibit a module to:
offline	<p>put a place holder for a module you are configuring</p> <p>The inhibit status is stored in the project. When you download the project, the module is still inhibited.</p>
online	<p>stop communication to a module</p> <p>If you inhibit a module while you are connected to the module, the connection to the module is closed. The modules' outputs go to the last configured program mode.</p> <p>If you inhibit a module but a connection to the module was not established (perhaps due to an error condition or fault), the module is inhibited. The module status information changes to indicate that the module is inhibited and not faulted.</p> <p>If you uninhibit a module (clear the checkbox), and no fault condition occurs, a connection is made to the module and the module is dynamically reconfigured (if the controller is the owner controller) with the configuration you created for that module. If the controller is configured for listen-only, it cannot reconfigure the module.</p> <p>If you uninhibit the module and a fault condition occurs, a connection is not made to the module. The module status information changes to indicate the fault condition.</p>

To inhibit a module from logic, you must first read the Mode attribute for the module using a GSV instruction. Set bit 2 to the inhibit status (1 to inhibit or 0 to uninhibit). Use a SSV instruction to write the Mode attribute back to the module. For example:



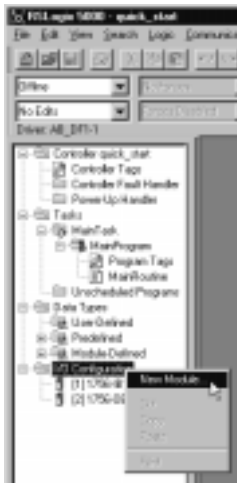
Configuring I/O in a Remote Chassis

Configuring I/O in a remote chassis is similar to configuring local I/O. The difference is that you must also configure the communication module in the local chassis and the communication module or adapter in the remote chassis.

The following example shows how to add the remote chassis and I/O to the controller organizer. How you configure the communication and I/O modules depend on the network. For details, see:

For a:	Use this module:	See this publication:
DH+ or remote I/O network	1756-DHRIO	<i>Data Highway Plus and Remote I/O Communication Interface Module User Manual</i> publication 1756-6.5.2
ControlNet network	1756-CNB	<i>ControlNet Communication Interface User Manual</i> publication 1756-6.5.3
Device Net network	1756-DNB	<i>DeviceNet Scanner Configuration User Manual</i> publication 1756-6.5.15
Ethernet network	1756-ENET	<i>Ethernet Communication Interface Module Manual</i> , publication 1756-6.5.1

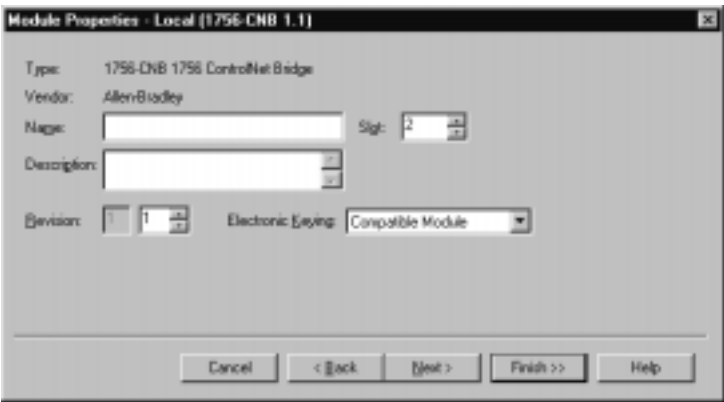
- 1. Select I/O Configuration.
- 2. Click the right mouse button and select New Module.



- 1. Select the local communication module.
- 2. Click the right mouse button and select New Module.



- 1. Configure a communication module for the local chassis. This module handles communications between the controller chassis and the remote chassis. Then specify this information:



In this field:	Enter:
Name	Enter a name for the module (required).
Description	Enter a description for the module (optional).
Slot Number	Enter the slot number where the module is installed.
Electronic Keying	Select an electronic keying method.

- 2. Configure a communication module or adapter for the remote chassis to communicate with the module you just configured. This module handles communication for the remote chassis. Then specify this information:



In this field:	Enter:
Name	Enter a name for the module. The name of a communication module is required. The programming software uses the name to create tag names for I/O in the chassis.
Description	Enter a description for the module (optional).
Slot Number	Enter the slot number where the module is installed.
Communication Format	Select one of the communication formats supported by the module. The format determines the I/O communication method. For more information on I/O communications, see chapter 7.
Node	Enter the node number of the module.
Chassis Size	Enter the chassis size (number of slots) of the remote chassis.
Electronic Keying	Select an electronic keying method.

When you click on a local communication module and add a remote communication module, the local module becomes the “parent module” to the remote module. The controller organizer shows this parent/child relationship between local and remote modules.

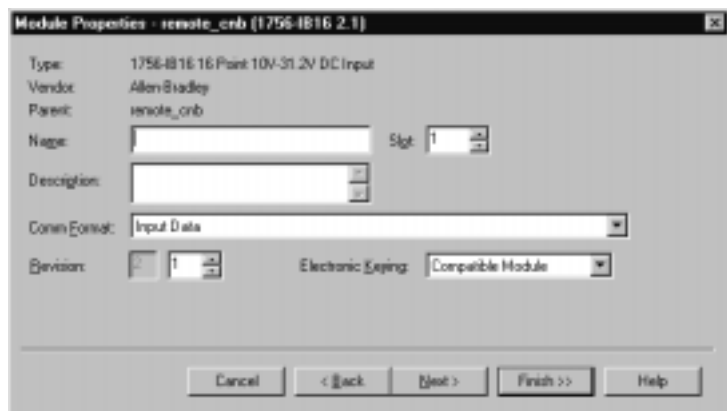
If you are configuring a 1756-CNB module for the remote chassis:

- A.** Add I/O to the chassis.
- B.** Run RSNetworkx software to configure the connections.
- C.** Download the project to the Logix5550 controller.

1. Select the remote communication module.
2. Click the right mouse button and select New Module.



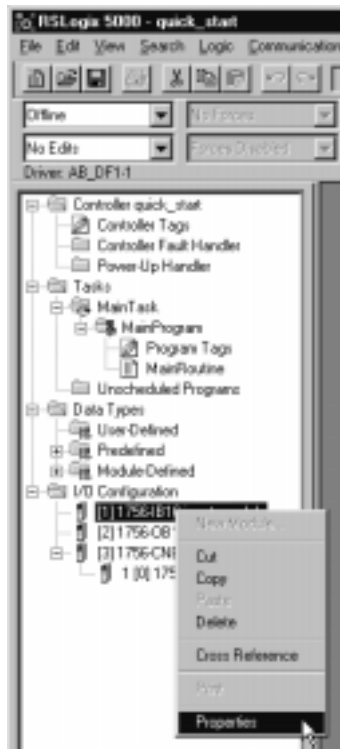
3. Now you can configure the I/O modules for the remote chassis by adding them to the remote communication module. Follow the same procedure as you do for configuring local I/O modules.



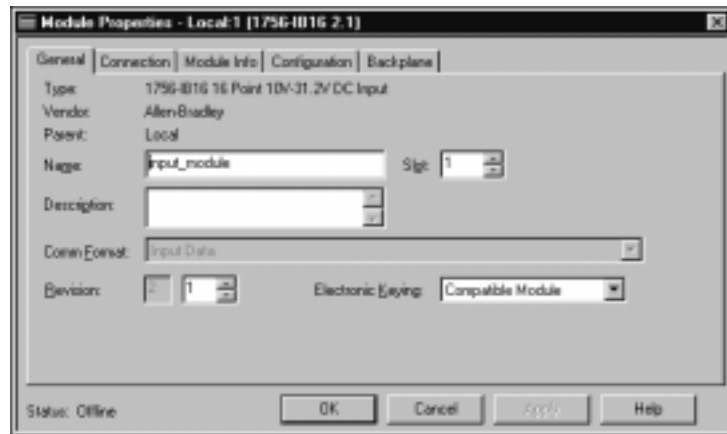
In this field:	Enter:
Name	Enter a name for the module (optional).
Description	Enter a description for the module (optional).
Slot Number	Enter the slot number where the module is installed.
Communication Format	Select one of the communication formats supported by the module. Some formats specify controller ownership of the module. A format can also define the data structure the module uses.
Electronic Keying	Select an electronic keying method.

Changing Configuration Information

1. Select a module ("1756-IB16" in this example).
2. Click the right mouse button and select Properties.



Once you configure an I/O module, you can change configuration information. The configuration tabs that are available depend on the type of module. To change the configuration of an existing module (this example is for a 1756-IB16 module):



On this tab:	In this field:	Enter:
General	Name	The programming software displays the current name of the program. Edit the name, if necessary.
	Description	The programming software displays the current description. Edit the description, if necessary.
	Slot Number	The programming software automatically displays the current slot number. Edit the slot number, if necessary.
	Communication Format	The programming software displays the current communication format. You cannot change the selection from here - you must delete the module and re-create it with a different selection.
	Electronic Keying	The programming software displays the current electronic keying method. Change this method, if necessary.
Connection	Requested Packet Interval	The programming software displays the current RPI setting. Edit the RPI, if necessary. You can select from 0.1-750.0 msec.
	Inhibit Module	The programming software displays whether or not the module is inhibited. Change this selection, if necessary.
	Major Fault	The programming software displays whether or not the controller generates a major fault if the connection to this module fails. Change this selection, if necessary.
Module Info	The programming software displays product and status information about the module. You can reset the module. There are no fields to select or enter data.	
Configuration	Enable Change of State	The programming software displays the current COS setting for each I/O point. Change these selections, if necessary.
	Input Filter Time	The programming software displays the current input filter time settings for the I/O module. Change these selections, if necessary.
Backplane	The programming software displays backplane status information. There are no fields to select or enter data. You can clear faults and reset the status counters.	

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 information 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

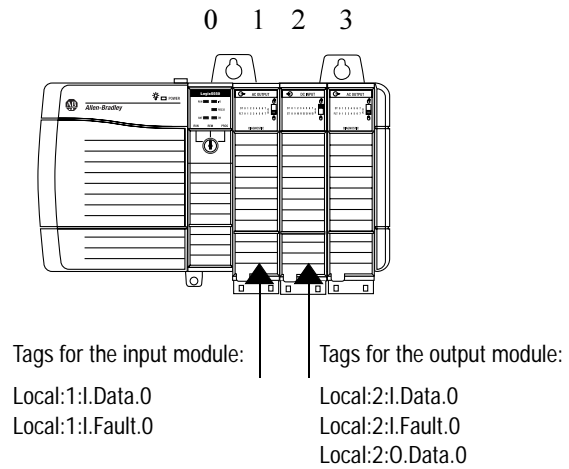
where:

This address variable:	Is:
Location	Identifies network location LOCAL = local chassis ADAPTER_NAME = identifies remote chassis communication adapter or bridge module
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 what 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 more information on tags, see chapter 4.

Example of local addressing

This example addresses a bit in an I/O module that resides in the local chassis.

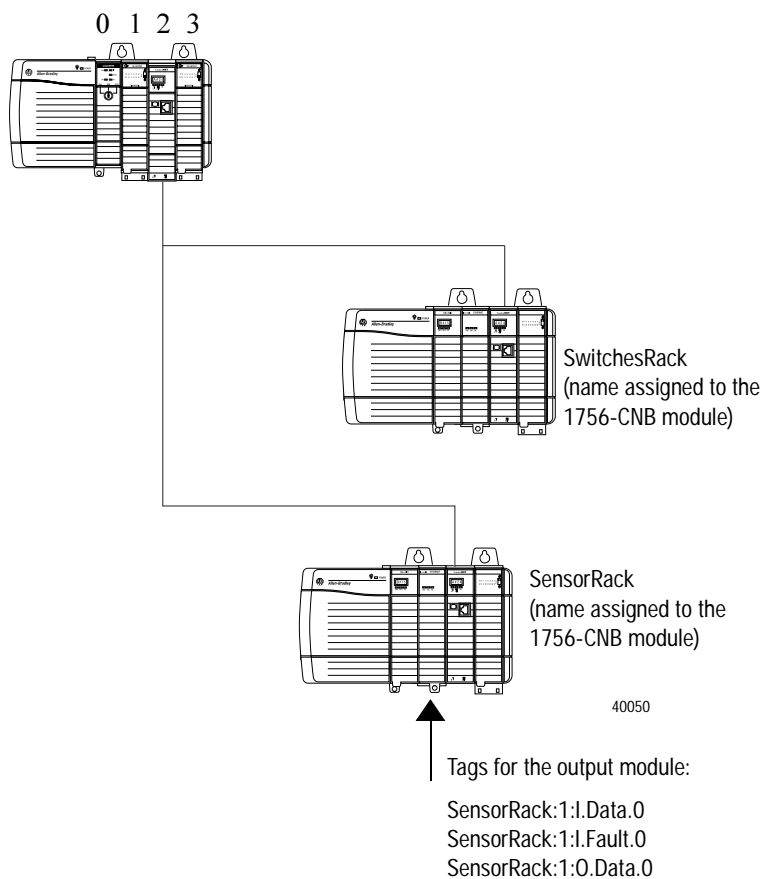


40049

The name *Local* indicates that these tags reference modules that are in the same chassis as the controller.

Example of remote addressing

This example addresses an I/O module in a remote chassis.





Defining aliases

A tag alias lets you create one tag that represents another tag. This is useful for defining simplified tag names for I/O values. For example:

Example:	Description:
I/O structure <i>Local:0:0.Data.0</i> <i>Local:0:1.Fault.0</i> alias <i>light_on</i> = <i>Local:0:0.Data.0</i> <i>light_off</i> = <i>Local:0:1.Fault.0</i>	This example uses simpler tags to refer to specific I/O points.

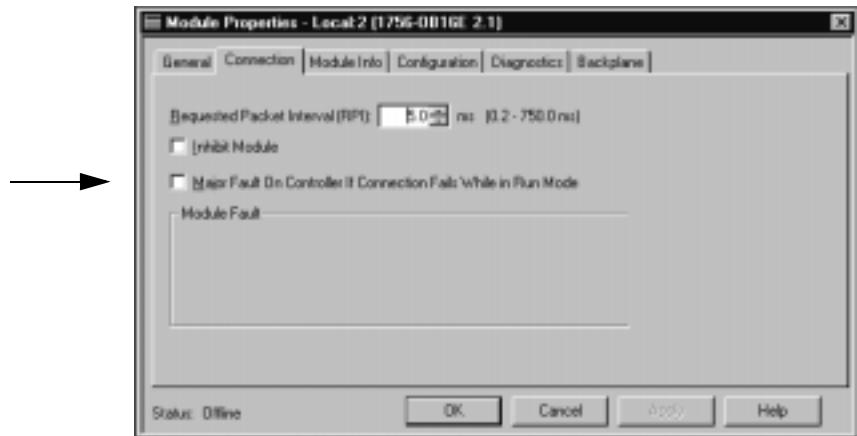
Viewing Module Fault Information

Each I/O module provides indication when a fault occurs. The programming software displays this fault information:

In this location:	The software displays:								
Controller organizer	<p>The I/O configuration portion displays the modules configured for the controller. If the controller detects a fault with one of these modules, the controller organizer displays a yellow attention symbol () over the device and over the I/O Configuration folder.</p> <p>If the module is inhibited, the controller organizer displays an attention symbol () only over the device.</p>								
Connection tab from module properties	<p>The module fault field displays the fault code returned to the controller (related to the module) and the text detailing the fault.</p> <p>Common categories for module errors are:</p> <table> <tr> <td>Connection request error</td><td>The controller is attempting to make a connection to the module and has received an error. The connection was not made.</td></tr> <tr> <td>Service request error</td><td>The controller is attempting to request a service from the module and has received an error. The service was not performed successfully.</td></tr> <tr> <td>Module configuration rejected</td><td>The configuration in the module is invalid. This is commonly caused by two unmatched owners.</td></tr> <tr> <td>Module key mismatch</td><td>Electronic keying is enabled and some part of the keying information differs between the software and the module.</td></tr> </table>	Connection request error	The controller is attempting to make a connection to the module and has received an error. The connection was not made.	Service request error	The controller is attempting to request a service from the module and has received an error. The service was not performed successfully.	Module configuration rejected	The configuration in the module is invalid. This is commonly caused by two unmatched owners.	Module key mismatch	Electronic keying is enabled and some part of the keying information differs between the software and the module.
Connection request error	The controller is attempting to make a connection to the module and has received an error. The connection was not made.								
Service request error	The controller is attempting to request a service from the module and has received an error. The service was not performed successfully.								
Module configuration rejected	The configuration in the module is invalid. This is commonly caused by two unmatched owners.								
Module key mismatch	Electronic keying is enabled and some part of the keying information differs between the software and the module.								

Each I/O module has status bits that indicate when a fault occurs. Your logic should monitor these status bits. If any faults exist, your application should take appropriate action, such as shutting down the system in a controlled manner.

You can configure modules to generate a major fault in the controller if they lose their connection with the controller.



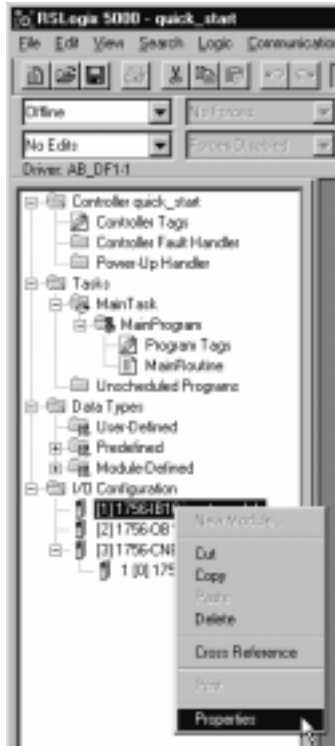
If you do not configure the major fault to occur, you should monitor the module status. If a module faults, outputs go to their configured faulted state. The controller and other I/O modules continue to operate based on old data from the faulted module.



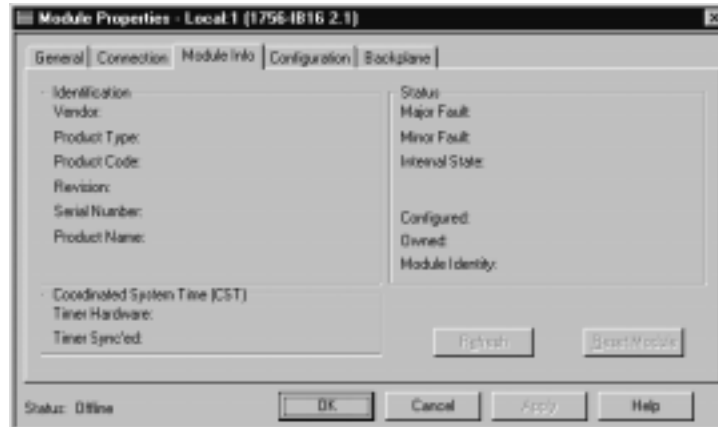
ATTENTION: Outputs respond to the “last, non-faulted” state of the controlling inputs. To avoid potential injury and damage to machinery, make sure this does not create unsafe operation. Configure critical I/O modules to generate a controller major fault when they lose their connections to the controller. Or monitor the status of I/O modules.

Using the programming software to view I/O faults

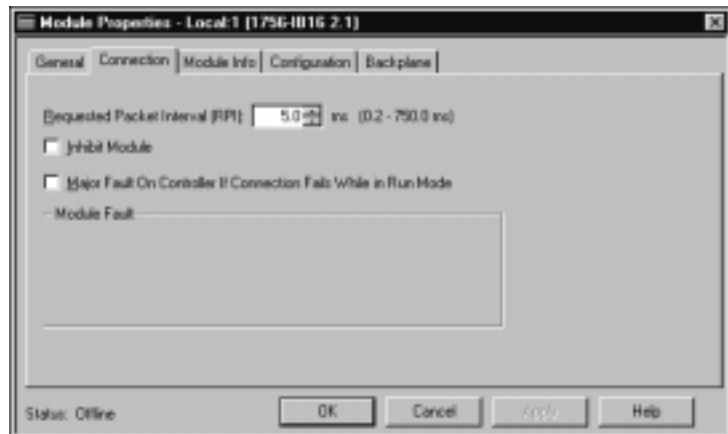
1. Select a module ("1756-IB16" in this example).
2. Click the right mouse button and select Properties.



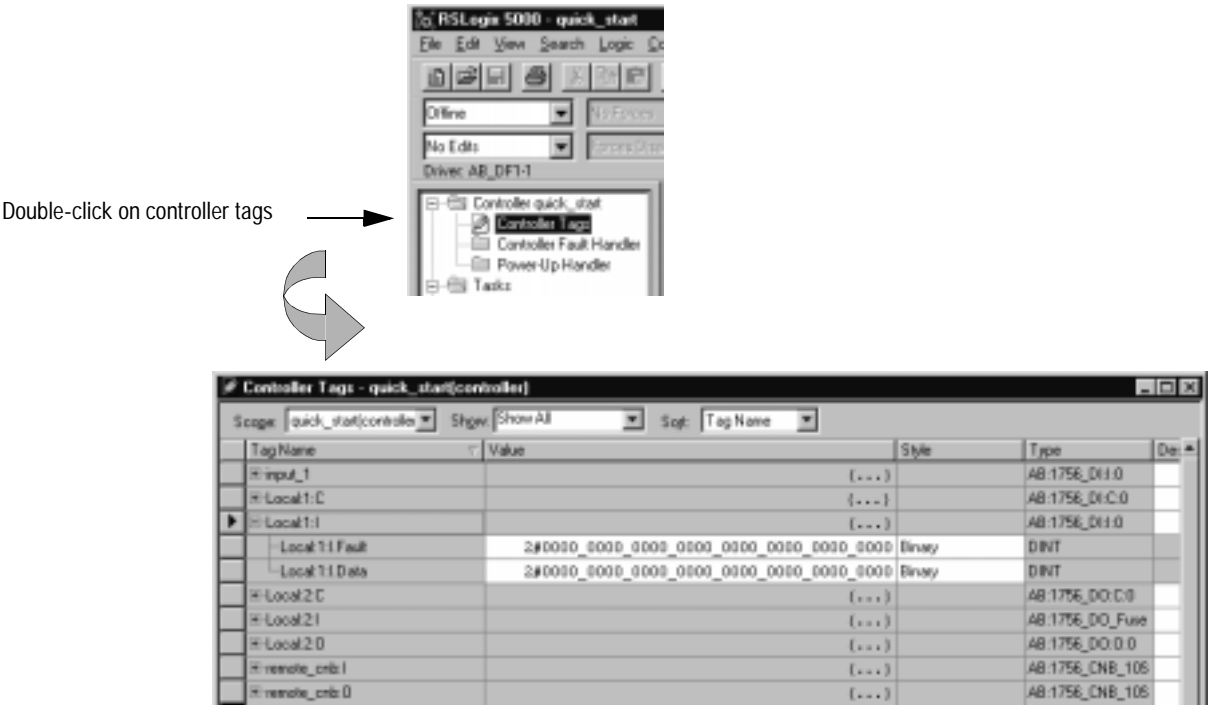
From the programming software, you can monitor the status of an I/O module. The programming software has a module information tab that displays module fault status and other information. You must be online to get actual data. This information is read from the actual module, so it's only available if the connection to the module is open. You can also reset the module from this tab.



You can also view I/O information from the connection tab. This information is read from the controller. Use this tab if the connection to the module is shut down.



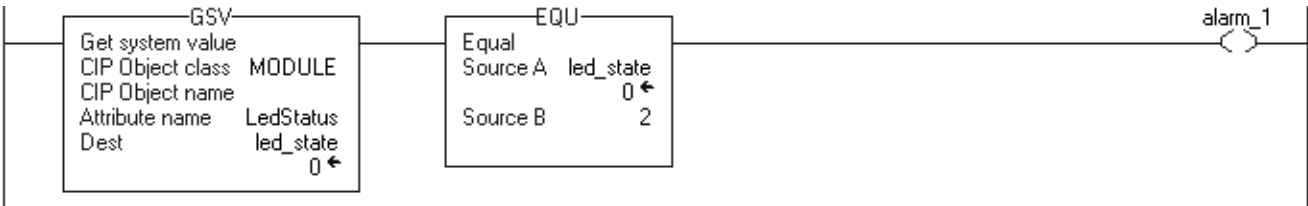
You can also monitor the status of the module I/O tags in the tag monitor. Most modules have status bits to indicate if data is being updated or if the module is inhibited.



Using logic to monitor I/O faults

You can also use logic to monitor I/O faults.

example 1



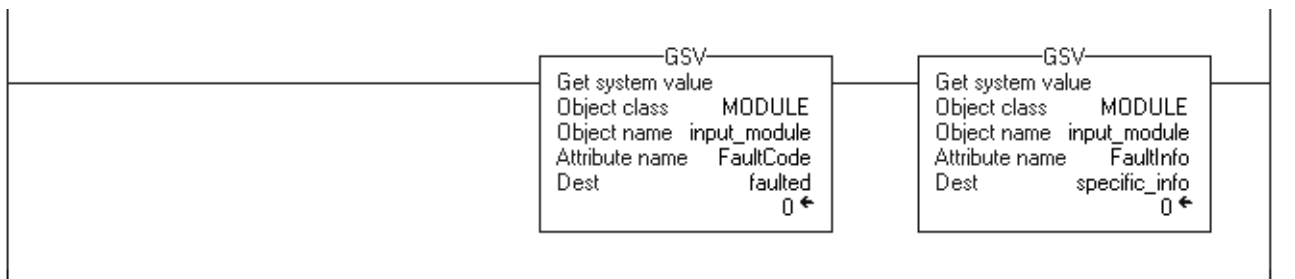
This example uses the MODULE object to determine whether any I/O module has lost its connection with the controller. If the controller I/O LED is flashing green, LedStatus = 2. The EQU example determines whether the I/O LED is flashing green.

example 2

This example uses the PROGRAM object to determine whether an I/O fault has occurred and was logged to the FAULTLOG. The *fault_record* structure is of this user-defined data type:

Member:	Data Type:	Style:	Value:	Description:
TimeStampLow	DINT	Decimal	xxxxxxx	time of fault (lower 32 bits)
TimeStampHigh	DINT	Decimal	xxxxxxx	time of fault (upper 32 bits)
Type	INT	Decimal	0003	I/O fault
Code	INT	Decimal	0016	I/O connection fault
ModulePort	DINT	Decimal	00000001	port 1, backplane
ModuleInstance	DINT	Decimal	xxxxxxx	module instance number
ErrorCode	DINT	Decimal	xxxxxxx	depends on module
ErrorInfo	DINT	Decimal	xxxxxxx	depends on module
spare1	DINT	Decimal;	00000000	unused
spare2	DINT	Decimal	00000000	unused
spare3	DINT	Decimal	00000000	unused
spare4	DINT	Decimal	00000000	unused

This would tell you the module that faulted. You can determine the ModuleInstance for a module by using a GSV: MODULE object name, module name, INSTANCE attribute.

example 3

This example uses the MODULE object to get a specific fault code and fault information from a specified module. In this case, you would already know which module to check.

For more information on handling faults, see chapter 11.

For more information on using the GSV instruction, see the *Logix5550 Instruction Set Reference Manual*, publication 1756-6.4.1.

Notes:

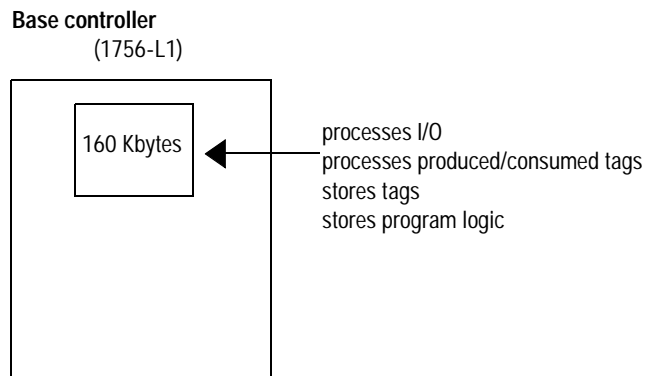
Organizing Data

Using This Chapter

For information about:	See page:
How the controller stores data	4-1
Creating tags	4-2
Using base tags	4-6
Using structures	4-9
Viewing an array as a collection of elements	4-13
Viewing an array as a block of memory	4-15
Aliasing tags	4-19
Scoping tags	4-20

How the Controller Stores Data

The Logix5550 controller memory stores both data and logic. There are 160 Kbytes of memory in the base controller.

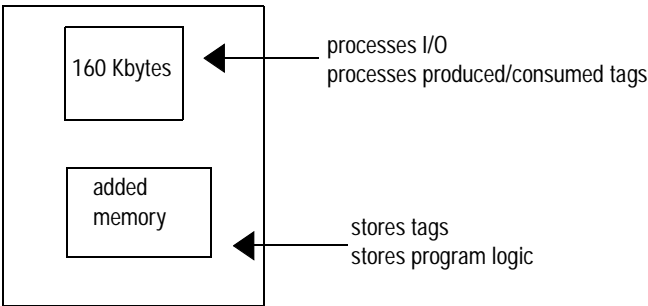


To increase memory capacity, you can add one 1756-Mx memory board. You can install a:

- 1756-M1 (512 Kbytes memory expansion board) **or**
- 1756-M2 (1 Mbytes memory expansion board) **or**
- 1756-M3 (2 Mbytes memory expansion board)

The memory expansion board changes how the controller stores data and logic. Once installed, the 160 Kbytes of memory of the base controller are dedicated to handling I/O and produced/consumed tags. The added memory is dedicated to logic and tag storage.

Controller with memory expansion board
(1756-L1Mx)



Creating Tags

The Logix5550 controller uses tags for storing and accessing data. A tag is similar to a variable, as used by programming languages. A tag has a name (that describes the data the tag stores) and a data type (that identifies the size and layout of data the tag can store).

The controller stores tags as you create them and as they fit into the controller memory. There are no pre-defined data tables, such as in PLC controllers. The Logix5550 controller uses its memory more efficiently by storing tags as needed and where they best fit in memory. Tags of similar data types are not necessarily grouped together in memory. If you want to group data, use an array.

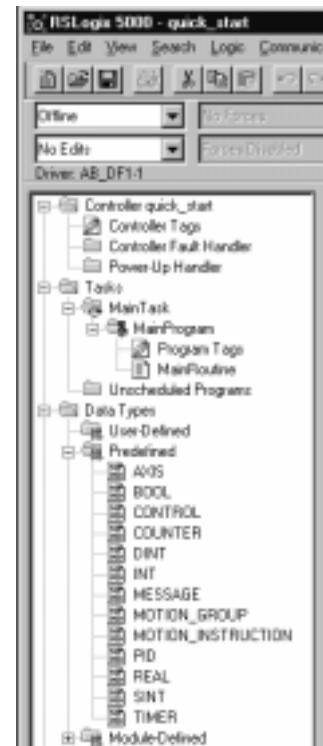
There are three types of tags.

Tag Type:	Description:
base	A base tag is a tag that actually defines the memory where a data element is stored.
alias	<p>An alias tag is a tag that references memory defined by another tag. An alias tag can refer to a base tag or another alias tag.</p> <p>Alias tags are useful for creating standardized programs that can be duplicated without having to readdress instructions. By using alias tags, each copy of the program can reference different base tags.</p>
consumed	A consumed tag is a tag whose data value comes from another controller.

Data types

When you develop a project, the controller provides a set of predefined data types.

predefined data types →



The controller data types follow the IEC 1131-3 defined data types. The predefined, atomic data types are:

Data type:	Description:	Range:
BOOL	1-bit boolean	0 = off 1 = on
SINT	1-byte integer	-128 to 127
INT	2-byte integer	-32,768 to 32,767
DINT	4-byte integer	-2,147,483,648 to 2,147,483,647
REAL	4-byte floating-point number	-3.402823E ³⁸ to -1.1754944E ⁻³⁸ (negative values) and 0 and 1.1754944E ⁻³⁸ to 3.402823E ³⁸ (positive values)

The REAL data type also stores ±infinity and ±NAN, but the software display differs based on the display format.

Display Format:	Equivalent:
Real	+infinite 1.\$ - infinite -1.\$ +NAN 1.#QNAN -NAN -1.#QNAN
Exponential	+infinite 1.#INF000e+000 - infinite -1.#INF000e+000 +NAN 1.#QNAN00e+000 -NAN -1.#QNAN00e+000

The predefined structures are:

Data type:	Description:
AXIS ¹	control structure for an axis
CONTROL	control structure for array (file) instructions
COUNTER	control structure for counter instructions
MESSAGE ¹	control structure for the MSG instruction
MOTION_GROUP ¹	control structure for a motion group
MOTION_INSTRUCTION	control structure for motion instructions
PID	control structure for the PID instruction
TIMER	control structure for timer instructions

1. These structures do not support arrays, cannot be nested in user-defined structures, and cannot be passed to other routines via a JSR instruction. These are controller-only tags.

Naming tags

Tag names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

Tag names are not case sensitive

You can also add descriptions to tags. Descriptions can have as many as 120 characters. You can use any printable character.

Entering tags

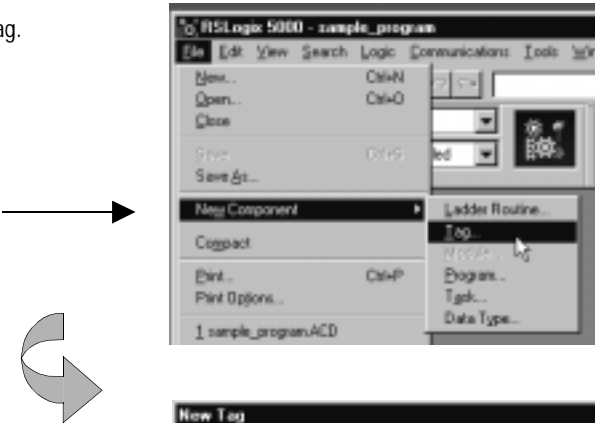
You use the programming software to create tags. You can either create a tag individually, or through the Edit Tags tab of the data monitor.

Whichever method you choose to create a tag, you define:

In this field:	Enter:
Name	Enter the name of the tag.
Description	Enter the description of the tag (optional).
Tag Type	Select one of these: Base normal tag Alias tag that references another tag or part of another tag Consumed tag whose value is produced by another controller
Data Type	Select the data type. The programming software displays a list of the available data types. The list consists of the predefined data types and any user-defined data types. If the tag is to be an array, specify the number of elements in each dimension. There can be as many as 3 dimensions. If the tag is not an array, or you do not want all 3 dimensions, set the dimension fields to zero (0).
Scope	Select the scope in which to create the tag. You can select controller scope or one of the existing program scopes.
Display Style	Select the display style of the tag. The programming software displays a list of the available styles, which depends on the data type. The style you select becomes the default display type when monitoring that tag with the programming software.
Produce this tag	Select whether to make this tag available to other controllers. Specify how many controllers can consume the tag.

To create a tag individually:

Select File → New Component → Tag.

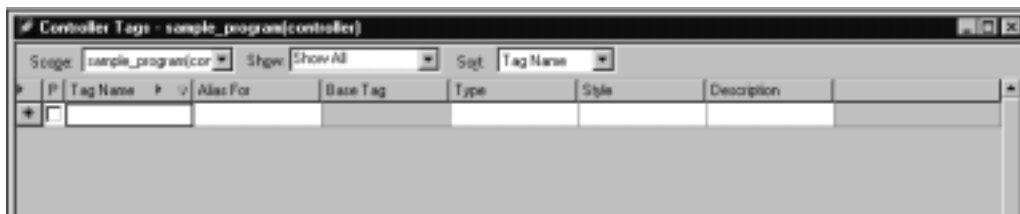


Define the tag.

The image shows a screenshot of the 'New Tag' dialog box. It contains fields for Name, Description, Tag Type (Base, Alias, Consumed), Data Type, Scope (sample_program(controller)), Style, and a checkbox for 'Produce this tag (tag is available to other controllers)'.

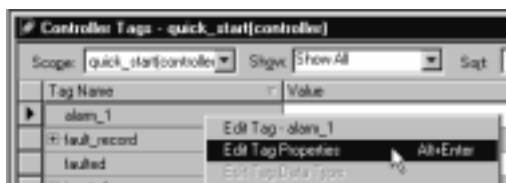
To create a tag from the Edit Tags tab of the data monitor:

1. Select controller tags or program tags.
2. Select Edit Tags.
3. Define the tag.

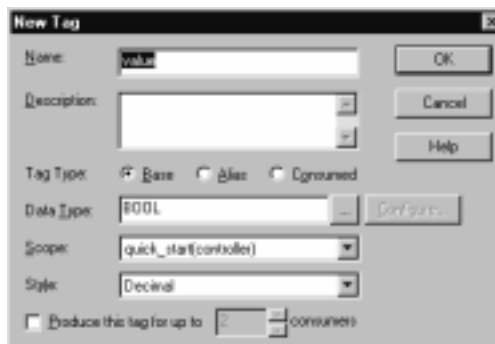
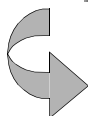


You can create tags before entering program logic or you can enter tag names as you enter logic and define the tags later using the New Tag dialog box.

1. Select the tag name in tag editor.
2. Right-click on the tag name.



4. Define the tag.



Using Base Tags

A base tag stores one value at a time. The type of value depends on the data type. Use these atomic data types to define base tags:

- BOOL
- SINT
- INT
- DINT
- REAL

Memory allocation for base tags

The amount of memory that a tag uses depends on the data type. The minimum allocation within the controller is four bytes.

Some of the data types are smaller than four bytes (BOOL, SINT, and INT). When you create a tag using one of these data types, the controller allocates four bytes, but the data only fills the part it needs. To use memory more efficiently, create arrays or structures to hold these smaller data types. (See the following examples.)

Most instructions do not operate on BOOL arrays. For BOOL data, it might be more efficient to create a structure instead of an array.

The following examples show memory allocation for base tags using the atomic data types:

`bool_value` as BOOL This example uses one bit of the data allocation.

Bit	31	1	0
allocation	not used		<code>bool_value</code>

`small_value` as SINT This example uses 8 bits of the data allocation.

Bit:	31	8	7	0
allocation	not used			<code>small_value</code>

`value` as INT This example uses 16 bits of the data allocation.

Bit:	31	16	15	0
allocation	not used			<code>value</code>

`big_value` as DINT This example uses all 32 bits of the data allocation.

Bit:	31	0
allocation	<code>big_value</code>	

`float_value` as REAL This example uses all 32 bits of the data allocation.

Bit:	31	0
allocation	<code>float_value</code>	

Data type conversions

If you mix data types for parameters within an instruction, some instructions automatically convert data to an optimal data type for that instruction. In some cases, the controller converts data to fit a new data type; in some cases the controller just fits the data as best it can.

Conversion:	Result:																								
large integer to small integer	<p>The controller truncates the upper portion of the larger integer and generates an overflow.</p> <p>For example:</p> <table><tr><td></td><td>Decimal</td><td>Binary</td></tr><tr><td>DINT</td><td>65,665</td><td>0000_0000_0000_0001_0000_0000_1000_0001</td></tr><tr><td>INT</td><td>129</td><td>0000_0000_1000_0001</td></tr><tr><td>SINT</td><td>-127</td><td>1000_0001</td></tr></table>		Decimal	Binary	DINT	65,665	0000_0000_0000_0001_0000_0000_1000_0001	INT	129	0000_0000_1000_0001	SINT	-127	1000_0001												
	Decimal	Binary																							
DINT	65,665	0000_0000_0000_0001_0000_0000_1000_0001																							
INT	129	0000_0000_1000_0001																							
SINT	-127	1000_0001																							
SINT or INT to REAL	No data precision is lost																								
DINT to REAL	Data precision could be lost. Both data types store data in 32 bits, but the REAL type uses some of its 32 bits to store the exponent value. If precision is lost, the controller takes it from the least-significant portion of the DINT.																								
REAL to SINT, INT, or DINT	<p>The controller rounds the fractional part and truncates the upper portion of the non-fractional part. If data is lost, the controller sets the overflow status flag.</p> <p>Rounding is to the nearest even number:</p> <table><tr><td>less than .5</td><td>round down</td></tr><tr><td>equal to .5</td><td>round to nearest even integer</td></tr><tr><td>greater than .5</td><td>round up</td></tr></table> <p>For example:</p> <table><tr><td>REAL (source)</td><td>DINT (result)</td></tr><tr><td>2.5</td><td>2</td></tr><tr><td>-2.5</td><td>-2</td></tr><tr><td>1.6</td><td>2</td></tr><tr><td>-1.6</td><td>-2</td></tr><tr><td>1.5</td><td>2</td></tr><tr><td>-1.5</td><td>-2</td></tr><tr><td>1.4</td><td>1</td></tr><tr><td>-1.4</td><td>-1</td></tr></table>	less than .5	round down	equal to .5	round to nearest even integer	greater than .5	round up	REAL (source)	DINT (result)	2.5	2	-2.5	-2	1.6	2	-1.6	-2	1.5	2	-1.5	-2	1.4	1	-1.4	-1
less than .5	round down																								
equal to .5	round to nearest even integer																								
greater than .5	round up																								
REAL (source)	DINT (result)																								
2.5	2																								
-2.5	-2																								
1.6	2																								
-1.6	-2																								
1.5	2																								
-1.5	-2																								
1.4	1																								
-1.4	-1																								

The controller cannot convert data to or from the BOOL data type.

Specifying bits

In addition to using BOOL-type tags to specify a bit, you can use bit specifiers within integer-type tags (SINT, INT, and DINT). The bit specifier identifies a bit within the tag. The bit-specifier range depends on the data type:

Data Type:	Bit Specifier Range:
SINT	0-7
INT	0-15
DINT	0-31

To specify a bit within an integer, specify:

x.*[y]*

where:

This	Is:
<i>x</i>	integer tag name
.	specifies that a bit reference follows
<i>[]</i>	encloses a bit reference only needed for a non-numeric bit reference
<i>y</i>	bit reference

The bit reference can be an immediate number, a tag, or an expression. You can use these operators to specify a bit:

Operator:	Description:
+	add
-	subtract/negate
*	multiply
/	divide
AND	AND
FRD	BCD to integer
NOT	complement
OR	OR
TOD	integer to BCD
SQR	square root
XOR	exclusive OR

For example:

Example:	Description:
<i>value.5</i> <i>value.[5]</i>	Both of these examples reference the 6 th bit in the integer <i>value</i> . When you use an immediate number you do not need the brackets.
<i>value.[another_value]</i>	This example references the bit identified by <i>another_value</i> within the integer <i>value</i> .
<i>value.[control.pos]</i>	This example references a bit identified by <i>control.pos</i> within the integer <i>value</i> .
<i>value.[control.pos - number + 5]</i>	This example uses an expression to identify a bit within the integer <i>value</i> .

Using Structures

A structure stores a group of data. Each member of the structure can be a different data type. The controller has its own predefined structures. Each I/O module has its own predefined structures. You can also create specialized user-defined structures, using any combination of predefined, atomic data types and most other structures.

For information about copying data to a structure, see the COP instruction in the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Predefined structures

The controller supports these predefined structures, each of which stores related information for specific instructions:

- AXIS¹
- CONTROL
- COUNTER
- MESSAGE¹
- MOTION_GROUP¹
- MOTION_INSTRUCTION
- PID
- TIMER

1. These structures do not support arrays and cannot be nested in user-defined structures.

For more information about the instructions that use these structures, see the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Module-defined structure

The Logix5550 controller automatically creates the I/O structures needed for each I/O module you configure for the system (see the previous chapter). These structures usually contain members for data, status information, and fault information.

User-defined structure

A user-defined structure groups different types of data into a single named entity. A user-defined structure contains one or more data definitions called members. Creating a member in a user-defined structure is just like creating an individual tag. The data type for each member determines the amount of memory allocated for the member. The data type for each member can be a/an:

- atomic data type
- predefined structure
- user-defined structure
- single dimension array of an atomic data type
- single dimension array of a predefined structure
- single dimension array of a user-defined structure

1. Select Data Types.
2. Click the right mouse button and select New Data Type.



Use the programming software to create your own structures.



For:	In this field:	Enter:
user-defined structure	Name	Enter the name of the structure.
	Description	Enter the description of the structure (optional).
each member of the structure	Name	Enter the name of the member.
	Data Type	Select the data type. The programming software displays a list of the available data types. The list consists of the predefined data types and any user-defined data types.
	Style	Select the display style of the member. The programming software displays a list of the available styles, which depends on the data type.
	Description	Enter the description of the member (optional).

You can create, edit, and delete user-defined structures only when programming offline.

If you modify a user-defined structure and change its size, the existing values of any tags that use the structure are set to zero (0).

Memory allocation for user-defined structures

The memory allocated for a user-defined structure depends on the data types for each member within the structure. Each member is allocated memory to start on an appropriate byte, INT, or DINT boundary. This is different than tags, which are always allocated as DINT. You can optimize memory by combining data as members within a structure.

For example:

User-Defined Structure Name:	Member:	Data Type:	Style:	Description:
Load_Info	Height	SINT	Decimal	load Height in feet
	Width	SINT	Decimal	load Width in feet
	Weight	REAL	Float	load Weight in pounds
	W_Flag	BOOL	Decimal	set true if load is stretch wrapped
	L_Flag	BOOL	Decimal	set true if load is labeled
Location	Source	INT	Decimal	system source location
	Destination	INT	Decimal	system destination location
	Info	Load_Info	none	load information structure

Memory is allocated in the order of the members.

Bit:	31	24	23	16	15	8	7	0
data allocation 1	Destination				Source			
data allocation 2	unused		unused		Width		Height	
data allocation 3	Weight							
data allocation 4	unused		unused		unused		bit 0 W_Flag bit 1 L_Flag	

Referencing members within a structure

You reference members in a structure by using the tag name and then the member name: *tag_name.member_name*

For example:

Example:	Description:
<i>timer_1.pre</i>	This example references the .PRE value of the <i>timer_1</i> structure.
<i>input_load</i> as data type <i>load_info</i> <i>input_load.height</i>	This examples references the <i>Height</i> member of the user-defined <i>input_load</i> structure

If the structure is embedded in another structure, use the tag name of the structure at the highest level followed by a substructure tag name and member name: *tag_name.substructure_name.member_name*

For example:

Example:	Description:
<i>input_location</i> as data type <i>location</i> <i>input_location.load_info.height</i>	This example references the <i>height</i> member of the <i>load_info</i> structure in the <i>input_location</i> structure.

If the structure defines an array, use the array tag, followed by the position in the array and any substructure and member names.

array_tag[position].member

or

array_tag[position].substructure_name.member_name

For example:

Example:	Description:
<i>conveyor</i> as array <i>location[100]</i>	This specifies a 100 word array. Each element in the array is data type <i>location</i> (a structure).
<i>conveyor[10].source</i>	This example references the <i>source</i> member of the 11 th element in the array (array elements are zero based).
<i>conveyor[10].info.height</i>	This example references the <i>height</i> member of the <i>info</i> structure in the 11 th element of the array (array elements are zero based).

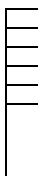
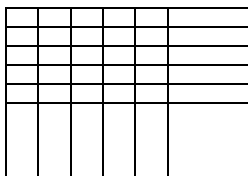
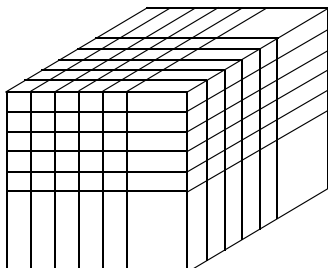
Viewing an Array as a Collection of Elements

Arrays let you group a set of data (of the same data type) by the same name and to use subscripts to identify individual elements. An element in an array can be an atomic data type or a structure.

You specify an element in an array by its subscript(s). Enter the array tag name followed by the subscript(s) in square brackets. The subscript(s) must specify a value for each dimension of the array. Dimensions are zero-based.

For this array:	Specify:
one dimension	<i>array_name[subscript_0]</i>
two dimension	<i>array_name[subscript_0, subscript_1]</i>
three dimension	<i>array_name[subscript_0, subscript_1, subscript_2]</i>

An array can have as many as three dimensions. The total number of elements in an array is the product of each dimension's size.

This array:	Stores data like:	For example:																				
one dimension		<table><tr><th>Tag name:</th><th>Type</th><th>Dimension 0</th><th>Dimension 1</th><th>Dimension 2</th></tr><tr><td><i>one_d_array</i></td><td>DINT[7]</td><td>7</td><td>--</td><td>--</td></tr><tr><td colspan="5">total number of elements = 7</td></tr><tr><td colspan="5">valid subscript range DINT[x] where x=0–6</td></tr></table>	Tag name:	Type	Dimension 0	Dimension 1	Dimension 2	<i>one_d_array</i>	DINT[7]	7	--	--	total number of elements = 7					valid subscript range DINT[x] where x=0–6				
Tag name:	Type	Dimension 0	Dimension 1	Dimension 2																		
<i>one_d_array</i>	DINT[7]	7	--	--																		
total number of elements = 7																						
valid subscript range DINT[x] where x=0–6																						
two dimension		<table><tr><th>Tag name:</th><th>Type</th><th>Dimension 0</th><th>Dimension 1</th><th>Dimension 2</th></tr><tr><td><i>two_d_array</i></td><td>DINT[4,5]</td><td>4</td><td>5</td><td>--</td></tr><tr><td colspan="5">total number of elements = 4 * 5 = 20</td></tr><tr><td colspan="5">valid subscript range DINT[x,y] where x=0–3; y=0–4</td></tr></table>	Tag name:	Type	Dimension 0	Dimension 1	Dimension 2	<i>two_d_array</i>	DINT[4,5]	4	5	--	total number of elements = 4 * 5 = 20					valid subscript range DINT[x,y] where x=0–3; y=0–4				
Tag name:	Type	Dimension 0	Dimension 1	Dimension 2																		
<i>two_d_array</i>	DINT[4,5]	4	5	--																		
total number of elements = 4 * 5 = 20																						
valid subscript range DINT[x,y] where x=0–3; y=0–4																						
three dimension		<table><tr><th>Tag name:</th><th>Type</th><th>Dimension 0</th><th>Dimension 1</th><th>Dimension 2</th></tr><tr><td><i>three_d_array</i></td><td>DINT[2,3,4]</td><td>2</td><td>3</td><td>4</td></tr><tr><td colspan="5">total number of elements = 2 * 3 * 4 = 24</td></tr><tr><td colspan="5">valid subscript range DINT[x,y,z] where x=0–1; y=0–2, z=0–3</td></tr></table>	Tag name:	Type	Dimension 0	Dimension 1	Dimension 2	<i>three_d_array</i>	DINT[2,3,4]	2	3	4	total number of elements = 2 * 3 * 4 = 24					valid subscript range DINT[x,y,z] where x=0–1; y=0–2, z=0–3				
Tag name:	Type	Dimension 0	Dimension 1	Dimension 2																		
<i>three_d_array</i>	DINT[2,3,4]	2	3	4																		
total number of elements = 2 * 3 * 4 = 24																						
valid subscript range DINT[x,y,z] where x=0–1; y=0–2, z=0–3																						

Indexing through arrays

To dynamically change the array element that your logic references, use tag or expression as the subscript to point to the element. This is similar to indirect addressing in PLC-5 logic. You can use these operators in an expression to specify an array subscript:

Operator:	Description:
+	add
-	subtract/negate
*	multiply
/	divide
AND	AND
FRD	BCD to integer
NOT	complement
OR	OR
TOD	integer to BCD
SQR	square root
XOR	exclusive OR

For example:

Definitions:		Example:	Description:
<i>my_list</i>	defined as DINT[10]	<i>my_list[5]</i>	This example references element 5 in the array. The reference is static because the subscript value remains constant.
<i>my_list</i>	defined as DINT[10]	MOV the value 5 into <i>position</i>	This example references element 5 in the array. The reference is dynamic because the logic can change the subscript by changing the value of <i>position</i> .
<i>position</i>	defined as DINT	<i>my_list[position]</i>	
<i>my_list</i>	defined as DINT[10]	MOV the value 2 into <i>position</i>	This example references element 7 (2+5) in the array. The reference is dynamic because the logic can change the subscript by changing the value of <i>position</i> or <i>offset</i> .
<i>position</i>	defined as DINT	MOV the value 5 into <i>offset</i>	
<i>offset</i>	defined as DINT	<i>my_list[position+offset]</i>	

Make sure any array subscript you enter is within the boundaries of the specified array. Instructions that view arrays as a collection of elements generate a major fault (type 4, code 20) if a subscript exceeds its corresponding dimension.

Specifying Bits Within Arrays

You can address bits within elements of arrays. For example:

Definitions:		Example:	Description:
<i>array1</i>	defined as DINT[5]	<i>array1[1].2</i>	This example references the bit 2 in element 1 of the array.
<i>array2</i>	defined as INT[17,36] 1st dimension has 17 elements 2nd dimension has 36 elements	<i>array2[3,4].15</i>	This example references the bit 15 of the element <i>array2[3,4]</i> .
<i>array3</i>	defined as SINT[2,4,6] 1st dimension holds 2 elements 2nd dimension holds 4 elements 3rd dimension holds 6 elements	<i>array3[1,3,2].4</i>	This example references bit 4 of the element <i>array3[1,3,2]</i> .
<i>MyArray</i>	defined as SINT[100]	<i>MyArray[(MyIndex AND NOT 7) / 8].[MyIndex AND 7]</i>	This example references a bit within an SINT array.
<i>MyIndex</i>	defined as SINT		
<i>MyArray</i>	defined as INT[100]	<i>MyArray[(MyIndex AND NOT 15) / 16].[MyIndex AND 15]</i>	This example references a bit within an INT array.
<i>MyIndex</i>	defined as INT		
<i>MyArray</i>	defined as DINT[100]	<i>MyArray[(MyIndex AND NOT 31) / 32].[MyIndex AND 31]</i>	This example references a bit within an DINT array.
<i>MyIndex</i>	defined as DINT		

You can also use the operators shown in the table on page 4-14 to specify bits.

Viewing an Array as a Block of Memory

The data in an array is stored contiguously in memory. The file (array) instructions typically require a starting address within an array and a length, which determines which elements and how many elements the instruction reads or writes.

These instructions manipulate array data as a contiguous block of memory (the remaining instructions manipulate array data as individual elements):

BSL	FLL
BSR	LFL
COP	LFU
DDT	SQI
FBC	SQL
FFL	SQO
FFU	

Important: If an instruction attempts to read data beyond the end of an array, the instruction reads whatever data happens to be there and processes it as if it were valid data (no error occurs). If an instruction attempts to write data beyond the end of an array, a major fault occurs (type 4, code 20).

How the controller stores array data

The following table shows the sequential order of the elements in the examples on page 4-13.

One-Dimensional Array Elements (ascending order):	Two-Dimensional Array Elements (ascending order):	Three-Dimensional Array Elements (ascending order):
<i>one_d_array[0]</i>	<i>two_d_array[0,0]</i>	<i>three_d_array[0,0,0]</i>
<i>one_d_array[1]</i>	<i>two_d_array[0,1]</i>	<i>three_d_array[0,0,1]</i>
<i>one_d_array[2]</i>	<i>two_d_array[0,2]</i>	<i>three_d_array[0,0,2]</i>
<i>one_d_array[3]</i>	<i>two_d_array[0,3]</i>	<i>three_d_array[0,0,3]</i>
<i>one_d_array[4]</i>	<i>two_d_array[0,4]</i>	<i>three_d_array[0,1,0]</i>
<i>one_d_array[5]</i>	<i>two_d_array[1,0]</i>	<i>three_d_array[0,1,1]</i>
<i>one_d_array[6]</i>	<i>two_d_array[1,1]</i>	<i>three_d_array[0,1,2]</i>
	<i>two_d_array[1,2]</i>	<i>three_d_array[0,1,3]</i>
	<i>two_d_array[1,3]</i>	<i>three_d_array[0,2,0]</i>
	<i>two_d_array[1,4]</i>	<i>three_d_array[0,2,1]</i>
	<i>two_d_array[2,0]</i>	<i>three_d_array[0,2,2]</i>
	<i>two_d_array[2,1]</i>	<i>three_d_array[0,2,3]</i>
	<i>two_d_array[2,2]</i>	<i>three_d_array[1,0,0]</i>
	<i>two_d_array[2,3]</i>	<i>three_d_array[1,0,1]</i>
	<i>two_d_array[2,4]</i>	<i>three_d_array[1,0,2]</i>
	<i>two_d_array[3,0]</i>	<i>three_d_array[1,0,3]</i>
	<i>two_d_array[3,1]</i>	<i>three_d_array[1,1,0]</i>
	<i>two_d_array[3,2]</i>	<i>three_d_array[1,1,1]</i>
	<i>two_d_array[3,3]</i>	<i>three_d_array[1,1,2]</i>
	<i>two_d_array[3,4]</i>	<i>three_d_array[1,1,3]</i>
		<i>three_d_array[1,2,0]</i>
		<i>three_d_array[1,2,1]</i>
		<i>three_d_array[1,2,2]</i>
		<i>three_d_array[1,2,3]</i>
For an array with only one dimension, <i>tag_name[subscript_0]</i> , <i>subscript_0</i> increments to its maximum value.	For an array with two dimensions, <i>tag_name[subscript_0,subscript_1]</i> , <i>subscript_0</i> is held fixed at 0 while <i>subscript_1</i> increments from 0 to its maximum value. <i>Subscript_0</i> then increments by 1 (if dimension 0 is greater than 1) and is held fixed while <i>subscript_1</i> increments through its range again. This pattern continues until both subscripts reach their maximum values.	For an array with three dimensions, <i>tag_name[subscript_0, subscript_1,</i> <i>subscript_2]</i> , <i>subscript_0</i> is held fixed at 0 while <i>subscript_1</i> and <i>subscript_2</i> increment just like a two- dimensional array. <i>Subscript_0</i> then increments by 1 (if dimension 0 is greater than 1) and is held fixed until <i>subscript_1</i> and <i>subscript_2</i> reach their maximum values. This pattern continues until all three subscripts reach their maximum values.

Varying a dimension

The AVE, SRT, and STD instructions have a Dimension to vary operand. The instruction uses this operand to calculate an offset that the instruction uses to determine which elements of the Array to read or write.

Array:	Dimension to vary:	Offset:
one dimension	0	1
two dimension	0	dimension_1
	1	1
three dimension	0	$(dimension_1) * (dimension_2)$
	1	dimension_2
	2	1

Memory Allocation for Arrays

The amount of memory that an array uses depends on the data type used to create the array. The minimum allocation within the controller is four bytes, which is the same as 32 BOOLs, 4 SINTs, 2 INTs, or 1 DINT.

The following examples show memory allocation for various arrays:

bit_values as BOOL[32]

This example is an array with 32 bit elements, each of data type BOOL (1 bit per element).

Bit:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
data allocation 1	[31]	[30]	[29]	[28]	[27]	[26]	[25]	[24]	[23]	[22]	[21]	[20]	[19]	[18]	[17]	[16]
Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data allocation 1 <i>continued</i>	[15]	[14]	[13]	[12]	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]

small_values as SINT[8]

This example is an array with 8 elements, each of data type SINT (1 byte per element).

Bit:	31	24	23	16	15	8	7	0
data allocation 1	small_values[3]			small_values[2]		small_values[1]		small_values[0]
data allocation 2	small_values[7]			small_values[6]		small_values[5]		small_values[4]

small_values as SINT[3]

This example is an array with 3 elements, each of data type SINT (1 byte per element). Because the minimum data allocation is 4 bytes, the last byte is zero.

Bit:	31	24	23	16	15	8	7	0
data allocation 1	0			small_values[2]		small_values[1]		small_values[0]

values as INT[4] This example is an array with 4 elements, each of data type INT (2 bytes per element).

Bit:	31	16	15	0
data allocation 1	values[1]		values[0]	
data allocation 2	values[3]		values[2]	

big_values as DINT[2] This example is an array with 2 elements, each of type DINT (4 bytes per element).

Bit:	31	0
data allocation 1	big_values[0]	
data allocation 2	big_values[1]	

timer_list as TIMER[2] This example is an array with 2 elements, each element is a TIMER structure (12 bytes per structure).

Bit:	31	0
data allocation 1	timer_list[0] status bits	
data allocation 2	timer_list[0].pre	
data allocation 3	timer_list[0].acc	
data allocation 4	timer_list[1] status bits	
data allocation 5	timer_list[1].pre	
data allocation 6	timer_list[1].acc	

small_values as SINT[2,2,2] This example is a three-dimensional array with 8 elements, each of data type SINT.

Bit:	31	24	23	16	15	8	7	0
data allocation 1	small_values[0,1,1]		small_values[0,1,0]		small_values[0,0,1]		small_values[0,0,0]	
data allocation 2	small_values[1,1,1]		small_values[1,1,0]		small_values[1,0,1]		small_values[1,0,0]	

big_values as DINT[2,2,2] This example is a three-dimensional array with 8 elements, each of type DINT.

Bit:	31	0
data allocation 1	big_values[0,0,0]	
data allocation 2	big_values[0,0,1]	
data allocation 3	big_values[0,1,0]	
data allocation 4	big_values[0,1,1]	
data allocation 5	big_values[1,0,0]	
data allocation 6	big_values[1,0,1]	
data allocation 7	big_values[1,1,0]	
data allocation 8	big_values[1,1,1]	

You can modify array dimensions when programming offline without loss of tag data. You cannot modify array dimensions when programming online.

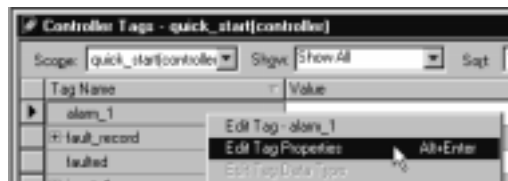
Aliasing Tags

A tag alias lets you create one tag that represents another tag. This is useful for defining simplified tag names for elements of structures or arrays. For example:

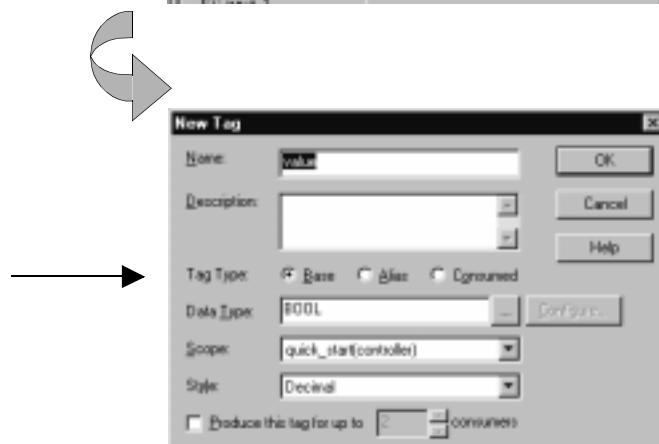
Definitions:	Alias Example:	Description:
<i>mylist[10]</i> array of 10 DINT elements	<i>half = my_list[5]</i>	This example uses the tag <i>half</i> as an alias to <i>my_list[5]</i> .
<i>cookies[5]</i> array of 5 recipe elements data type <i>recipe</i> <i>flour</i> as data type REAL <i>sugar</i> as data type REAL <i>timer</i> as array <i>timer[5]</i>	<i>oatmeal = cookies[1]</i> <i>oatmeal_flour = cookies[1].flour</i> <i>oatmeal_preset = cookies[1].timer[2].pre</i>	This example uses tags referring to <i>oatmeal</i> as aliases for different elements of the structure in the first element of array <i>cookies</i> .
I/O structures	<i>light_1 = local:0:I.Data.0</i> <i>motor_1 = local:1:O.Data.0</i>	This example uses simpler tags to refer to specific I/O points.
input point <i>local:0:I.Data.0</i>		
output point <i>local:1:O.Data.0</i>		

You can use the Tag Editor to create an alias or you can enter the alias tag as you enter logic and define the alias later using the New Tag dialog box.

1. Select the tag name in tag editor.
2. Right-click on the tag name.



3. Define the tag.
4. Click Alias.



To create an alias using the New Tag dialog box, you define the tag and select the alias tag type:

In this field:	Enter:
Name	Enter the name of the tag. This is the alias name.
Description	Enter the description of the tag (optional).
Tag Type	Select: Alias tag that references another tag with the same characteristics
Refers To	Enter the name of the tag that you are representing by the alias name. The programming software displays a list of the available tags you can reference.
Data Type	This field is automatically selected. Displays the data type of the resulting alias tag. This is based on the tag you select for the Refers To field (described above). You cannot specify array dimensions for an alias tag.
Scope	This field is automatically selected. Select the scope in which to create the tag. You can select controller scope or one of the existing programs.
Display Style	This field is automatically selected. Select the display style of the tag. The programming software displays a list of the available styles, which depends on the data type.
Produce this tag	Select whether to make this tag available to other controllers through controller-to-controller messaging. Specify how many controllers can consume the tag. The tag must be a controller-scoped tag. You can only choose to produce a tag when programming offline.

Scoping Tags

You can group (scope) tags within an individual program or make them available to instructions anywhere in the controller. When you define tags, you specify them as either program tags (local) or controller tags (global).

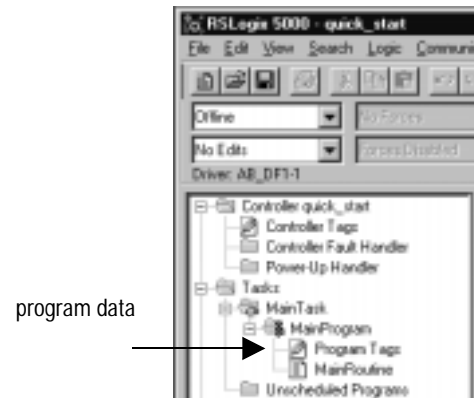
You can have multiple tags with the same name, as long as each tag has a different scope (one as a controller tag and the others as program tags; or all as program tags). Certain limitations apply to data in both scopes.

Action:	Controller Scope Tags	Program Scope Tags
referencing the tag	any routine	routines within the same program
naming a tag	within the current program, a controller scope tag is not available if a program scope tag of the same name exists you cannot reference both a controller scope tag and a program scope tag with the same name in a routine	defaults to the program scope tag
messaging	no limitation tag must be controller scope	tag cannot be program scope
producer/consumer	no limitation tag must be controller scope	tag cannot be program scope

Scoping tags local to a program

Program tags consist of data that is used exclusively by the routines within a program. These tags are local to a program. The routines in other programs cannot access the program tags of another program.

Tag scope:	Description:
program	Program tags are data that is used exclusively by the routines within a program. Other programs cannot access this data.



Scoping tags global to a controller

Controller tags consist of data that is available to all the routines within a controller, regardless of what tasks or programs contain the routines. These tags are global to the controller.

Tag scope:	Description:
controller	Controller tags are data that is available to all the tasks, programs, and routines within a controller project.



All I/O tags are created as controller-scoped tags when you create the module for the controller.

Notes:

Developing Programs

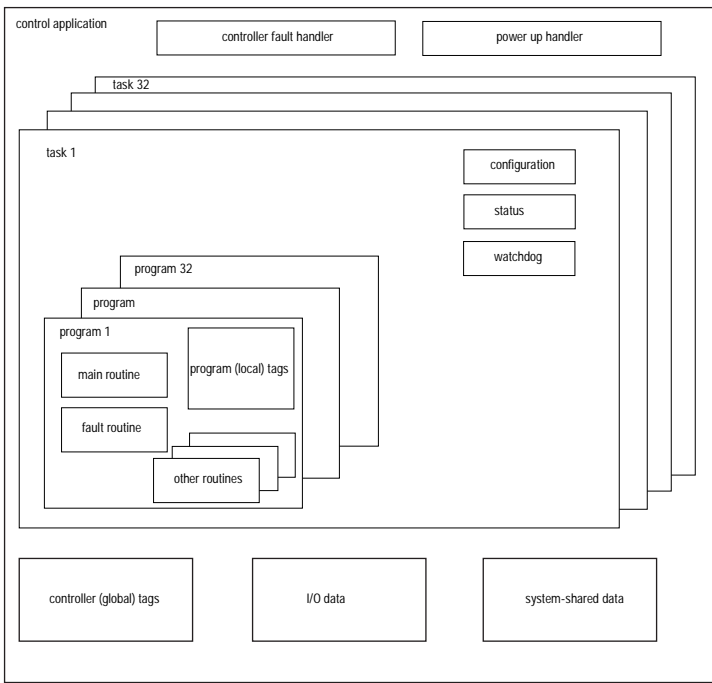
Using This Chapter

For information about:	See page:
Organizing projects	5-1
Defining tasks	5-2
Defining programs	5-8
Defining routines	5-11
Entering ladder logic	5-13
Scheduling system overhead	5-15
Downloading a program	5-16

Organizing Projects

The controller operating system 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



The operating system is preemptive in that it provides the ability to interrupt an executing task, switch control to a different task, and then return control back to the original task, once the interrupting tasks completes its execution. The controller is single-threaded in that only one task can be executing at one time. In any given task, only one program is executing at one time.

Defining Tasks

A task provides scheduling and priority information for a set of one or more programs that execute based on specific criteria. You can configure tasks as either continuous or periodic.

Task Type:	Number Supported by the Logix5550 Controller:
continuous	1
periodic	31 if there is a continuous task 32 if there is no continuous task

Each task in the controller has a priority level. The operating system uses the priority level to determine which task to execute when multiple tasks are triggered. There are 15 configurable priority levels for periodic tasks that range from 1-15, with 1 being the highest priority and 15 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 a periodic task.

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.

Each task has a watchdog timer that monitors the execution of a task. The watchdog timer begins to time when the task is initiated and stops when all the programs within the task have executed.



ATTENTION: If the watchdog timer reaches a configurable preset, a major fault occurs. Depending on the controller fault handler, the controller might shut down.

Programs within a task access input and output data directly from controller-scoped memory. Logic within any task can modify controller-scoped data. Data and I/O values are asynchronous and can change during the course of a task's execution. An input value referenced at the beginning of a task's execution can be different when referenced later.



ATTENTION: Take care to ensure that data memory contains the appropriate values throughout a task's execution. You can duplicate or buffer data at the beginning of the scan to provide reference values for your logic.

Using a continuous task

A continuous task operates in a self-triggered mode. It restarts itself after each completion. You can create one continuous task for the controller. The continuous task operates as the lowest priority task in the controller (one priority level lower than the lowest periodic task). This means that all periodic tasks will interrupt the continuous task.

The continuous task is a background task because any CPU time not allocated to other operations (such as motion, communications, and periodic tasks) is used to execute the programs within the continuous task.

When you create a project, the default MainTask is a continuous task. You can leave this task as it is, or you can change its characteristics

Using a periodic task

A periodic task, also known as a selectable timed interrupt (STI), is triggered by the operating system at a repetitive period of time. This type of task is useful for projects that require accurate or deterministic execution. Periodic tasks always interrupt the continuous task. Depending on the priority level, a periodic task may interrupt other periodic tasks in the controller.

Use the programming software to configure the time period from 1 msec to 2000 seconds. The default is 10 msec.



ATTENTION: Ensure that the time period is longer than the sum of the execution times of all the programs assigned to the task. If the controller detects that a periodic task trigger occurs for a task that is already operating, a major fault occurs.

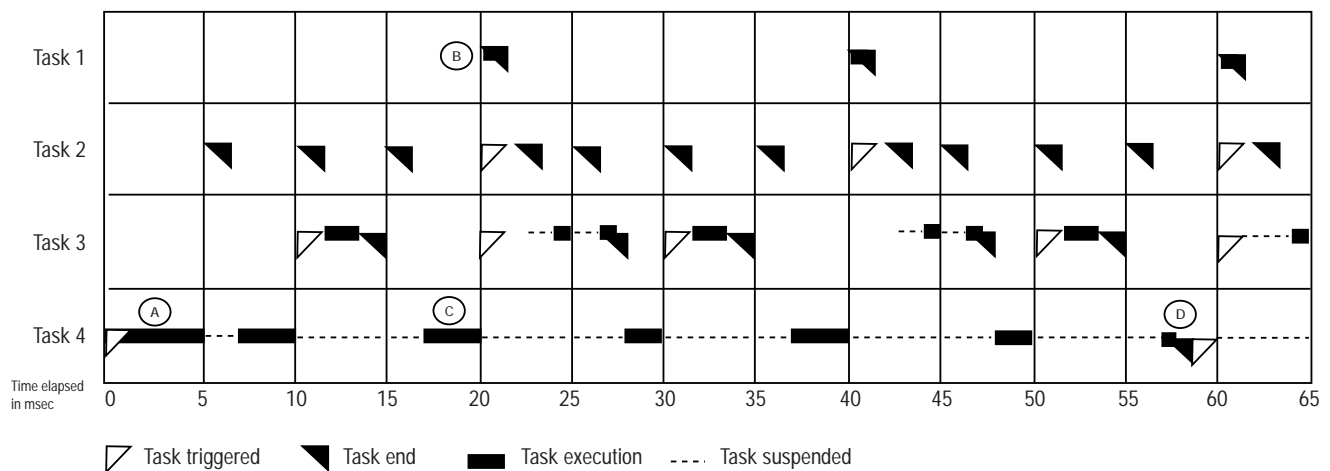
When working with multiple periodic tasks, make sure that sufficient CPU time is made available to handle task interruption.

Periodic tasks at the same priority execute on a time-slice basis at 1ms intervals.

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

Example:

Task:	Priority Level:	Task Type:	Actual Execution Time:	Worst Case Execution Time:
1	5	20ms periodic	2ms	2ms
2	10	5 ms periodic	1ms	3ms
3	15	10ms periodic	4ms	8ms
4	none (lowest)	continuous	24ms	80ms



Notes:

- A.** The continuous task runs at the lowest priority and is interrupted by all other tasks.
- B.** The highest priority task interrupts all lower priority tasks.
- C.** A lower priority task can be interrupted multiple times by a higher priority task.
- D.** When the continuous task completes a full scan it restarts immediately, unless a higher priority task is running.



ATTENTION: The rate that a periodic task is triggered determines the period by which the logic is executed and the data is manipulated within the task. Data and outputs established by the programs in a task retain their values until the next execution of the task or they are manipulated by another task.

Creating tasks

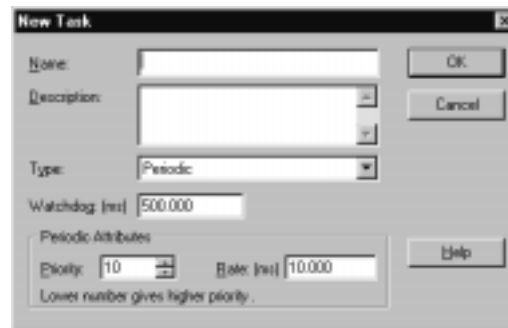
When you open a new controller project in your programming software, the MainTask is already defined as a continuous task. You can change this default task to fit your project.

The default task is MainTask. →



To create a new task:

1. Select Tasks.
2. Click the right mouse button and select New Task.



In this field:	Enter:
Name	Enter the name of the task.
Description	Enter a description of the task (optional).
Type	Select Continuous or Periodic. The controller supports only 1 continuous task. The remaining tasks must be periodic.
Watchdog	Enter the time in msec for the watchdog timer. If any program scheduled for a task takes too long to scan, or is interrupted by a higher-priority task, causing the total time to execute the task to exceed the watchdog timer value, the controller generates a major fault. The default watchdog timer is 500 msec.
Priority	If you defined a periodic task, specify the priority of the task by entering a number from 1 to 15. The lower the number, the higher the priority. The number 1 is the highest priority; the number 15 is the lowest priority.
Rate (ms)	If you defined a periodic task, enter the rate (in msec) at which the controller executes the task. The valid range is 1 msec to 2,000,000 msec (2000 seconds).

Naming tasks

Task names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

Task names are not case sensitive.

You can also add descriptions to tasks. Descriptions can have as many as 128 characters. You can use any printable character.

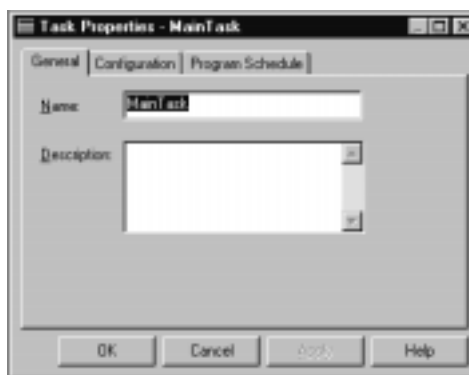
Configuring tasks

Once you create a task, there are other properties that you need to configure, such as how the programs within the task execute. You can prioritize the tasks up to 15 levels.

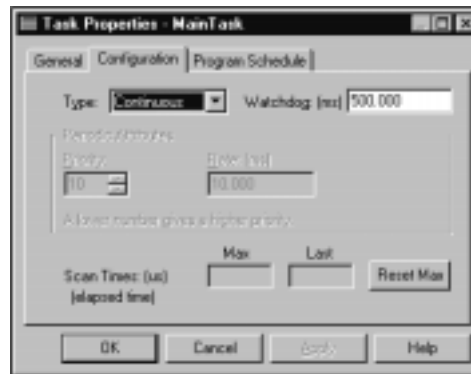
- The higher the number, the lower the priority (15 is the lowest priority you can select for a periodic task).
- The continuous task has a non-selectable priority that is one lower than the lowest, configured periodic-task priority.
- A task at a higher priority (such as 1) preempts one at a lower priority (such as 15).
- Tasks at the same priority execute on a time-slice basis at 1ms intervals.
- Periodic tasks always interrupt the continuous task.

To configure an existing task:

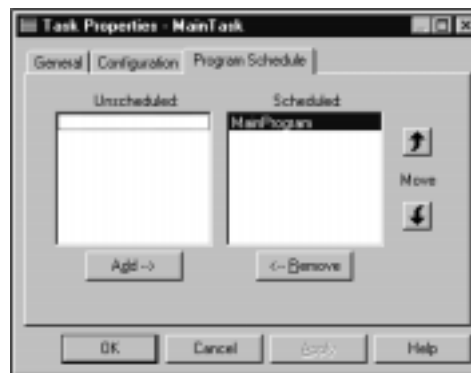
1. Select a task ("MainTask" in this example).
2. Click the right mouse button and select Properties.



On this tab:	In this field:	Enter:
General	Name	The programming software displays the current name of the task. Edit the name, if necessary.
	Description	The programming software displays the current description. Edit the description, if necessary.



On this tab:	In this field:	Enter:
Configuration	Type	The programming software displays the current type. Select another type, if necessary. The controller supports only 1 continuous task. The remaining tasks must be periodic.
	Watchdog (ms)	Specify a watchdog timeout for the task. The valid range is 1 msec to 2,000,000 msec (2000 seconds). The default is 500 msecs.
	Priority	If you defined a periodic task, specify the priority of the task by entering a number from 1 to 15. The lower the number, the higher the priority. The number 1 is the highest priority; the number 15 is the lowest priority.
	Rate (ms)	If you defined a periodic task, enter the rate (in msec) at which the controller executes the task. The valid range is 1 msec to 2,000,000 msec (2000 seconds). The default is 10 msecs.
	Scan Time (μs)	While online, the programming software displays the maximum scan time and the last scan time in μsec for the current task. These values are elapsed time, which includes any time spent waiting for higher-priority tasks. These values are display only.



On this tab:	In this field:	Enter:
Program Schedule	Unscheduled	The programming software displays the programs that have not been scheduled by a task.
	Scheduled	Add or remove programs from this list to create a list of programs associated with the current task. The task executes programs to completion, in order from the top of the list to the bottom of the list.

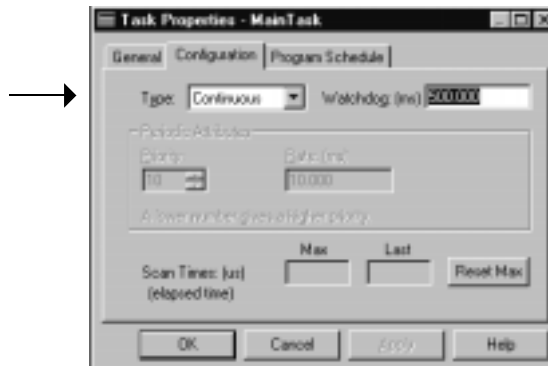
1. Select a task ("MainTask" in this example).
2. Click the right mouse button and select Properties.



Setting the task watchdog

Each task has its own watchdog timer. If all the programs scheduled for a task take too long to scan, or are interrupted by higher-priority tasks, and exceed the watchdog timer value, the controller executes the fault routine, if one exists, for the program that was executing when the watchdog expired. You can change the watchdog timer by using the programming software via the configuration tab of the task properties.

To change the task watchdog timer:



Avoiding periodic task overlap

Make sure the watchdog timer is greater than the time it takes to execute all the programs in the task. A watchdog timeout fault (major fault) occurs if a task is executing and it is triggered again. This can happen if a lower-priority task is interrupted by a higher-priority task, delaying completion of the lower-priority task.

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.

Creating programs

When you open a new controller project in your programming software, the MainProgram is already defined for the MainTask. You can modify this program, as well as add other programs.

To create a new program:

1. Select a task ("MainTask" in this example).
2. Click the right mouse button and select New Program.



In this field:	Enter:
Name	Enter the name of the program.
Description	Enter a description of the program (optional).
Schedule In	Select the task in which you plan to schedule the program. The programming software displays a list of available tasks
Type	The programming software automatically selects Normal. Other valid program types are Fault Handler and Power-Up Handler.

Naming programs

Program names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (__)

Program names are not case sensitive.

You can also add descriptions to programs. Descriptions can have as many as 128 characters. You can use any printable character.

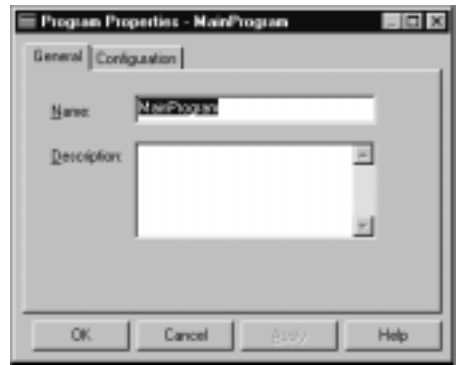
Configuring programs

- 1. Select a program ("MainProgram" in this example).
- 2. Click the right mouse button and select Properties.



Once you create a program, there are other properties that you need to configure. You must have a main routine. The fault and power-up routines are optional.

To configure an existing program:



On this tab:	In this field:	Enter:
General	Name	The programming software displays the current name of the program. Edit the name, if necessary.
	Description	The programming software displays the current description. Edit the description, if necessary.



On this tab:	In this field:	Enter:
Configuration	Assigned Routine	The programming software displays the name of the Main Routine and the Fault Routine, if any. Change the selections, if necessary.
	Scan Time (µs)	While online, the programming software displays the maximum scan time and the last scan time in µsec for the current program. These values are execution times for the program and do not include any time spent waiting for other programs or higher-priority tasks. These values are display only.

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.

Creating routines

When you open a new controller project in your programming software, the MainRoutine is already defined for the MainProgram. You can modify this routine, as well as add other routines.

To create a new routine:

1. Select a program ("MainProgram" in this example).
2. Click the right mouse button and select New Routine



In this field:	Enter:
Name	Enter the name of the routine.
Description	Enter a description of the routine (optional).
Type	Select the programming language used to create the routine. Ladder is the default.
In Program	Select the program in which you plan to run the routine. The programming software displays a list of the available programs.

Naming routines

Routine names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

Routine names are not case sensitive.

You can also add descriptions to routines. Descriptions can have as many as 128 characters. You can use any printable character.

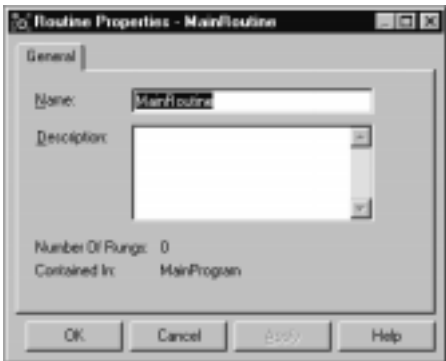
Configuring routines

1. Select a routine ("MainRoutine" in this example).
2. Click the right mouse button and select Properties.



Once you create a routine, you can change the name or the description of the routine.

To configure an existing routine:

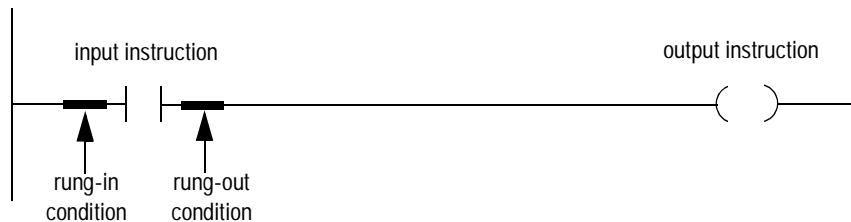


In this field:	Enter:
Name	The programming software displays the current name of the routine. Edit the name, if necessary.
Description	The programming software displays the current description. Edit the description, if necessary.
Number of rungs	<i>display only</i>
Contained in	<i>display only</i>

Entering Ladder Logic

The Logix5550 controller supports multiple output instructions per rung of logic. The output instructions can be in sequence on the rung (serial) or input and output instructions can be mixed, as long as the last instruction on the rung is an output instruction.

The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-condition-in). Based on the rung-condition-in and the instruction, the controller sets the rung condition following the instruction (rung-condition-out), which in turn, affects any subsequent instruction.

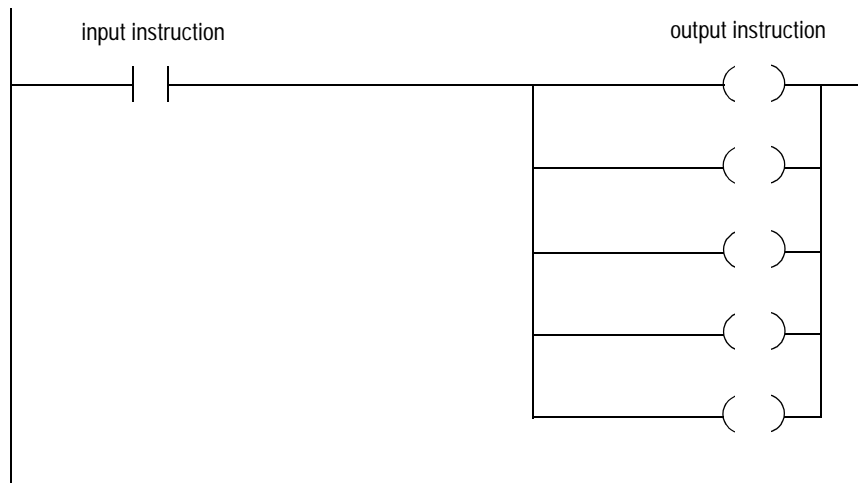


If the rung-condition-in to an input instruction is true, the controller evaluates the instruction and sets the rung-condition-out based on the results of the instruction. If the instruction evaluates to true, the rung-condition-out is true; if the instruction evaluates to false, the rung-condition-out is false.

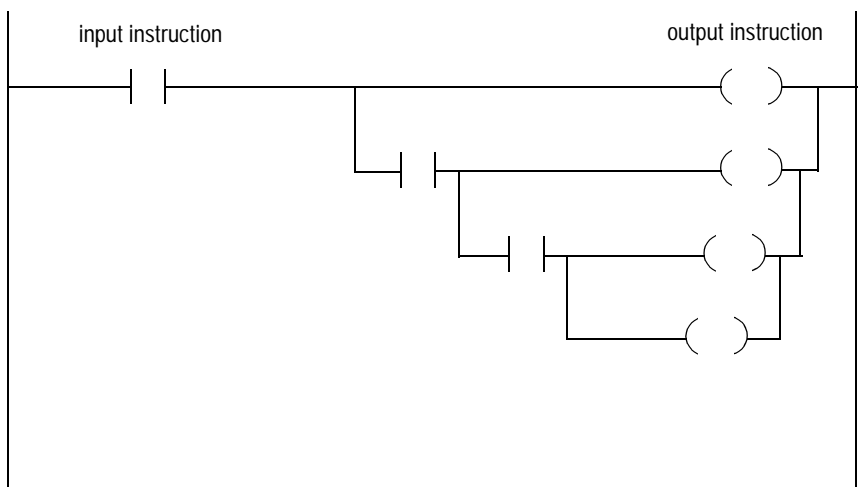
An output instruction does not change the rung-condition-out. If the rung-condition-in to an output instruction is true, the rung-condition-out is set to true. If the rung-condition-in to an output instruction is false, the rung-condition-out is set to false.

Entering branches

There is no limit to the number of parallel branch levels the controller supports. The following figure shows a parallel branch with five levels. The main rung is the first branch level, followed by four additional branches.



You can nest branches to as many as 6 levels. The following figure shows a nested branch. The bottom output instruction is on a nested branch that is three levels deep.

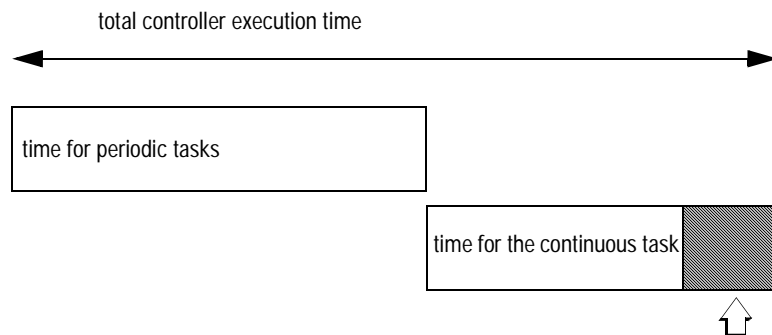


Scheduling System Overhead

The controller has a system overhead time slice that determines the percentage of controller time that is available for background functions, such as:

- communications with programming and MMI devices (such as the programming software)
- messaging, 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 the controller's serial port to other ControlLogix devices via the ControlLogix backplane

The percentage you select is taken from the time available to execute the continuous task. The percentage you select does not take time away from executing periodic tasks.

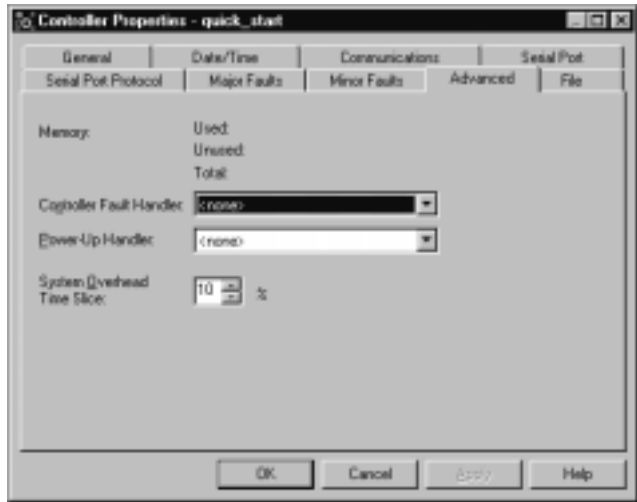


This time is available for communication functions. If there are no communication functions, this time is used by the continuous task.

Motion planning runs at the coarse iteration rate set for the motion group and will preempt all user tasks. Worst case task-execution time increases by the time needed for motion planning.

Select a percentage of the execution time for the continuous task to use for background functions and system overhead.

- 1. Select the controller (“quick_start” in this example).
- 2. Click the right mouse button and select Properties.



On this tab:	In this field:	Enter:
Advanced	SystemOverhead Time Slice	Select the percentage number (10-90%).

Use the default percentage (10%) unless your application is communication-intensive or communications aren’t being completed. As you increase the percentage, you reduce the time available to execute the continuous task, which may impact its overall execution time.

Increase the percentage if your application has all periodic tasks. In this case, there is no continuous task to execute.

Downloading a Project

To download a project to the controller:

- 1. Make sure the communication driver you need for the controller is properly configured through RSLinx. A communication driver makes sure the controller can communicate over a network. There is a different driver for each supported network.

The default communication driver for the controller is the DF1 driver, which you would use if the programming terminal is connected directly to the controller through the serial port.

If the programming terminal is connected to the controller through some other network path, configure the necessary communication driver.

For more information about configuring a connection path, see chapter 9.

1. From the Communications menu item, select Configure.
2. Select the Communications tab



2. Select the communication driver to use.



In this field:	Enter:
Driver	<p>Use the drop-down menu to select the driver. These selections are only available if they have already been configured through RSLinx communication software:</p> <ul style="list-style-type: none"> • ControlNet (AB_KTC) • DF1 (AB_DF1) • DH+ (AB_KT) • Ethernet (TCP) <p>The communication protocol is displayed next to selected driver.</p>
Path	<p>Specify the communication path to the controller from the device that is downloading the project.</p> <p>For more information about connection paths, see chapter 9.</p>

If the programming terminal is directly connected to the serial port of the controller (DF1 protocol), leave the connection path field empty.

1. From the Communications menu item, select Download.



3. The controller must be in Program or Remote Program mode to download a project.

If the controller is in Remote Run or Remote Test, you will be prompted for the software to change the mode to Remote Program for the download. When the download is complete, you will be prompted again for the software to change back to the previous mode.

Notes:

Communicating with Other Controllers

Using This Chapter

For information about:	See page:
Using MSG instructions	6-1
Using produced and consumed tags	6-6
Planning your system to support produced and consumed tags	6-9
Producing a tag	6-12
Consuming a tag	6-14
Sending large arrays of data	6-17

Using MSG Instructions

You can use MSG instructions to communicate between a Logix5550 controller and another controller.

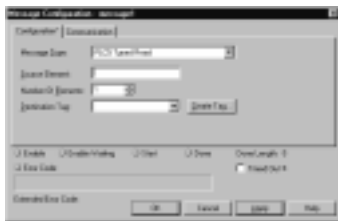
Communicating with another Logix5550 controller

Logix5550 controllers can use MSG instructions to communicate with each other. The following examples show how to use tags in MSG instructions between Logix5550 controllers.

Type of MSG Instruction:	Example Source and Destination:	
Logix5550 writes to Logix5550 (CIP Data Table Write)	source tag	<i>array_1</i>
	destination tag	<i>array_2</i>
Logix5550 reads from Logix5550 (CIP Data Table Read)	source tag	<i>array_1</i>
	destination tag	<i>array_2</i>

When you enter source and destination tags for a MSG instruction between two Logix5550 controllers:

- Both the source tag and the destination tag must be controller-scoped tags.
- Both the source tag and the destination tag can be of any data type, except for AXIS, MESSAGE, or MOTION_GROUP.
- You cannot specify array dimensions or structure members.
Use an alias instead. For example, instead of *array_1[3]*, specify *mytimer.acc*, which is an alias for that array element.
- You cannot transfer a portion of an array.
Either specify the entire array (by entering the array name) or one element of an array (by entering an alias).



Communicating with other processors

The Logix5550 controller also uses MSG instructions to communicate with PLC and SLC processors. The MSG instructions differ depending on which controller initiates the instruction.

For MSG instructions originating from a Logix5550 controller to a PLC or SLC processor:

Type of MSG Instruction:	Example Source and Destination:		Supported File Types:
Logix5550 writes to PLC-5	source element	<i>array_1</i>	for PLC-5: SINT, INT, DINT, or REAL for SLC: INT
Logix5550 writes to SLC or MicroLogix1000	destination tag	<i>N7:10</i>	for PLC-5 typed write: S, B, N, or F for PLC-5 word-range write: S, B, N, F, I, O, A, or D for SLC: B or N
	You can use an alias tag for the source tag. If you want to start at an offset within an array, use an alias to point to the offset.		
Logix5550 writes to PLC-2		<i>array_1</i>	SINT, INT, DINT, or REAL
Logix5550 reads from PLC-5	source element	<i>N7:10</i>	for PLC-5 typed read: S, B, N, or F for PLC-5 word-range read: S, B, N, F, I, O, A, or D for SLC: B or N
Logix5550 reads from SLC or MicroLogix1000	destination tag	<i>array_1</i>	for PLC-5: SINT, INT, DINT, or REAL for SLC: INT
	You can use an alias tag for the destination tag. If you want to start at an offset within an array, use an alias to point to the offset.		
Logix5550 reads from PLC-2	source element	<i>010</i>	SINT, INT, DINT, or REAL
	destination tag	<i>array_1</i>	

The Logix5550 controller can send typed or word-range commands to PLC-5 controllers. These commands read and write data differently. The following diagrams show how the typed and word-range commands differ.

Typed read command

16-bit words in
PLC-5 processor

1
2
3
4

32-bit words in
Logix5550 controller

	1
	2
	3
	4



The typed commands maintain data structure and value.

Word-range read command

16-bit words in
PLC-5 processor

1
2
3
4

32-bit words in
Logix5550 controller

2	1
4	3



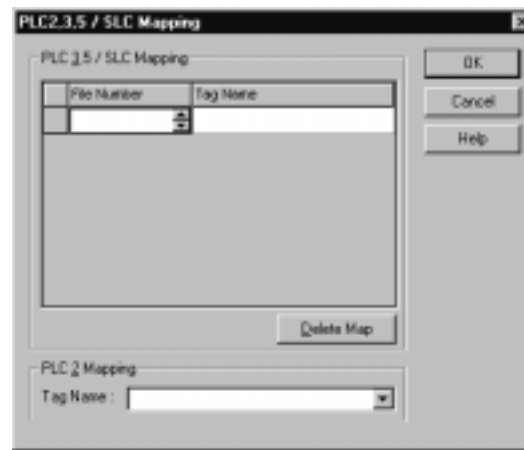
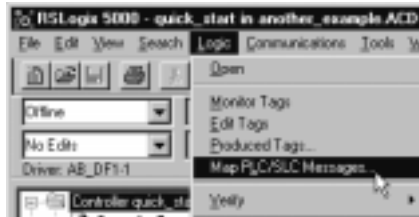
The word-range commands fill the destination tag contiguously. Data structure and value change depending on the destination data type.

The Logix5550 controller can process messages initiated from PLC or SLC processors. These messages use data table addresses. In order for these processors to access tags within the Logix5550 controller, you map tags to data table addresses.

The programming software includes a PLC/SLC mapping tool which allows you to make an existing controller array tag in the local controller available to PLC-2, PLC-3, PLC-5, or SLC processors.

Mapping addresses

To map addresses, specify this information.



For:	In this field:	Specify:	For example:
PLC-3, PLC-5, and SLC processors	File Number	Enter the file number of the data table in the PLC/SLC controller.	10
	Tag Name	Enter the array tag name the local controller uses to refer to the PLC/SLC data table address.	array_1
PLC-2 processors	Tag Name	Enter the tag name to be the PLC-2 compatibility file.	200

The tag in the local controller must be an integer array (SINT, INT, or DINT) that is large enough to support the message data.

You can map as many tags as you want to a PLC-3, PLC-5, or SLC processor. You can map only one tag to a PLC-2 processor.

The following examples show example source and destination tags and elements for different controller combinations.

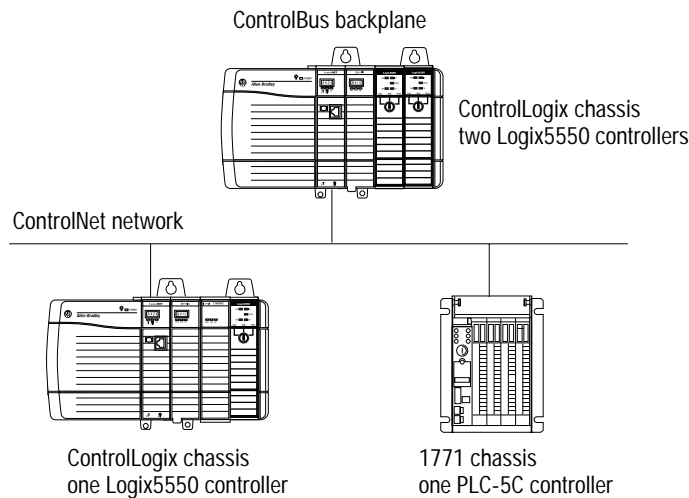
Type of MSG Instruction:	Example Source and Destination:
PLC-5 writes to Logix5550	source element <i>N7:10</i>
SLC writes to Logix5550	destination tag <i>"array_1"</i>
SLC 5/05	The PLC-5, PLC-3, and SLC processors support logical ASCII addressing so you do not have to map a compatibility file for MSG instructions initiated by a PLC-5, PLC-3, or SLC processor. Place the Logix5550 tag name in double quotes ("").
SLC 5/04 OS402 and above	
SLC 5/03 OS303 and above	
	You could optionally map a compatibility file. For example, if you enter <i>10</i> for the compatibility file, you enter <i>N10:0</i> for the destination tag.
PLC-2 writes to Logix5550	source element <i>010</i>
	destination tag <i>200</i>
	The destination tag is the three-digit PLC-2 address you specified for PLC-2 mapping.
PLC-5 reads from Logix5550	source tag <i>"array_1"</i>
SLC reads from Logix5550	destination element <i>N7:10</i>
SLC 5/05	The PLC-5, PLC-3, and SLC processors support logical ASCII addressing so you do not have to map a compatibility file for MSG instructions initiated by a PLC-5, PLC-3, or SLC processor. Place the Logix5550 tag name in double quotes ("").
SLC 5/04 OS402 and above	
SLC 5/03 OS303 and above	
	You could optionally map a compatibility file. For example, if you enter <i>10</i> for the compatibility file, you enter <i>N10:0</i> for the source tag.
PLC-2 reads from Logix5550	source tag <i>200</i>
	destination element <i>010</i>
	The source tag is the three-digit PLC-2 address you specified for PLC-2 mapping.

When the Logix5550 controller initiates messages to PLC or SLC controllers, you do not have to map compatibility files. You enter the data table address of the target device just as you would a tag name.

SLC 5/05 processors, SLC 5/04 processors (OS402 and above), and SLC 5/03 processors (OS303 and above) support logical ASCII addressing and support PLC/SLC mapping (see the examples above). For all other SLC or MicroLogix1000 processors, you must map a PLC-2 compatibility file (see the PLC-2 examples above).

Using Produced and Consumed Tags

The Logix5550 controller supports the ability to produce (broadcast) and consume (receive) system-shared tags. Produced and consumed data is accessible by multiple controllers over the ControlBus backplane or over a ControlNet network.



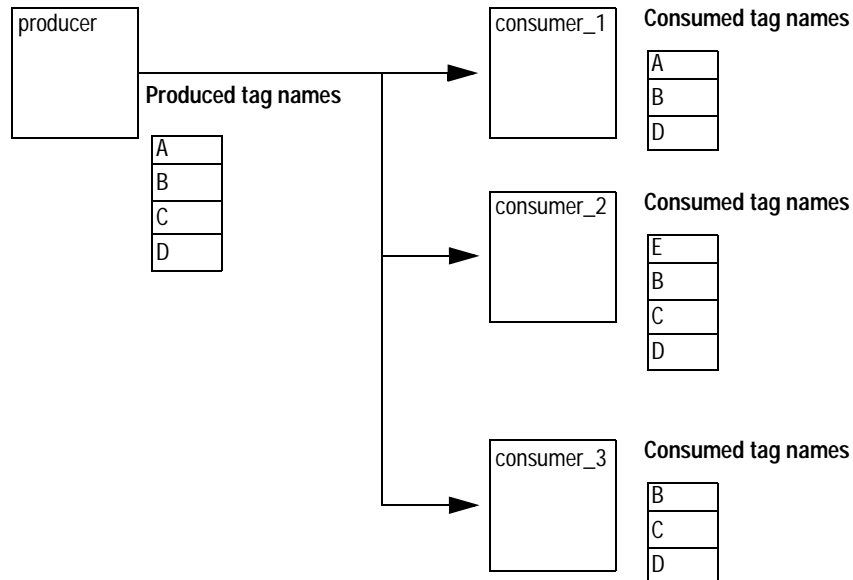
41029

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.	<ul style="list-style-type: none">• Enabled for producing• How many consumers allowed
consumed	These are tags whose values are produced by another controller.	<ul style="list-style-type: none">• 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

Processing produced and consumed tags

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. In the following example, *consumer_2* does not have the correct tags.



When *consumer_2* tries to access the shared tags, the connections fail. Even though three of the tags are specified correctly (B, C, and D), the connections fail for all the consumed tags because one was incorrect (E).

The other consumers (*consumer_1* and *consumer_3*) can still access the 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 following table shows the total number of produced and consumed tags a controller supports:

As a:	The controller supports:
consumer	$(\text{number of consumed tags}) \leq 250$ If your controller consumes 250 tags, these tags must come from more than one controller. A controller can only produce as many as 127 tags. See the producer numbers in this table.
producer	$(\text{number of produced tags}) \leq 127$

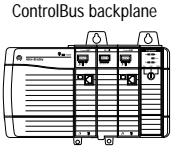
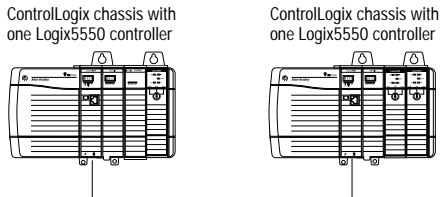
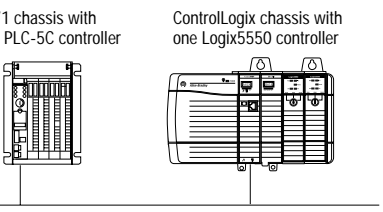
Each produced tag uses one unidirectional connection for the tag and one unidirectional connection for each controller that consumes the tag. With these maximum numbers in mind, the total combined consumed and produced tags that a controller supports is (this is also the maximum number of unidirectional connections; see chapter 7):

$$(\text{number of produced tags}) + (\text{number of consumed tags}) \leq 255$$

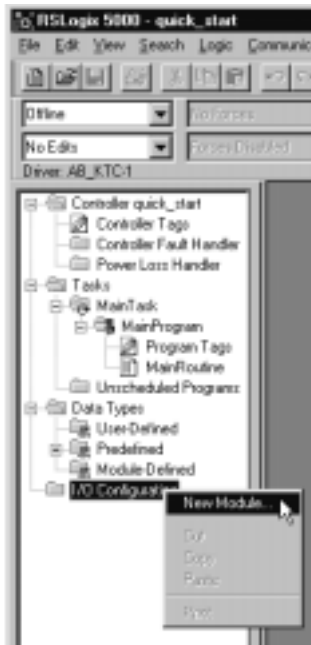
As the number of consumed tags increases, the number of available produced tags decreases. You increase the number of consumed tags either by creating consumed tags or by adding additional consumers to a produced tag.

Planning Your System to Support Produced and Consumed Tags

Before the Logix5550 controller can share produced or consumed tags, the other controllers must be configured in the controller organizer of the consuming controller. You can produce and consume data between these controllers:

You can share data between:	Over this network:
Logix5550 controller and local Logix5550 controller	ControlBus backplane
<div data-bbox="574 449 927 596">  <p>ControlBus backplane</p> <p>ControlLogix chassis with two Logix5550 controllers</p> </div>	
Logix5550 controller and remote Logix5550 controller	ControlNet network
<div data-bbox="428 716 1019 911">  <p>ControlLogix chassis with one Logix5550 controller</p> <p>ControlLogix chassis with one Logix5550 controller</p> <p>ControlNet network</p> </div>	
Logix5550 controller and PLC-5 ControlNet processor	ControlNet network
<div data-bbox="428 1079 1019 1289">  <p>1771 chassis with one PLC-5C controller</p> <p>ControlLogix chassis with one Logix5550 controller</p> <p>ControlNet network</p> </div>	

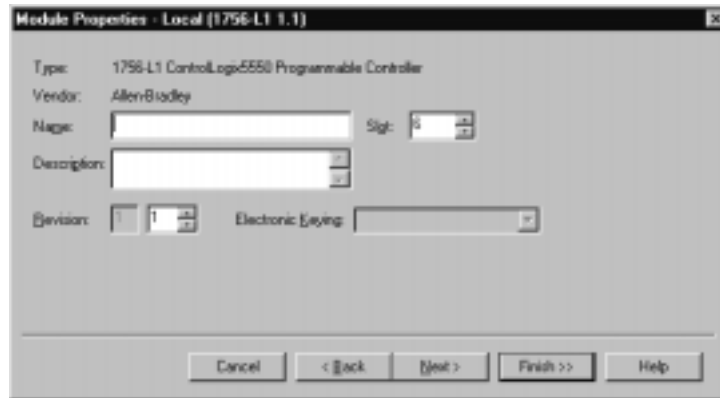
1. Select I/O Configuration
2. Click the right mouse button and select New Module



Identifying another local controller

Identifying another Logix5550 controller in the same chassis is similar to adding local I/O modules to the controller organizer.

To identify another Logix5550 controller, select the 1756-L1 controller. Specify:

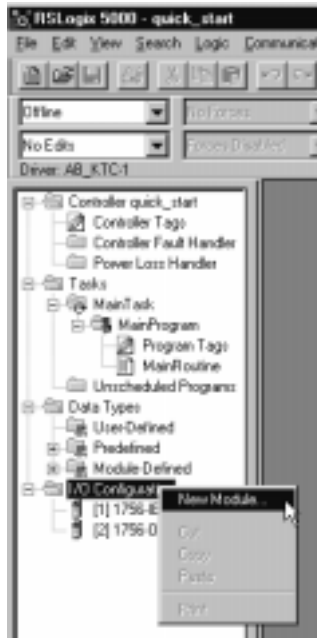


In this field:	Enter:
Name	Enter a name for the controller (required).
Description	Enter a description for the controller (optional).
Slot Number	Enter the slot number where the controller is installed.
Electronic Keying	Electronic keying is disabled, but you can still select a minor revision of the controller.

Identifying a remote controller

Identifying a remote controller is similar to adding I/O modules to the controller organizer. You can follow these steps for either a remote Logix5550 controller or a remote ControlNet PLC-5 controller. You must use a 1756-CNB or 1756-CNBR module.

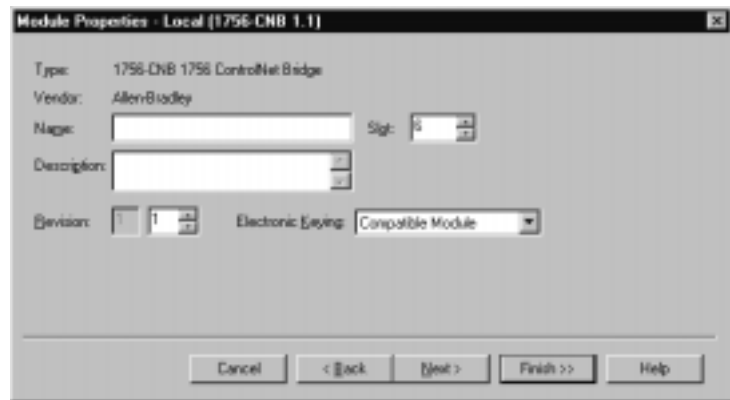
1. Select I/O Configuration
2. Click the right mouse button and select New Module



1. Select the 1756-CNB module
2. Click the right mouse button and select New Module



1. Configure a 1756-CNB module for the local chassis. This module handles communications between the local controller's chassis and the remote chassis. Specify:



In this field:	Enter:
Name	Enter a name for the module (required).
Description	Enter a description for the module (optional).
Slot Number	Enter the slot number where the module is installed.
Electronic Keying	Select an electronic keying method.

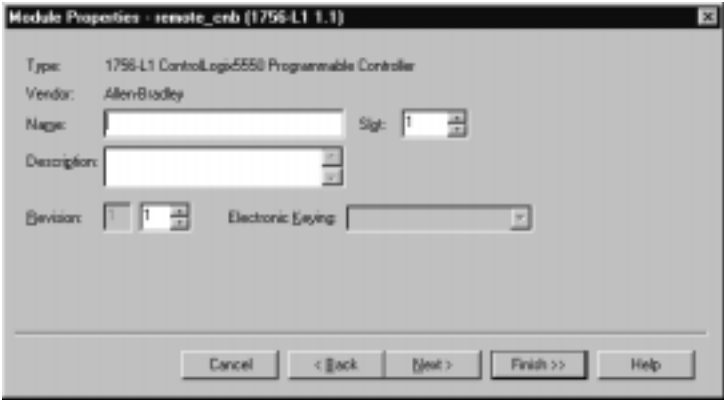
2. Configure another 1756-CNB communication module, this time for the communication module you just configured. This module handles communication for the remote chassis. Specify:



In this field:	Enter:
Name	Enter a name for the module (required).
Node	Enter a ControlNet node number for the module.
Description	Enter a description for the module (optional).
Chassis Size	Enter the number of slots in the chassis that contains the module.
Slot Number	Enter the slot number where the module is installed.
Communication Format	Select controller ownership of the module by selecting the communication format.
Electronic Keying	Select an electronic keying method.

1. Select the remote communication module

2. Click the right mouse button and select New Module
3. Identify a Logix5550 controller processor for the remote 1756-CNB module. This controller can share system data with the local Logix5550 controller. Specify:



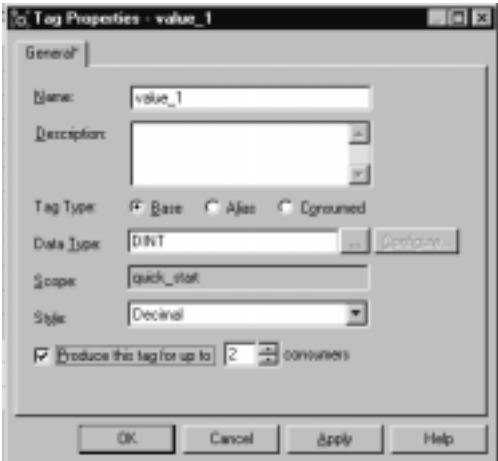
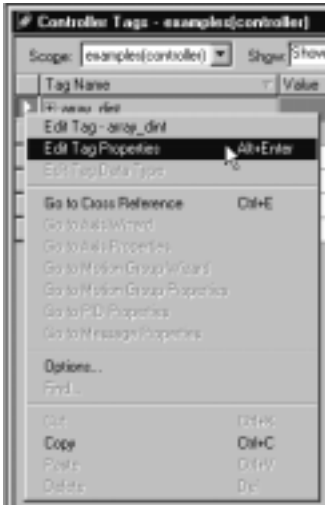
In this field:	Enter:
Name	Enter a name for the controller (required).
Description	Enter a description for the module (optional).
Slot Number	Select the slot number where the module is installed.
Electronic Keying	Electronic keying is disabled, but you can still select a minor revision of the controller.

If you want to add a PLC-5 ControlNet processor, add it to the local 1756-CNB communication bridge module.

Producing a Tag

1. In the Tag Editor, select the tag.

2. Click the right mouse button and select Tag Properties



To specify a produced tag:

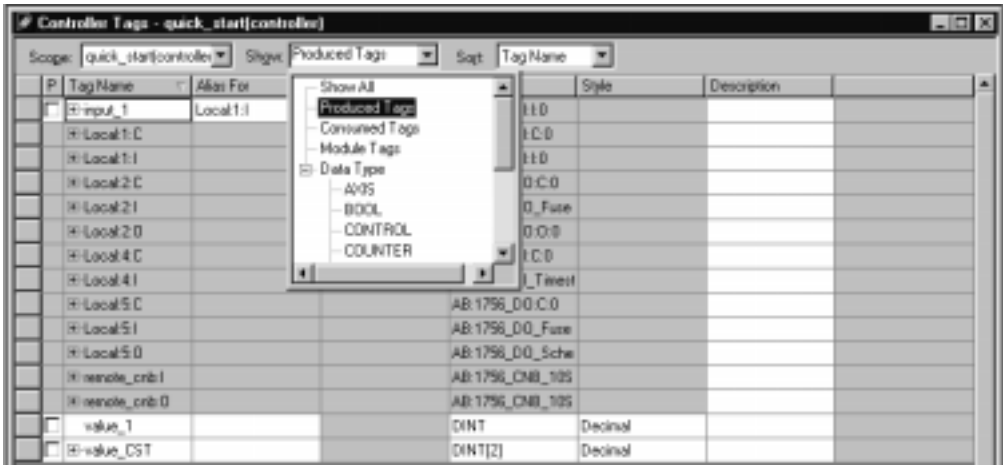
In this field:	Enter:
Name	Enter a name for the tag.
Description	Enter a description for the tag (optional).
Tag Type	Select one of these: Base normal tag Alias tag that represents another tag with the same characteristics Consumed tag that is produced by another controller whose data you want to use in this controller
Data Type	Select the data type. The programming software displays a list of the available data types. The list consists of the predefined data types and any user-defined data types. If the tag is to be an array, specify the number of elements in each dimension. There can be as many as 3 dimensions. If the tag is not an array, or you do not want all 3 dimensions, set the dimension fields to zero (0).
Scope	All produced tags must have controller scope.
Style	Select the display style of the tag. The programming software displays a list of the available styles, which depends on the data type. The style you select becomes the default display type when monitoring that tag with the programming software.
Produce this tag	Select whether to make this tag available to other controllers. Specify the maximum number of other controllers that can consume the tag. You can only choose to create a produced tag when programming offline.

A produced or consumed tag cannot be larger than 500 bytes. You can produce a base, alias, or consumed tag.

The consumed tag in a Logix5550 controller must have the same data type as the produced tag in the originating Logix5550 controller. The Logix5550 controller performs type checking to ensure proper data is being received.

You can display a list of produced tags in the tag editor of the current project.

Select Produced Tags.



Produced tags require connections. The number of connections depend on the amount of data and how many controllers are producing and consuming tags. For more information, see chapter 7.

Producing a tag from a Logix5550 controller to a ControlNet PLC-5 processor

To produce a tag that a ControlNet PLC-5 processor can consume, follow these steps:

1. Create a produced tag in the Logix5550 controller.
2. In RSNetworkx communication software, in the ControlNet configuration for the target PLC-5 controller, create a Receive Scheduled Message.

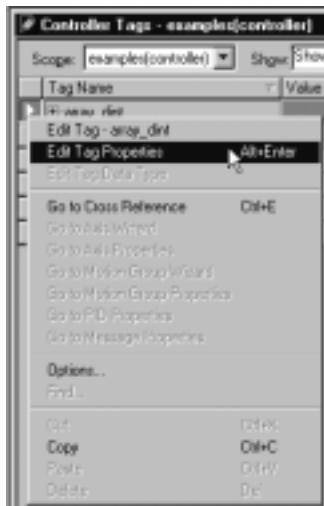
The input size of the scheduled message must match the number of bytes in the Logix5550 tag. A produced tag in the Logix5550 controller is always a multiple of 32 bits (DINT, REAL, or structure).

3. Schedule the link using RSNetworkx software.

The ControlNet PLC-5 controller does not perform type checking. Make sure the PLC-5 data type can correctly receive the Logix5550 produced tag to ensure proper data is being received.

Consuming a Tag

1. In the Tag Editor, select the tag.
2. Click the right mouse button and select Tag Properties



To create a consumed tag, create a tag and select the consumed tag type:

In this field:	Enter:
Name	Enter a name for the tag.
Description	Enter a description for the tag (optional).
Tag Type	Select: Consumed tag that receives data from a producing tag in another controller
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	Enter a name for the tag in the other controller you want to consume.
Remote Instance	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)	Enter 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.
Data Type	Select the data type. The programming software displays a list of the available data types. The list consists of the predefined data types and any user-defined data types. If the tag is an array, specify the number of elements in each dimension. There can be as many as 3 dimensions. If the tag is not an array, or you do not want all 3 dimensions, set the dimension fields to zero (0).
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.
Produce this tag	Select whether to make this tag available to other controllers. Specify how many controllers can consume the tag. You can only create a produced tag when programming offline.

Important: All consumed tags are automatically controller-scope.

A produced or consumed tag cannot be larger than 500 bytes. You can only create a consumed tag when programming offline.

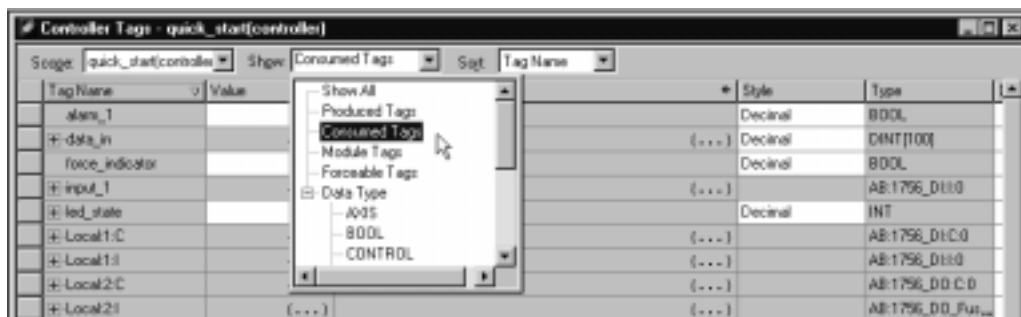
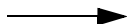
If a consumed tag is configured over a ControlNet network, you must use RSNetworkx to schedule its connection over the network.

The produced tag in the originating Logix5550 controller must have the same data type as the consumed tag in the other Logix5550 controller. The Logix5550 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 as well.

You can display a list of consumed tags in the tag editor of the current project.

Select Consumed Tags.



Produced tags require connections. The number of connections depend on the amount of data and how many controllers are producing and consuming tags. For more information, see chapter 7.

Consuming a tag from a ControlNet PLC-5 processor to a Logix5550 controller

To consume a tag from a ControlNet PLC-5 processor, follow these steps:

1. In RSNetworkx communication software in the ControlNet configuration for the PLC-5 controller, create a Send Scheduled Message.

Use an output size of at least 2 (for 32 bits).

2. In RSLogix5000 programming software, create a user-defined structure. The first member is a DINT. The second member is an INT array. The size of the INT array should match the output size entered in RSNetworkx.
3. Create a consumed tag of this user-defined type.

When you specify the tag as consumed, specify the instance of the Send Scheduled Data entry as the Remote Instance of the tag.

The requested packet interval (RPI) can be as low as the network update time (NUT).

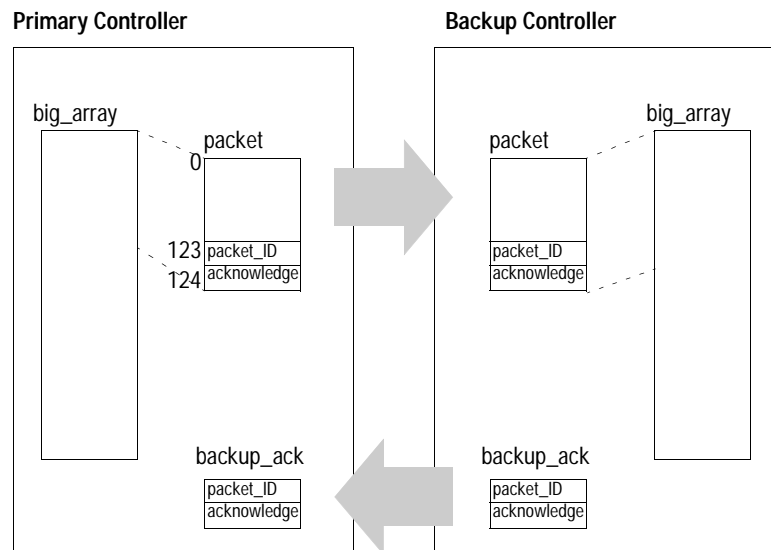
4. In RSNetworkx communication software, schedule the link.

Sending Large Arrays of Data

The Logix5550 controller can send as many as 500 bytes of data over a single scheduled connection. This corresponds to 125 DINT or REAL elements of an array. To transfer a larger amount of data, you could break that data into multiple arrays, each with a length of 125 elements. The problem with this approach is that the controller only supports 250 connections. An array with 5000 elements would take 40 connections ($5000/125=40$) using this approach.

Another way to transfer a large array is to use one PLC-5 type message to send the data as one large array. The problem with this approach is that messages are unscheduled and are executed only during the “system overhead” portion of the Logix5550 execution. Therefore, messages can take a fairly long time to complete the data transfer. You can improve the transfer time by increasing the amount of system time available to overhead, but this also diminishes the performance of the continuous task.

A better approach to transferring a large array is to use a one produced/consumed tag of 125 elements to “packetize” the array in one controller and send it piecemeal to another controller. The following example transfers `big_array` from a primary controller and creates a backup version on a second controller.



In this example, the primary controller moves 123 elements from `big_array` into a packet and appends two words to the end. The first word added (element 123) contains the `packet_ID`. The `packet_ID` is the starting offset of `big_array` from which the first element of the packet is obtained. The `packet_ID` will have values of 0, 123, 246, etc. The second word added (element 124) is an acknowledge word.

The primary controller creates a packet by determining what section of `big_array` to copy into the packet and setting the `packet_ID` equal to the first element location of that data section in the `big_array`. The primary controller then waits for the acknowledge word of the `backup_ack` array to be set equal to `-999` by the backup controller, which indicates that the backup controller has received a new packet.

The backup controller waits until it sees a value of the `packet_ID` element of the packet that is not equal to the previous value of the `packet_ID`. This signals the backup controller that a new packet has started to be received. The backup controller then sets the acknowledge word of `backup_ack` array equal to `-999` and waits to see this `-999` value returned by the primary controller in the acknowledge word of the packet.

When the primary controller sees that the backup controller has set the acknowledge word of `backup_ack` to `-999`, the primary controller moves the value of `-999` into the acknowledge word of the packet.

Once the backup controller sees that the primary controller has set the acknowledge word of the packet to `-999`, the backup controller knows that it has received a complete copy of the packet and can copy that data into the appropriate location in its own `big_array`. After the backup controller finishes copying the packet, the backup controller sets the acknowledge word of `backup_ack` array equal to zero and sets the `packet_ID` element of `backup_ack` array equal to the `packet_ID` of the packet. This signals the primary controller that the backup controller has finished copying the packet.

When the primary controller sees that the `packet_ID` of `backup_ack` is equal to the current value of the `packet_ID`, the primary controller then resets the acknowledge word of the packet equal to zero. The primary controller can now build the next packet.

This cycle of building and producing packets continues until the `packet_ID` value returned in `backup_ack` added to 123 is greater than or equal to the size of `big_array`. When this is true, the `packet_ID` in the packet is set equal to zero to start rebuilding packets starting at the beginning of `big_array`. Because the final packet might be right at the end of `big_array`, be sure that you create `big_array` to be at least 122 elements larger than the largest amount of data you will ever want to transfer. This ensures that you will never end up trying to copy more elements from or into `big_array` than really exist.

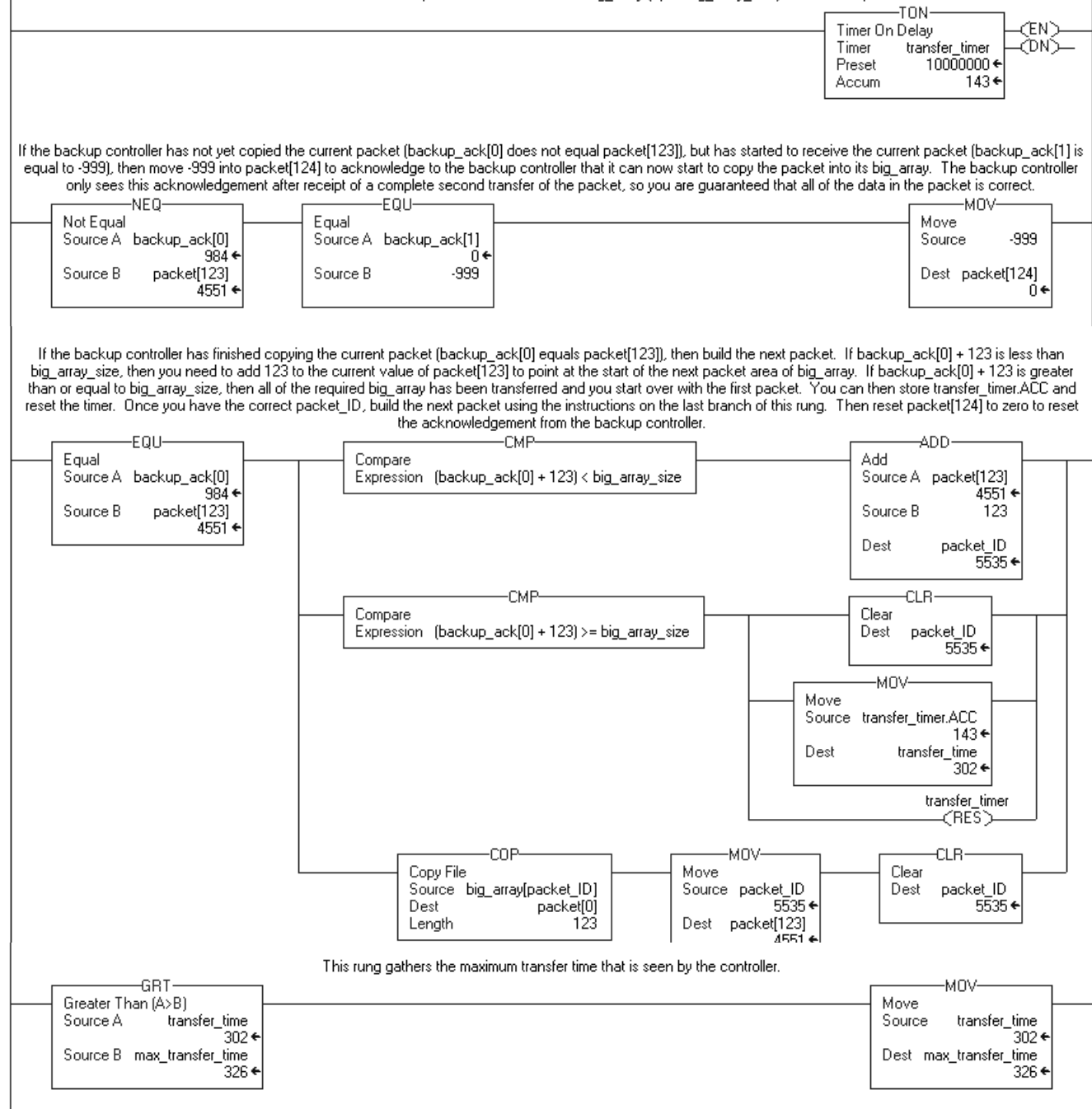
Produced data over the ControlBus backplane gets sent in 50 byte segments. This is asynchronous to the program scan in the controller. You need the acknowledge words to make sure that each controller is complete with its copying before either controller attempts its next copy. Otherwise, the primary controller could be building a new packet while the backup controller is still copying the previous packet, resulting in inaccurate data.

Example programming for the primary controller

The following ladder logic uses these tags:

Controller Tags - primary(controller)							
Scope:	primary(controller)	Show:	Show All	Sort:	Tag Name		
P	Tag Name	Alias For	Base Tag	Type	Style	Description	
<input type="checkbox"/>	[-] backup_array			DINT[12]	Decimal		
<input type="checkbox"/>	[-] big_array			DINT[10200]	Decimal		
<input type="checkbox"/>	[-] big_array_size			DINT	Decimal		
<input type="checkbox"/>	[-] max_transfer_time			DINT	Decimal		
<input checked="" type="checkbox"/>	[-] packet			DINT[125]	Decimal		
<input type="checkbox"/>	[-] packet_ID			DINT	Decimal		
<input type="checkbox"/>	[-] transfer_time			DINT	Decimal		
<input type="checkbox"/>	[-] transfer_timer			TIMER			

Use a timer to determine the total time required to transfer the entire big_array (up to big_array_size) to the backup controller.

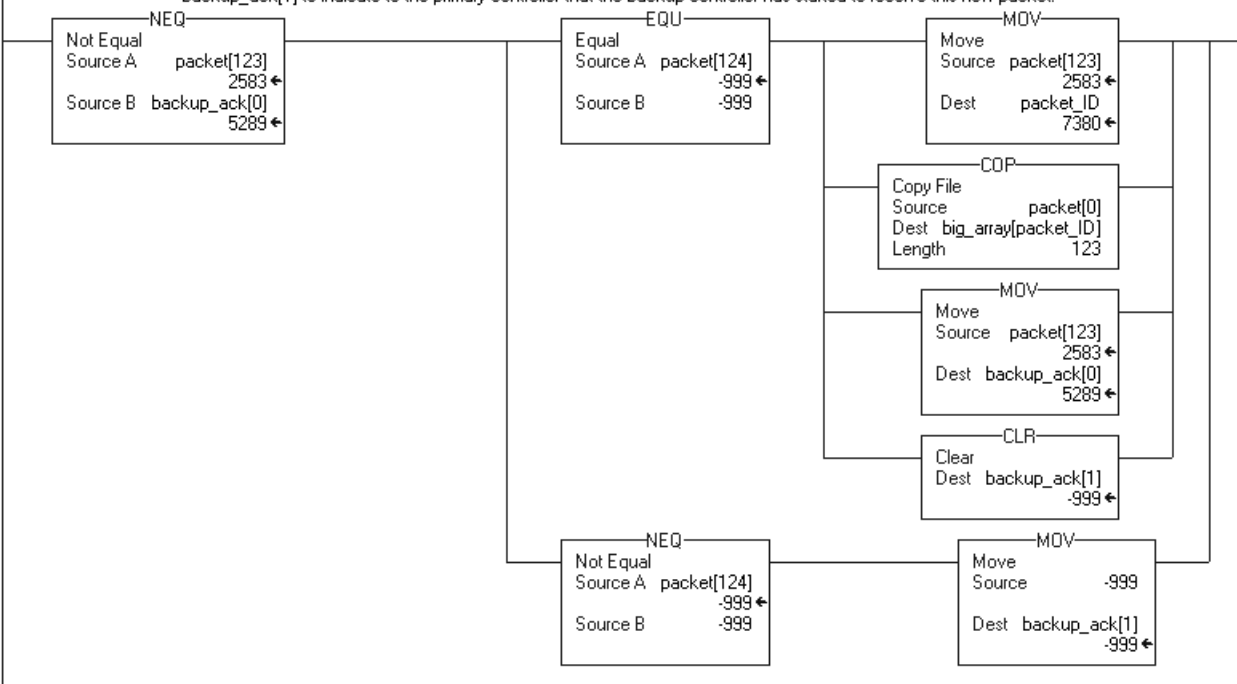


Example programming for the backup controller

The following ladder logic uses these tags:

Scope	Tag Name	Alias For	Base Tag	Type	Style	Description
	backup_ack			DINT[2]	Decimal	
	big_array			DINT[10200]	Decimal	
	packet			DINT[125]	Decimal	
	packet_ID			DINT	Decimal	
	stop_bit1			BOOL	Decimal	

This rung first checks to see if packet[123] received from the primary controller is different than backup_ack[0], which was set equal to the packet_ID for the previous packet that was copied into big_array on the backup controller. If these values are different, you know that a new packet has started to be received. Then the rung checks to see if packet[124] is equal to -999. If so, then the backup controller has received the entire packet and can begin copying it into the backup controller's big_array, set backup_ack[0] equal to packet[123], and reset backup_ack[1]. If packet[123] is different than backup_ack[0], move -999 into backup_ack[1] to indicate to the primary controller that the backup controller has started to receive this new packet.



Allocating Communication Connections

Using This Chapter

For information about:	See page:
How the ControlLogix system uses connections	7-1
Determining I/O connection requirements	7-2
Determining connections for produced and consumed tags	7-6
Determining connections for messaging	7-7
Determining total connection requirements	7-8

How the ControlLogix System Uses Connections

The ControlLogix system uses a connection to establish a communication link between two devices. This includes controllers, communication modules, input/output modules, produced/consumed tags, and messages. Connections take many forms:

- controller direct to local I/O or local communication module
- controller direct to remote I/O or remote communication module
- controller to remote chassis (rack optimized)
- produced and consumed tags
- messaging, including block-transfers

You indirectly determine the number of connections that the Logix5550 controller requires by configuring the controller to communicate with other devices in the system.

Each module in the ControlLogix system supports a limited number of active connections. Take these connection limits into account when designing your system. These modules support these number of connections:

Device:	Description:	Connections:
1756-L1	Logix5550 Controller	250 bidirectional (500 unidirectional)
1756 I/O modules	ControlLogix I/O modules	16 bidirectional
1756-CNB	ControlLogix ControlNet Bridge	64 bidirectional
1756-CNBR		
1756-ENET	ControlLogix Ethernet Bridge	16 bidirectional
1756-DHRIO	ControlLogix DH+ Bridge and Remote I/O Scanner	32 bidirectional per DH+ channel 32 bidirectional rack connections and 16 bidirectional block-transfer connections per remote I/O channel
1756-DNB	ControlLogix DeviceNet Bridge	2 bidirectional

Determining Connections for I/O Modules

All I/O modules can have a direct, bidirectional connection to the controller. A 1756-CNB ControlNet bridge module supports the ability to organize a chassis of digital I/O modules into one bidirectional connection (rack connection), rather than requiring a direct bidirectional connection for each individual I/O module.

You can configure these types of connections to these modules:

A Logix5550 connection to:	Can use this connection type:
local I/O	direct connection only
remote I/O	direct connection
	or
	rack optimized connection

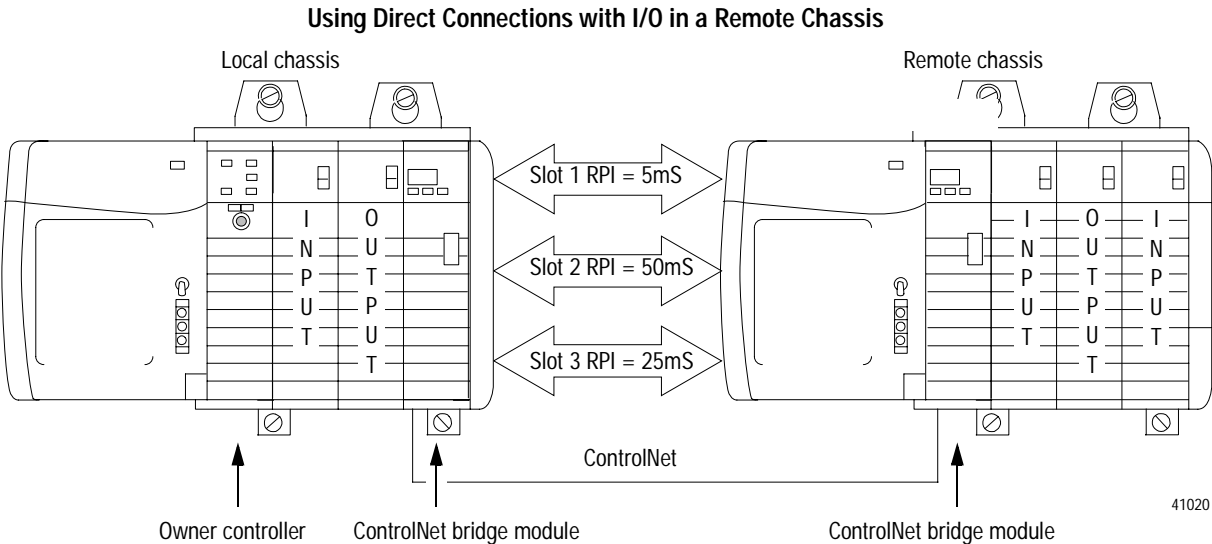
Direct connections for I/O modules

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 from the chassis while under power, causes the controller to set fault status bits in the data area associated with the module.

If a controller has a module configuration that references a slot in the control system, the controller periodically checks for the presence of a device in that slot. When a device's presence is detected there, the controller automatically sends the module configuration.

If the module configuration is appropriate for the I/O module found in the slot, a connection is made and operation begins. If the module configuration is not appropriate, the connection is rejected. You can view the fault message on the Connection tab of the module's properties. Module configuration can be inappropriate for any of a number of reasons. For example, a mismatch in electronic keying that prevents normal operation.

In this example, the owner controller has three direct connections with I/O modules in the remote chassis.



The local controller in this example uses these bidirectional connections:

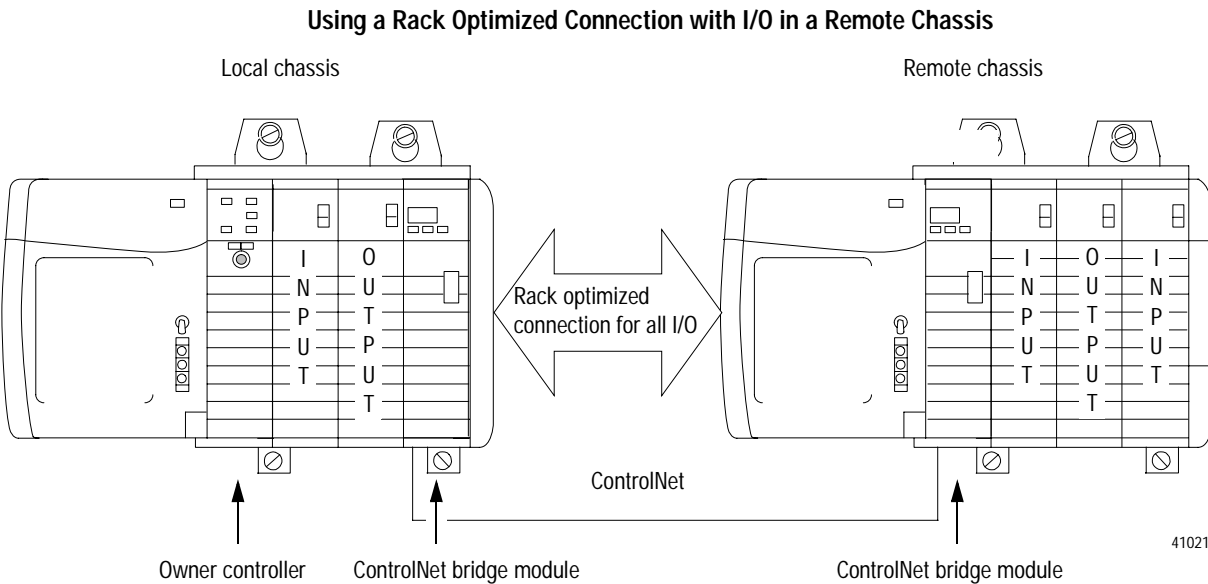
Connection Type:	Module Quantity:	Bidirectional Connections per Module:	Total Connections:
controller to local I/O module	2	1	2
controller to remote I/O module	3	1	3
controller to remote 1756-CNB module	1	1	1
total			6

When you select the communication format for the I/O module, the owner and listen-only formats are direct connections.

Rack optimized connections for I/O modules

When a digital I/O module is located in a remote chassis (with respect to its owner), you can select rack optimized communication. A rack optimized connection consolidates connection usage between the owner and the digital I/O in the remote chassis. Rather than having individual, direct connections for each I/O module, there is one connection for the entire chassis.

In this example, the owner controller communicates with all the digital I/O in the remote chassis but uses only one connection. The data from all three modules is sent together simultaneously at a rate specified by the 1756-CNB connection. This option eliminates the need for the three separate connections shown in the previous example.

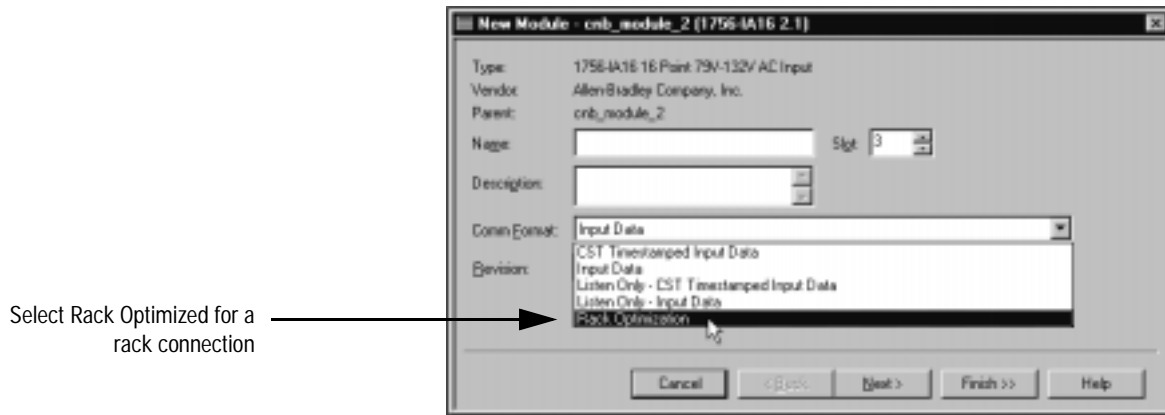


The local controller in this example uses these bidirectional connections:

Connection Type:	Module Quantity:	Bidirectional Connections per Module:	Total Connections:
controller to local I/O module	2	1	2
controller to remote 1756-CNB module	1	1	1
total			3

The rack optimized connection conserves ControlNet connections and bandwidth, but it limits the status and diagnostic information that is available from the I/O modules.

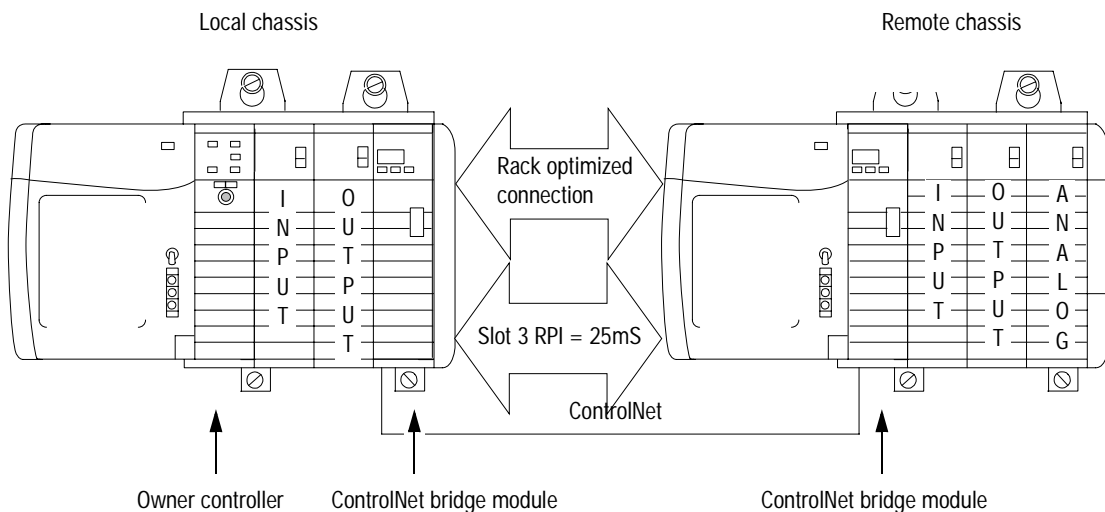
To configure an I/O module for a rack optimized connection, you select the Rack Optimization communication format. Most of the discrete I/O modules support a rack optimized connection. If this option does not appear when you are selecting communication format for an I/O module, the module does not support the rack optimized connection.



Combining direct and rack optimized connections

A remote chassis can have both a rack optimized connection and direct connections. In this example, the owner controller uses a rack optimized connection to communicate with two digital I/O modules. The owner controller also uses a direct connection to communicate with an analog module in the same chassis.

Using a Rack Optimized Connection and a Direct Connection with I/O in a Remote Chassis



41030

The local controller in this example uses these bidirectional connections:

Connection Type:	Module Quantity:	Bidirectional Connections per Module:	Total Connections:
controller to local I/O module	2	1	2
controller to remote analog I/O module	1	1	1
controller to remote 1756-CNB module	1	1	1
total			5

Determining Connections for Produced and Consumed Tags

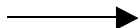
The Logix5550 controller supports the ability to produce (broadcast) and consume (receive) system-shared tags. System-shared data is accessible by multiple controllers over the ControlBus backplane or over a ControlNet network. Produced and consumed tags each require connections.

Connections for produced tags

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 unidirectional connection for each consumer and one more unidirectional connection for the produced tag. The default produced tag requires three unidirectional connections.

You define the number of consumers through the Tag Properties.

specify the maximum number of consumers for this produced tag



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.

Optimizing produced tags

Each produced tag requires connections that can be used for other controller operations. To minimize the of produced tags, and the number of required connections, consider grouping data into an array or a user-defined structure and producing only that array or structure, as long as the array or structure is not larger than 500 bytes.

For example:

Definitions:		Produced Tags:	Unidirectional Connections (default number of 2 consumers):
<i>height</i>	DINT data type	<i>height</i>	3
<i>width</i>	DINT data type	<i>width</i>	3
<i>weight</i>	REAL data type	<i>weight</i>	3
<i>W_flag</i>	DINT data type	<i>W_flag</i>	3
<i>L_flag</i>	DINT data type	<i>L_flag</i>	3
			total: 15 unidirectional connections
<i>Load_Info</i>	structure of:	<i>Load_Info</i>	3
<i>height</i>	DINT data type	total: 3 unidirectional connections	
<i>width</i>	DINT data type		
<i>weight</i>	REAL data type		
<i>W_flag</i>	DINT data type		
<i>L_flag</i>	DINT data type		

Connections for consumed tags

Each consumed tag requires one unidirectional connection for the controller that is consuming the tag.

Determining Connections for Messaging

The Logix5550 controller uses connections to perform messaging, including block-transfers. When your logic uses a message instruction to read or write information to or from another module, that instruction requires one bidirectional connection for the duration of the transmission. Depending on how you configure the message instruction using the .EN_CC enable caching bit, the connection remains open until the controller stops executing the logic or the connection is closed after the message transmission.

Message instructions that execute repeatedly should keep the connection open (set the .EN_CC bit) to optimize execution time. Opening a connection each time to execute an instruction would increase execution time. Message instructions that operate infrequently can close connections upon completion to free up connections for other uses.

Determining Total Connection Requirements

The Logix5550 controller supports 250 bidirectional connections. Use the following table to tally connection requirements for a controller. This table calculates **bidirectional** connections:

Connection Type:	Module Quantity:	Bidirectional Connections per Module:	Total Connections:
I/O modules (direct connections)		1	
to 1756-M02AE servo module		3	
to local 1756-CNB module		0	
to remote 1756-CNB module		2	
to 1756-DHRIO module		1	
to 1756-ENET module		0	
to 1756-DNB module		2	
to Universal Remote I/O adapter module		1	
produced tags			
produced tag		.5	
number of consumers		.5	
consumed tags		.5	
block-transfer messages		1	
other messages		1	
total			

Communicating with Devices on a Serial Link

Using This Chapter

For information about:	See page:
Using RS-232	8-1
Connecting to the serial port	8-2
Using the DF1 serial protocol	8-3
Configuring serial communications	8-5

Using RS-232

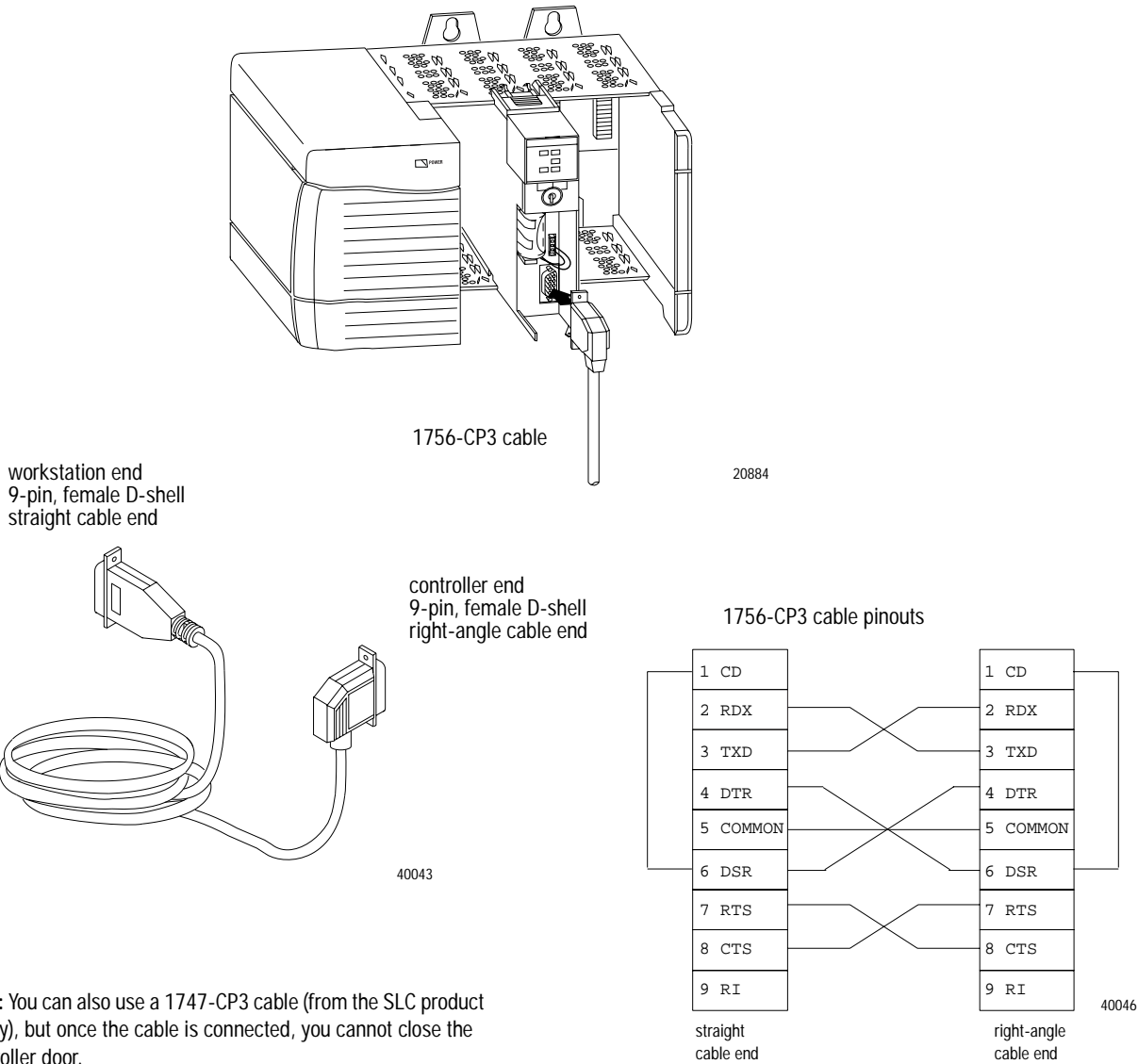
The controller supports RS-232 on the serial port. Use RS-232 when you have:

- a data transmission range of up to 50 ft. (15.2m).
- an application that requires modems or line drivers.

The maximum cable length for RS-232 communications is 15.2m (50 ft.).

Connecting to the Serial Port

The controller has a 9-pin serial port on the front panel.

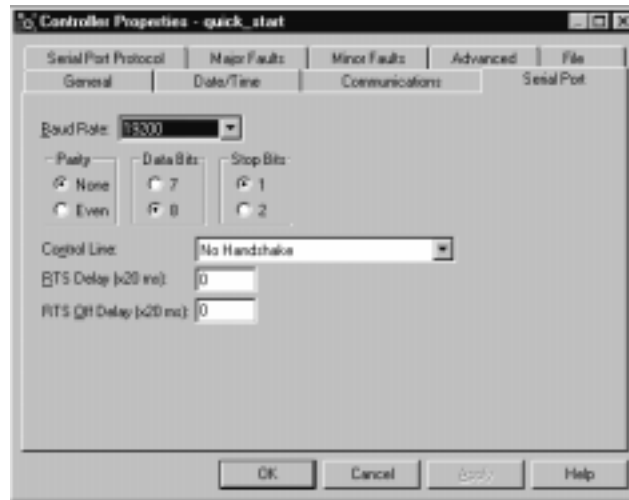


Configuring the controller to use the serial port

To configure the serial port, specify these characteristics (default values are shown in bold):



or



Characteristic:	Description (default is shown in bold):
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.
Parity	Specifies the parity setting for the serial port. Parity provides additional message-packet error detection. Select None or Even.
Data bits	Specifies the number of bits per message packet. Select 8 .
Stop bits	Specifies the number of stop bits to the device with which the controller is communicating. Select 1 or 2.
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.

Characteristic:	Description (default is shown in bold):
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.

Using the DF1 Serial Protocol

All data is encapsulated inside a DF1 protocol packet. The controller can communicate only with peripheral devices that support the DF1 protocol. Examples of DF1 peripheral devices are:

- programming terminals
- communication modules
- display terminals

The available system modes are:

Use this mode:	For:	See this 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.	8-6
DF1 master mode	control of polling and message transmission between the master and each remote node. The master/slave network includes one controller configured as the master node and as many as 254 slave nodes. You 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).	8-7
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, you 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.	8-8

Master/slave communication methods

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

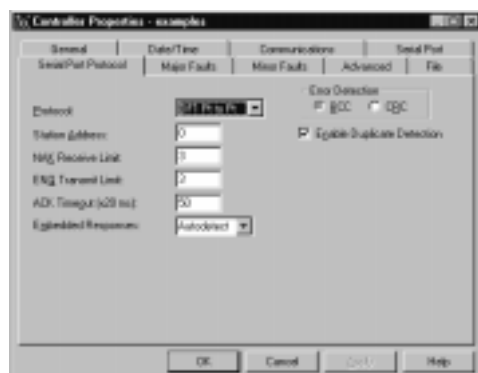
Name:	This method:	Benefits:
standard communication mode	<p>Initiates polling packets to slave stations according to their position in the polling array(s).</p> <p>Polling packets are formed based on the contents of the normal poll array and the priority poll array.</p>	<p>This communication method is most often used for point-to-multipoint configurations.</p> <p>This method provides these capabilities:</p> <ul style="list-style-type: none"> • 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 <p>The poll array resides in a user-designated data file. You can configure the master:</p> <ul style="list-style-type: none"> • to send messages during its turn in the poll array <p>or</p> <ul style="list-style-type: none"> • for between-station polls (master transmits any message that it needs to send before polling the next slave station) <p>In either case, configure the master to receive multiple messages or a single message per scan from each slave station.</p>
message-based communication mode	<p>initiates communication to slave stations using only user-programmed message (MSG) instructions.</p> <p>Each request for data from a slave station must be programmed via a MSG instruction.</p> <p>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.</p>	<p>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.</p> <p>Also choose this method if you need to communicate with non-intelligent remote terminal units (RTUs).</p>

Configuring Serial Communications

You configure the controller for the DF1 protocol on the Serial Port Protocol tab of the Controller Properties. Select one of these modes:

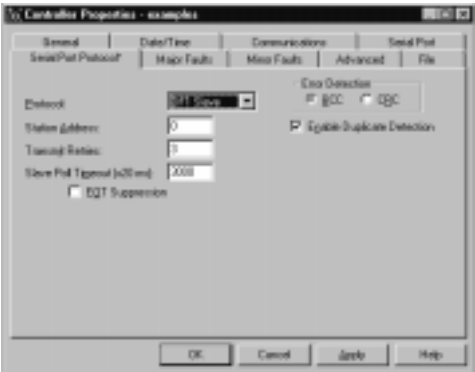
- DF1 point-to-point
- DF1 master
- DF1 slave

Configuring a DF1 point-to-point station



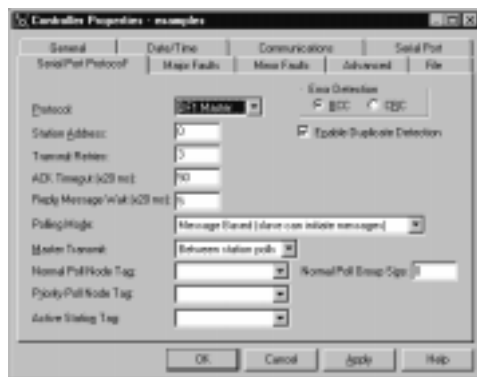
This field:	Description:
Station address	The station address for the serial port on the DF1 point-to-point network. Enter a valid DF1 address (0-254). Address 255 is reserved for broadcast messages. The default is 0.
NAK receive limit	Specifies the number of NAKs the controller can receive in response 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.

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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • between station polls (default) • in polling sequence
Normal poll node tag	Standard polling modes only An integer array that contains the station addresses of the slave stations (in the order in which to poll the 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: <i>list[0]</i> contains total number of stations to poll <i>list[1]</i> contains address of station currently being polled <i>list[2]</i> contains address of first slave station to poll <i>list[3]</i> contains address of second slave station to poll <i>list[n]</i> 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.

This field:	Description:
Priority poll node tag	<p>Standard polling modes only</p> <p>An integer array that contains the station addresses of the slave stations you need to poll more frequently (in the order in which to poll the stations).</p> <p>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.</p> <p>This tag must be controller-scoped. The format is: <i>list[0]</i> contains total number of stations to be polled <i>list[1]</i> contains address of station currently being polled <i>list[2]</i> contains address of first slave station to poll <i>list[3]</i> contains address of second slave station to poll <i>list[n]</i> contains address of last slave station to poll</p>
Active station tag	<p>Standard polling modes only</p> <p>An array that stores a flag for each of the active stations on the DF1 link.</p> <p>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.</p> <p>Create a single-dimension array of data type SINT that has 32 elements (256 bits). This tag must be controller-scoped.</p>
Error detection	<p>Select BCC or CRC error detection.</p> <p>Configure both stations to use the same type of error checking.</p> <p>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.</p> <p>CRC: the controller sends and accepts messages with a 2-byte CRC for error checking. CRC is a more complete method.</p>
Enable duplicate detection	<p>Select whether or not the controller should detect duplicate messages. The default is duplicate detection enabled.</p>

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.

Notes:

Communicating with a Workstation

Using This Chapter

For information about:	See page:
Configuring communications to the controller from a workstation	9-1
Defining connection paths	9-2

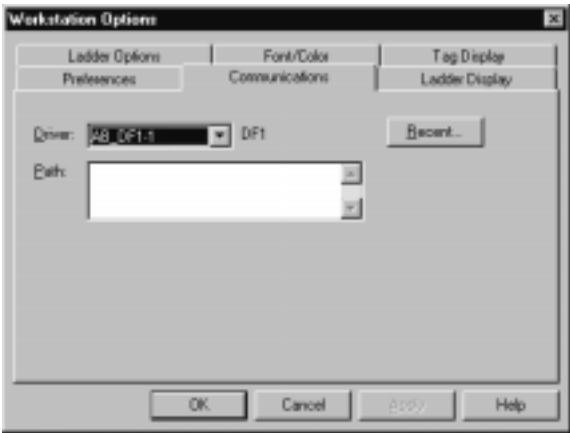
This chapter discusses configuring connection paths so the controller can communicate over networks. For information about configuring serial parameters, see the previous chapter, “Communicating with Devices on a Serial Link.”

Configuring Communications to the Controller from a Workstation

1. From the Communications menu item, select Configure.
2. Select the Communications tab.



To communicate from a workstation to a controller, you must configure the appropriate communication driver for the network that links the workstation and the controller. The communication driver enables the controller to communicate over the network. You must configure communication drivers with RSLinx software and then select the appropriate driver in the programming software.



In this field:	Enter:
Driver	<p>This is a display-only field that describes the communication protocol of the selected driver.</p> <p>Use the drop-down menu to select the driver. Only drivers that have been configured using RSLinx software appear. The type of driver is reflected in the name of the driver:</p> <ul style="list-style-type: none">• ControlNet (AB_KTC)• DF1 (AB_DF1)• DH+ (AB_KT)• Ethernet (TCP)
Path	<p>This is the connection path to the controller you wish to communicate with from the communications card you are connected to. The path consists of a sequence of decimal numbers separated by commas.</p> <p>The field displays up to three lines for a long path, and a scroll-bar appears if the entire field cannot be displayed.</p>
Recent	<p>This button navigates to the Recent Configurations dialog where you can choose from the recent configurations stored on the workstation.</p>

Only those drivers that have been configured in RSLinx software can be used to communicate to the controller.

Defining Connection Paths

For ControlNet and DH+ communications, the connection path starts with the controller or the communications card in the workstation. For Ethernet and DF1 communications, the connection path starts with the communication module in the chassis.

The following steps construct a communication path. Separate the number or address entered in each step with a comma. All numbers are in decimal by default. You can enter any number, other than an Ethernet IP address, in another base by using the IEC-1131 prefix (8# for octal, 16# for hexadecimal). Ethernet IP addresses are always decimal numbers separated by periods.

If you are using a DF1 point-to-point connection directly from the workstation to the serial port of the controller, leave the path blank.

To construct the path, you enter one or more path segments that lead to the controller. Each path segment takes you from one module to another module over the ControlBus backplane or over a DH+, ControlNet, or Ethernet network.

Each path segment contains two numbers:

x,y

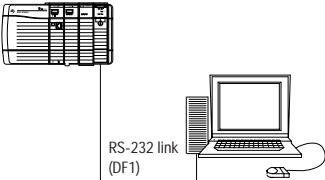
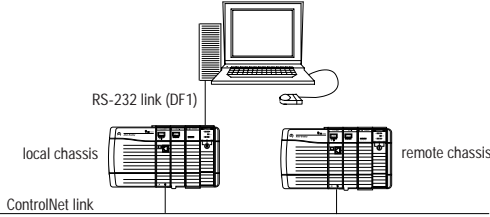
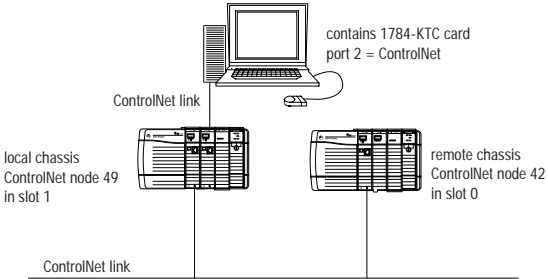
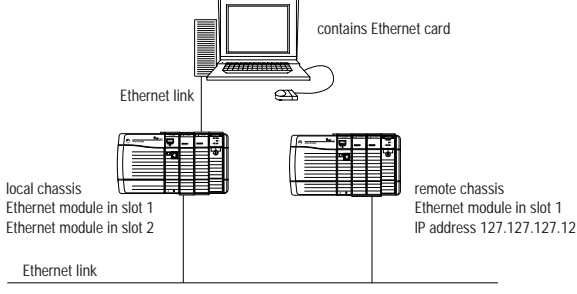
Where:

This	Is:
x	number of the type of port you use to exit from the module you are at:
	0 DH+ port from a KT, KTx, or KTxD card
	1 ControlBus backplane from any 1756 module
	2 DF1 port from a 1756-L1 controller
	2 ControlNet port from a KTC card or a 1756-CNB module
	2 Ethernet port from a 1756-ENET module
	2 DH+ port over channel A from a 1756-DHRIO module
	3 DH+ port over channel B from a 1756-DHRIO module
,	separates the starting point and ending point of the path segment
y	address of the module you are going to
	For
	ControlBus backplane slot number
	DF1 network serial port station address (0-254)
	ControlNet network node number (1-99 decimal)
	DH+ network node number (0-77 octal)
	Ethernet network IP address (four decimal numbers separated by periods)

If you have multiple path segments, you must also separate each path segment with a comma (,).

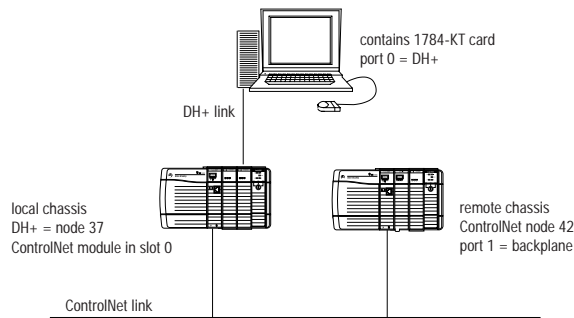
Connection path examples

The following examples are based on this system:

Example:	Description:
<p>serial Use DF1 to connect directly to the controller in the local chassis.)</p> 	<p>Select a DF1 driver. Leave the connection path blank.</p>
<p>serial Use DF1 to connect to the controller in the remote chassis.</p> 	<p>Select a DF1 driver. Enter connection path: 1,0,2,42,1,3</p> <p>1 = backplane port of the Logix5550 controller in slot 3 of the local chassis 0 = slot number of the 1756-CNB module in the local chassis 2 = ControlNet port of the 1756-CNB module in slot 0 of the local chassis 42 = ControlNet node of the 1756-CNB module in slot 0 of the remote chassis 1 = backplane port of the 1756-CNB module in slot 0 of the remote chassis 3 = slot number of the controller in the remote chassis</p>
<p>ControlNet Use ControlNet to connect to the controller in the remote chassis.</p> 	<p>Select a ControlNet driver. Enter connection path: 2, 49, 1, 0, 2, 42, 1, 3</p> <p>2 = ControlNet port of the KTC communications card in the workstation 49 = ControlNet node of the 1756-CNB module in slot 1 of the local chassis 1 = backplane port of the 1756-CNB module in slot 1 of the local chassis 0 = slot number of the 1756-CNB module in the local chassis 2 = ControlNet port of the 1756-CNB module in slot 0 of the local chassis 42 = ControlNet node of the 1756-CNB module in slot 0 of the remote chassis 1 = backplane port of the 1756-CNB module in slot 0 of the remote chassis 3 = slot number of the controller in the remote chassis</p>
<p>Ethernet Use Ethernet to connect to the controller in the remote chassis.</p> 	<p>Select an Ethernet driver. Enter connection path: 1, 1, 2, 127.127.127.12, 1, 3</p> <p>1 = backplane port of the 1756-ENET module in slot 2 of the local chassis 1 = slot number of the other 1756-ENET module in the local chassis 2 = Ethernet port of the 1756-ENET module in slot 1 of the local chassis 127.127.127.12 = IP address of the 1756-ENET module in the remote chassis 1 = backplane port of the 1756-ENET module in slot 1 of the remote chassis 3 = slot number of the controller in the remote chassis</p>

Example:**Description:**

DH+ Connect to the local chassis through a DH+ link. Go out through a ControlNet link to connect to the controller in the remote chassis.



Select a DH+ driver.

Enter connection path: 0, 8#37, 1, 0, 2, 42, 1, 3

0 = DH+ port of the KT communications card in the workstation

8#37 = octal DH+ node of the 1756-DHRIO module in slot 2 of the local chassis

1 = backplane port of the 1756-DHRIO module in slot 2 of the local chassis

0 = slot number of the 1756-CNB module in the local chassis

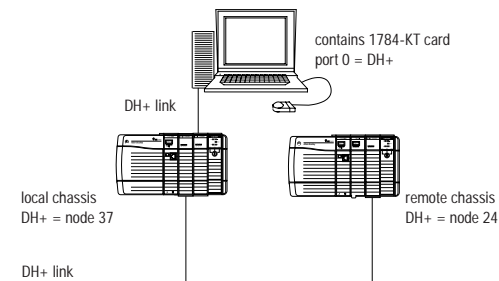
2 = ControlNet port of the 1756-CNB module in slot 0 of the local chassis

42 = ControlNet node of the 1756-CNB module in slot 0 of the remote chassis

1 = backplane port of the 1756-CNB module in slot 0 of the remote chassis

3 = slot number of the controller in the remote chassis

DH+ Use DH+ to connect to the controller in the remote chassis.



Select a DH+ driver.

Enter connection path: 0, 8#37, 1, 2, 3, 8#24, 1, 3

0 = DH+ port of the KT communications card in the workstation

8#37 = octal DH+ node of the 1756-DHRIO module in slot 2 of the local chassis

1 = backplane port of the 1756-DHRIO module in slot 2 of the local chassis

2 = slot number of the other 1756-DHRIO module in the local chassis

3 = Channel B of the 1756-DHRIO module in slot 1 of the local chassis, configured for DH+

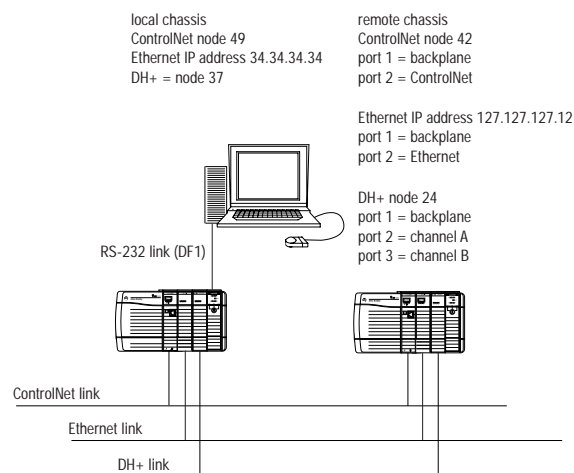
8#24 = DH+ node of the 1756-DHRIO module in slot 2 of the remote chassis

1 = backplane port of the 1756-DHRIO module in slot 2 of the remote chassis

3 = slot number of the controller in the remote chassis

ControlNet Use several network connections across different network bridges.

- ControlNet to the remote chassis
- Ethernet back to the local chassis
- DH+ back to the remote chassis



Select a DF1 driver (to handle worst case performance)

Enter connection path: 1, 0, 2, 42, 1, 1, 2, 34.34.34.34, 1, 2, 2, 8#24, 1, 3

1 = backplane port of the Logix5550 controller in slot 3 of the local chassis

0 = slot number of the 1756-CNB module in the local chassis

2 = ControlNet port of the 1756-CNB module in slot 0 of the local chassis

42 = ControlNet node of the 1756-CNB module in slot 0 of the remote chassis

1 = backplane port of the 1756-CNB module in slot 0 of the remote chassis

1 = slot number of the 1756-ENET module in the remote chassis

2 = Ethernet port of the 1756-ENET module in slot 1 of the remote chassis

34.34.34.34 = IP address of the 1756-ENET module in slot 1 of the local chassis

1 = backplane port of the 1756-ENET module in slot 1 of the local chassis

2 = slot number of the 1756-DHRIO module in the local chassis

2 = Channel A of the 1756-DHRIO module in slot 2 of the local chassis, configured for DH+

8#24 = DH+ node of the 1756-DHRIO module in slot 2 of the remote chassis

1 = backplane port of the 1756-DHRIO in slot 2 of the remote chassis

3 = slot number of the controller in the remote chassis

Notes:

Integrating Motion

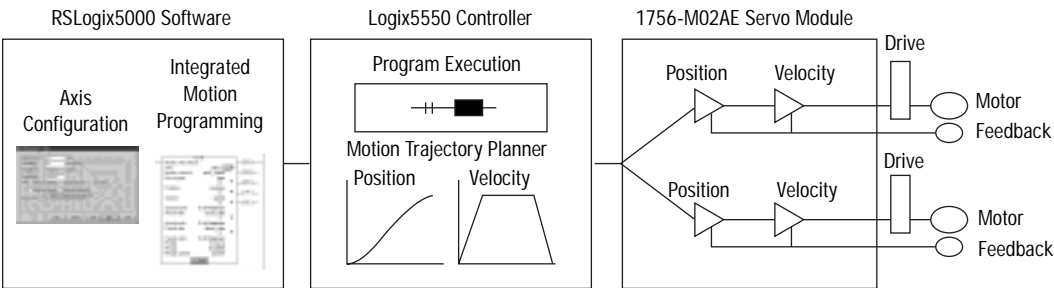
Using This Chapter

For information about:	See page:
Developing a Motion Control Application Program	10-2
Writing a Motion Application Program	10-12

Introduction

The Logix5550 controller, 1756-M02AE servo module, and RSLogix5000 programming software provide integrated motion control support.

- The Logix5550 controller contains a high-speed motion task, which executes the ladder motion commands and generates position and velocity profile information. This profile information is sent to one or more 1756-M02AE servo modules. Several Logix5550 controllers can be used in each chassis. Each controller and chassis can control up to 16 1756-M02AE servo modules.
- The 1756-M02AE servo module connects to a servo drive and closes a high-speed position and velocity loop. Each Logix5550 controller can support up to 16 1756-M02AE servo modules. Each 1756-M02AE module can control up to two axes.
- RSLogix 5000 programming software provides complete axis configuration and motion programming support.



41383

Developing a Motion Control Application Program

Developing a motion control application program involves:

Step:	Description:	See page:
Select the master controller for coordinated system time	Set one controller as the master controller. Once you complete this step, you can synchronize all the motion modules and Logix5550 controllers in your chassis.	10-2
Add a servo module	Add a motion module to your application program.	10-3
Name an axis	Add an axis to your application program.	10-4
Configure an axis	Configure each axis for motion control.	10-5
Run hookup diagnostics and auto tuning	Complete hookup diagnostics and auto tuning for each axis.	10-11
Develop a motion application program	Create a program for your motion control application.	10-12

The following sections provide an overview of each of these steps. For more information about completing these steps, see the *ControlLogix Motion Module User Manual*, publication 1756-6.5.16.

Selecting the master controller for coordinated system time

Important: Only one controller in a chassis can be the CST master.

To select the master controller for coordinated system time, open the controller properties window and select the Date/Time tab.

To open the controller properties window:

1. Place the cursor over the Controller folder.
2. Click the right mouse button and select Properties.



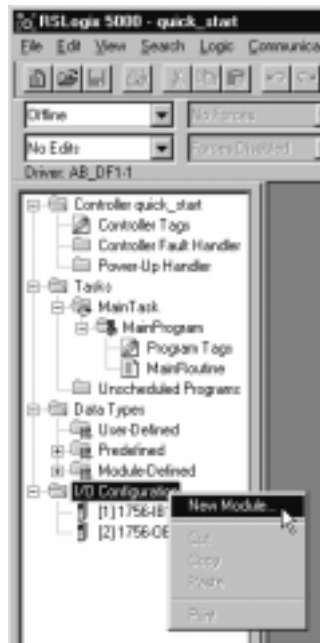
If:	And:	Then:
Your controller uses a motion axis	No other controllers in your chassis are configured as the master controller	<ul style="list-style-type: none"> • Select Make this controller the master • Select OK
Your controller uses a motion axis	Another controller in your chassis is configured as the master controller	Select OK

Adding a 1756-M02AE module

To add a servo module, open the new module window and select a 1756-M02AE module.

To open the new module window:

1. Select I/O Configuration.
2. Click the right mouse button and select New Module.



In the module properties window, specify this information:

A screenshot of the 'Module Properties - Local (1756-M02AE 1.1)' dialog box. The fields are: Type: 1756-M02AE 2-Axis Analog/Encoder Servo; Vendor: Allen-Bradley Company, Inc.; Parent: Local; Name: (empty); Slot: 6; Description: (empty); Associated Axes: Channel 0: <none>, Channel 1: <none>; Revision: 1; Electronic Keying: Compatible Module. Buttons at the bottom are: Cancel, < Back, Next >, Finish >>, and Help.

In this field:	Enter:
Name	Enter the name of the servo module.
Description	Enter a description of the servo module (optional).
Slot	Enter the slot number where the module is installed.
Revision	Enter the revision number for this module. Depending on the electronic keying option you choose, the module checks the revision number to ensure that the physical module matches the configured module.
Electronic Keying	Select an electronic keying method.

Naming an axis

To name an axis, click New Axis in the module properties window.

Make sure you have entered
a servo module name.

Specify this information:

In this field:

Enter:

Name

Enter the name of the axis.

Description

Enter a description of the axis (optional).

Configuring a servo axis

To configure your new axis:

1. Click Configure in the new tag window.

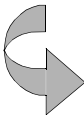
Make sure you have entered an axis name.

The 'New Tag' dialog box is shown. It has fields for Name, Description, Tag Type (Base, Alias, Grouped), Data Type, Scope, and Style. The 'Configure...' button is visible next to the Data Type field.

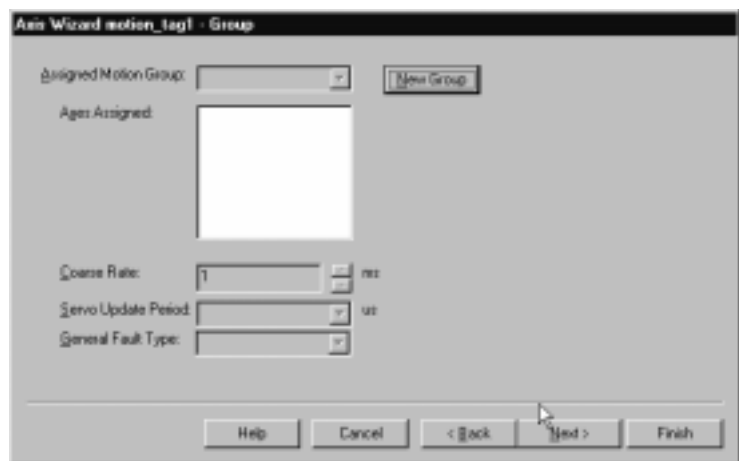
2. Select the type of axis and axis positioning.

The 'Axis Wizard motion_tag1 - General' dialog box is shown. It has fields for Module, Channel, Type, and Positioning Mode. The 'Type' is set to 'Servo' and the 'Positioning Mode' is set to 'Linear'. Navigation buttons (Help, Cancel, < Back, Next >, Finish) are at the bottom.

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



Click Next.
Go to step 3.

3. Assign a motion group.**If:**

You want to create a new motion group

Then:

Go to step 4

You want to use an existing motion group

Go to step 6

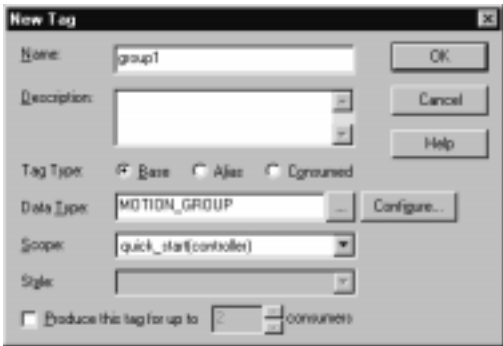
4. Create a new motion group, click New Group.

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. (For more information on the MOTION_GROUP tag, see appendix C Structures in the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.)

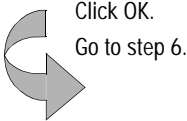


5. Specify this information:

Make sure you enter a group name. →



In this field:	Enter:
Name	Enter the name of the motion group.
Description	Enter a description of the motion group (optional).



Click OK.
Go to step 6.

6. Assign the axis to a motion group and specify this information:

Select the motion group. →

Axis Wizard motion_tag1 - Group

Assigned Motion Group: group1

New Group

Axis Assigned: motion_tag1

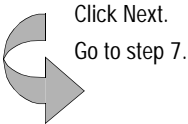
Coarse Rate: 1 ms

Servo Update Period: 200 us

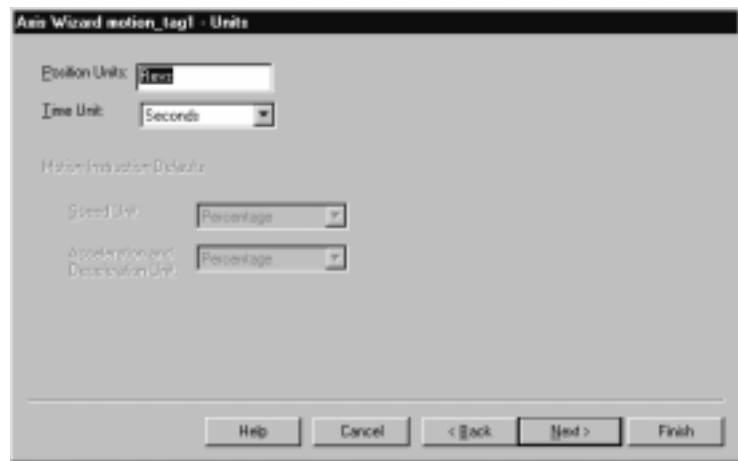
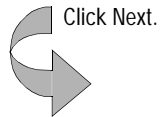
General Fault Type: Non Major Fault

Help Cancel < Back Next > Finish

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



7. Define units.

The image shows a software dialog box titled "Axis Wizard motion_tag1 - Units". It contains several input fields and dropdown menus. The "Position Units" field is set to "Inch". The "Time Unit" dropdown menu is set to "Seconds". Under the "Motion Instruction Defaults" section, the "Speed Unit" dropdown is set to "Percentage" and the "Acceleration and Deceleration Unit" dropdown is also set to "Percentage". At the bottom of the dialog, there are five buttons: "Help", "Cancel", "< Back", "Next >", and "Finish".

8. To continue configuring your axis, complete the entries in each Axis Wizard window. To move to the next window, click Next.

Important: The Axis Wizard will gray-out the online diagnostic testing and auto tuning options until your controller is online. Before going online, complete the configuration of all your servo modules and download your application program.

Important: There are several Axis Wizard windows. When you are finished configuring the axis, click Finish.

9. Assign the axis to a channel.



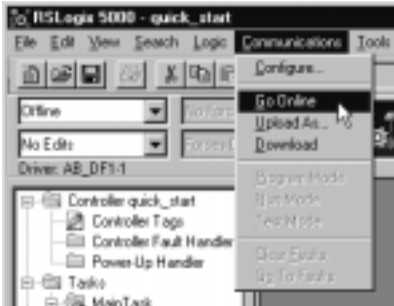
If:	Then:
You want to assign your axis to channel 0	In the <i>Channel 0</i> field, select your axis from the drop-down menu
You want to assign your axis to channel 1	In the <i>Channel 1</i> field, select your axis from the drop-down menu
You want to add another axis	Click New Axis. See page 10-4.
You do not want to add another axis	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 auto tuning

Once you have added and configured your modules and axes, you can download your program. After going online, you can complete hookup diagnostics and auto tuning.

1. Download your project.

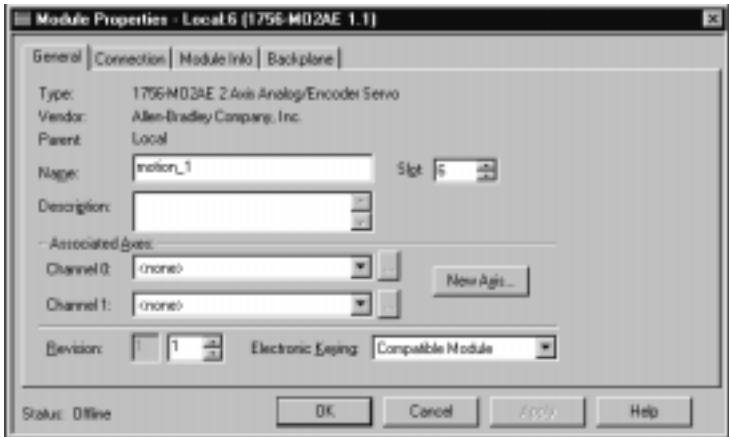


Your program can be a blank program, but it must include complete configuration information for all your modules and axes.

2. In the module properties window, select the channel that you assigned to the axis.

To open the module properties window:

1. Select the servo module.
2. Click the right mouse button and select Properties.



If:	Then:
You assigned your axis to channel 0	Select the ... button next to Channel 0
You assigned your axis to channel 1	Select the ... button next to Channel 1

3. Select the Hookup tab and run the hookup diagnostics.
4. Select the Tune Servo tab and run auto tuning.
5. When diagnostic testing and auto tuning are complete, click OK.

For more information about hookup diagnostics, see the *ControlLogix Motion Module User Manual*, publication 1756-6.5.16.

Writing a Motion Application Program

To write a motion application program, you can insert motion instructions directly into your ladder logic program. The motion instruction set consists of:

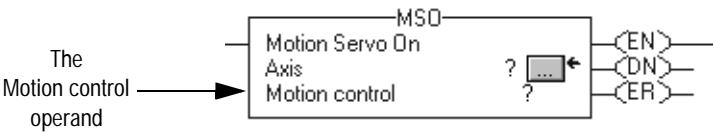
Group:	Instructions:
Motion state instructions	<ul style="list-style-type: none"> • Motion Servo On (MSO) • Motion Servo Off (MSF) • Motion Axis Shutdown (MASD) • Motion Axis Shutdown Reset (MASR) • Motion Direct Drive On (MDO) • Motion Direct Drive Off (MDF) • Motion Axis Fault Reset (MAFR)
Motion move instructions	<ul style="list-style-type: none"> • Motion Axis Stop (MAS) • Motion Axis Home (MAH) • Motion Axis Jog (MAJ) • Motion Axis Move (MAM) • Motion Axis Gearing (MAG) • Motion Change Dynamics (MCD) • Motion Redefine Position (MRP)
Motion group instructions	<ul style="list-style-type: none"> • Motion Group Stop (MGS) • Motion Group Program Stop (MGPS) • Motion Group Shutdown (MGSD) • Motion Group Shutdown Reset (MGSR) • Motion Group Strobe Position (MGSP)
Motion event instructions	<ul style="list-style-type: none"> • Motion Arm Watch (MAW) • Motion Disarm Watch (MDW) • Motion Arm Registration (MAR) • Motion Disarm Registration (MDR)
Motion configuration instructions	<ul style="list-style-type: none"> • Motion Apply Axis Tuning (MAAT) • Motion Run Axis Tuning (MRAT) • Motion Apply Hookup Diagnostics (MAHD) • Motion Run Hookup Diagnostics (MRHD)

These instructions operate on one or more axes. You must identify and configure axes before you can use them. For more information about configuring axes, see the *ControlLogix Motion Module User Manual*, publication 1756-6.5.16.

For more information on individual motion instructions, see the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Understanding the MOTION_INSTRUCTION tag

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 motion control operand in other instructions can cause unintended operation of the control variables.

For more information about the MOTION_INSTRUCTION tag, see appendix C Structures in the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Using motion status and configuration parameters

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

Method:	Example:
Directly accessing the MOTION_GROUP and AXIS structures	<ul style="list-style-type: none">• Axis faults• Motion status• Servo status
Using the GSV instruction	<ul style="list-style-type: none">• Actual position• Command position• Actual velocity

For more information on these methods, see the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Modifying motion configuration parameters

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 *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Handling motion faults

Two types of motion faults exist.

Type	Description	Example
Errors	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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 *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

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. For more information about handling error codes, see chapter 11.

Understanding a programming example

The following figure shows several rungs of a motion control application program.

Rung 0:

Enables the Feed and Cut axes when you press the servo_on button.

Rung 1:

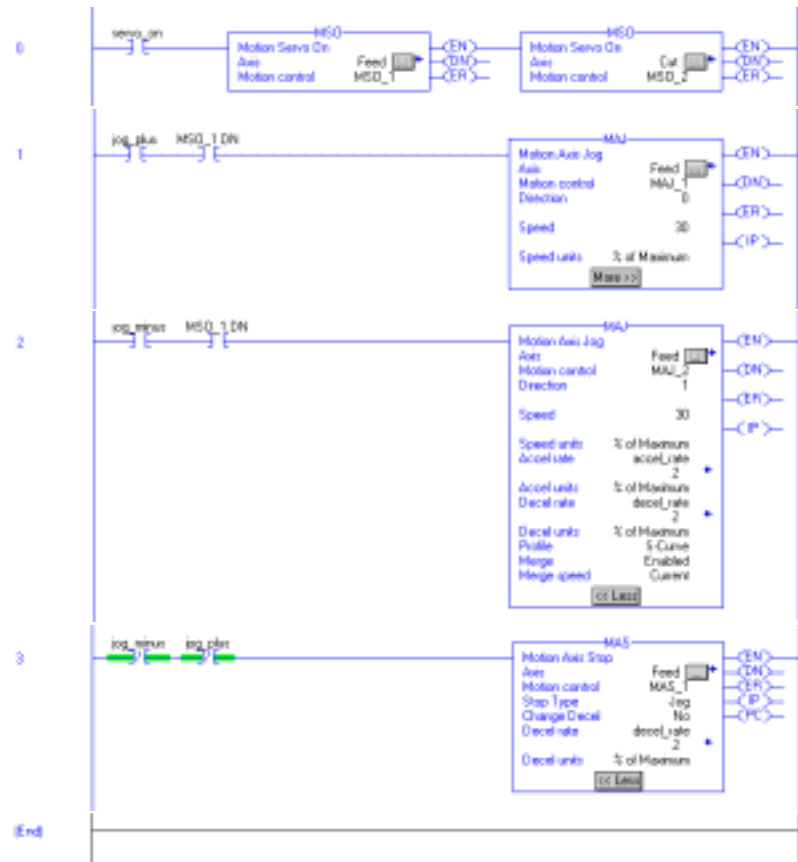
Jogs the Feed axis in the positive direction when you press the jog_plus button.

Rung 2:

Jogs the Feed axis in the reverse direction when you press the jog_minus button.

Rung 3:

Stops the Feed axis when you release with the jog_plus button or the jog_minus button.



Notes:

Forcing I/O

Using This Chapter

For information about:	See page:
Forcing	11-1
Entering forces	11-2
Enabling forces	11-4
Disabling forces	
Removing forces	11-5
Monitoring forces	11-6

Forcing

Forcing lets you override an I/O module's values in the controller. You can force:

- a structure member of an I/O tag

Since an I/O tag is a structured tag, the force is applied to its structure members (of type BOOL, SINT, INT, DINT, or REAL). You can force all I/O data, except for configuration data.

- an alias to an I/O structure member (of type BOOL, SINT, INT, DINT, or REAL)

Forcing an input value overrides the actual input value being received from the controller, but will not affect the value received by other controllers monitoring that physical input module. Forcing an input value overrides the value regardless of the state of the physical input module.

Forcing an output value overrides the logic for the physical output module. Other controllers monitoring that output module in a listen-only capacity will also see the forced value.

Forces are applied to the actual modules at the end of every program scan when data arrives at the module.

Important: Forcing increases logic execution time. The more values you force, the longer it takes to execute the logic.

Important: Forces are held by the controller and not by the programming workstation. Forces remain even if the programming workstation is disconnected.



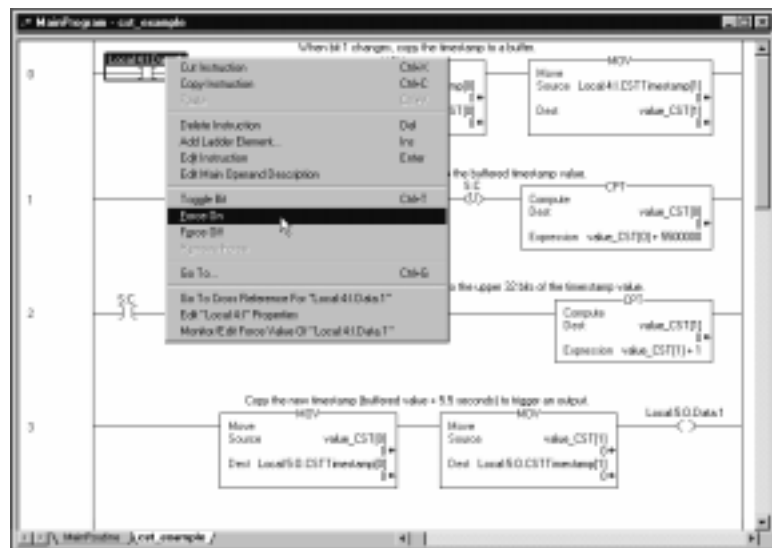
ATTENTION: If forces are enabled and anything is forced, keep personnel away from the machine area. Forcing can cause unexpected machine motion that could injure personnel.

If you want to:	Do this:
force a whole SINT, INT, DINT, or REAL value	To force a whole value, type a force value in the Force Mask column, using a decimal, octal, hexadecimal, or float/exponential format. For a REAL value, you must use a float/exponential format. To remove a force for a whole value, type a space.
force bits within a value	To force an individual bit in a SINT, INT, or DINT value, expand the value and edit the Force Mask column. The force value is displayed in binary style, where: <ul style="list-style-type: none"> • "0" indicates force off • "1" indicates force on • "." indicates no force You can also use the bit pallet to select a bit to force.
force a BOOL	To force a BOOL, enter the force value, where: <ul style="list-style-type: none"> • "0" indicates force off • "1" indicates force on To remove a force, type a space.

Entering forces from the ladder editor

From the ladder editor, you can set forces only for BOOL tags or integer bit values used in bit instructions.

Right-click on the BOOL tag or bit value.
Select Force On, Force Off, or Remove Force.



For forced values in the more complex instructions, you can only remove forces. You must use the data monitor to set force values for these values.

Right-click on the forced value.
Select Remove Force.



Enabling Forces

Once you set which values or bits to force, you enable forces for the force values to take affect. You can only enable and disable forces at the controller level. You cannot enable or disable forces for a specific module, tag collection, or tag element.



ATTENTION: Enabling forces causes input and output values to change. Keep personnel away from the machine area. Forcing can cause unexpected machine motion that could injure personnel.

You enable forces from the Online Bar.

Forces Installed indicates that force values have been entered.

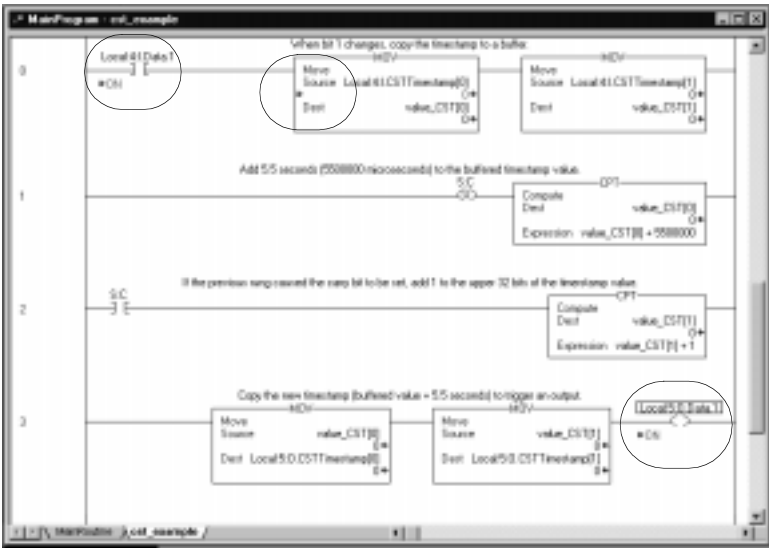
Select Enable all forces.



Important: If you download a project that has forces enabled, the programming software prompts you to enable or disable forces after the download completes.

When forces are enabled, a > appears next to the forced value in the ladder editor.

When forces are enabled, the ladder editor indicates which forces are on.



Disabling Forces

You can disable forces without removing forces from individual values or from the controller. By disabling forces, the project can execute as programmed. Forces are still entered, but they are not executed.

Select Disable all forces.



Removing Forces

You can remove forces from individual values or from the entire controller.

You can remove individual forces from the data monitor.

If you want to remove a force from a:	Do this:
whole SINT, INT, DINT, or REAL value	Right-click on the value in the data monitor and select Remove Force.
bits within a value	Expand the value and edit the Force Mask column. Change the bit value to "." to indicate no force.
BOOL value	Type a space.

If the force is on a BOOL tag or bit value, you can also remove forces from the ladder editor. Right-click on the value and select Remove Force.

If you remove each force individually, forces can still be enabled.

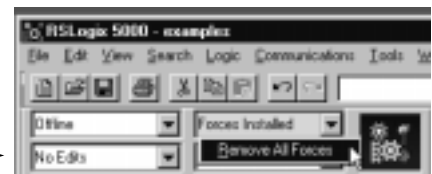


ATTENTION: If you have removed forces, but forces are still enabled and you set a force value, it takes affect immediately. Keep personnel away from the machine area. Forcing can cause unexpected machine motion that could injure personnel.

Removing a force on an alias tag also removes the force on the base tag.

At the controller level, you can remove all forces. Removing all forces disables forces and clears all force mask values.

Select Remove all forces. →



Monitoring Forces

Important: The only way to monitor force status is through the programming software or from logic. The Logix5550 controller does not have a LED to indicate force status.

The following example shows how to check whether forces are present and enabled and set your own LED indicator.



Handling Controller Faults

Using This Chapter

For information about:	See page:
Understanding controller faults	12-1
Viewing controller faults	12-2
Monitoring I/O faults	12-2
Handling hardware faults	12-3
Processing minor faults	12-3
Minor fault types and codes	12-8
Processing major faults	12-9
Major fault types and codes	12-14
Creating a program fault routine	12-16
Creating a controller fault handler	12-16
Accessing the FAULTLOG	12-20

Understanding Controller Faults

The controller detects three main categories of faults. In general:

If the controller detects a:	It means:	The controller:
major fault	A fault condition, either hardware or instruction, occurred. The fault condition is severe enough for the controller to shut down, unless the condition is cleared.	<ol style="list-style-type: none"> 1. Sets a major fault bit 2. Runs user-supplied fault logic, if it exists 3. If the user-supplied fault logic cannot clear the fault, the controller goes to faulted mode 4. Sets outputs according to their output state during program mode 5. OK LED flashes red
minor fault	A fault condition, either hardware or instruction occurred. The fault condition is not severe enough for the controller to shut down.	<ol style="list-style-type: none"> 1. Sets a minor fault bit 2. Continues with the program scan 3. no LEDs change state
hardware fault	A fault occurred with the controller hardware. The controller shuts down. You must repair or replace the controller.	<p>Sets outputs according to their output state during fault mode</p> <p>The controller OK LED is solid red.</p>

Viewing Controller Faults

The programming software displays fault information.

A. Place the cursor over the Controller quick_start folder.

B. Click the right mouse button and select Properties



Select the Major Faults tab or the Minor Faults tab to view current fault information.



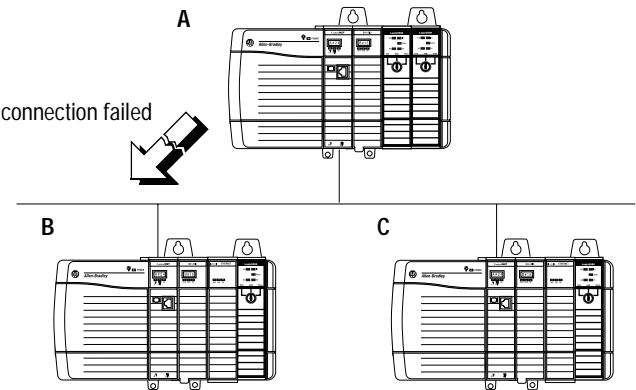
Monitoring I/O Faults

Each I/O module has status bits that indicate when a fault occurs. Your control application should monitor these status bits. If any faults exist, your application should take appropriate action, such as shutting down the system in a controlled manner.

If the controller detects a fault with one of its I/O modules, the programming software displays a yellow attention symbol (!) over the device and the I/O Configuration folder in the controller organizer. You can also view I/O faults on the connection tab of the module properties.

For more information, see *Viewing Module Fault Information* on page 3-19.

You can configure the controller so that the controller generates a major fault if it loses its connection with an I/O rack or module. If you do not configure the controller for this possibility, you should monitor the device status within your logic. If the connection between a rack or module and the controller is lost and the controller is not configured to generate a major fault, all outputs dependent on inputs from the failed connection continue to be controlled based on the now static input. The control application continues to make control decisions on data that may or may not be correct.



The connection between chassis A and B failed, so data in chassis B remains at its last values. Any outputs in chassis C that are controlled by inputs from chassis B are based on stale data.

Handling Hardware Faults

If you encounter a hardware fault:

1. Power down then power up the controller.
2. Reload the program.
3. Run the program again.

If you continue to encounter a hardware fault, call your Allen-Bradley representative.

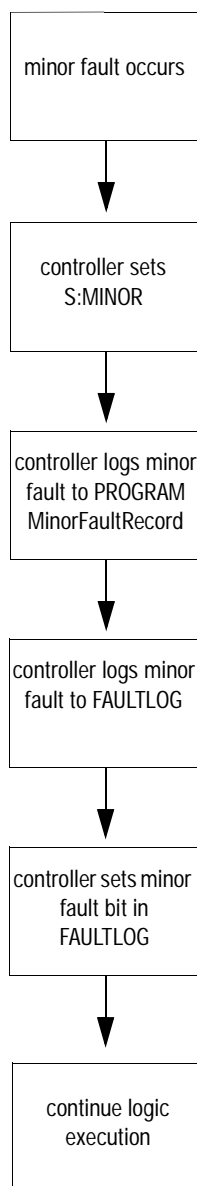
Processing Minor Faults

Minor faults do not impact controller operation. The controller continues to execute. However, to optimize execution time and ensure program accuracy, you should identify and correct minor faults.

There are two main categories of minor faults:

Category:	Description:
instruction execution	problem occurs when executing logic
other	minor problem occurs with the: <ul style="list-style-type: none">• serial port• battery

Processing instruction-execution minor faults



When an instruction-execution minor fault occurs, the controller logs the minor fault information to the current PROGRAM object. Then the controller logs the minor fault information to the FAULTLOG object, but this fault information is mainly historical. Use the PROGRAM fault information for accurate, current fault information.

Writing logic for instruction-execution minor faults

To check for an instruction-execution minor fault, follow these steps:

1. Create a user-defined structure to store the fault information.
This can be the same structure you use for major fault information. The format must be as follows (you can change the structure and member names, but the data types and sizes must be the same as shown below):

Name: Size: byte(s)

Description:

Members:

	Name	Data Type	Style	Description
	TimeLow	DINT	Decimal	Low 32 bits of fault timestamp value
	TimeHigh	DINT	Decimal	Upper 32 bits of fault timestamp value
	Type	INT	Decimal	Fault Type (Program, I/O, etc)
	Code	INT	Decimal	Unique (by type) Code for the fault
	Info	DINT[8]	Hex	Fault Specific information - content varies by type and code
*				

2. Monitor S:MINOR to determine when a minor fault occurs.

The S:MINOR flag is a status bit that is set if at least one minor fault has been generated. The controller sets this bit when a minor fault occurs due to program execution. The controller does not set this bit for minor faults that are not related to program execution, such as battery low.

3. Use a GSV instruction to get the MINORFAULTRECORD of the current program (THIS). The destination should be a tag of the user-defined structure type you specified above.

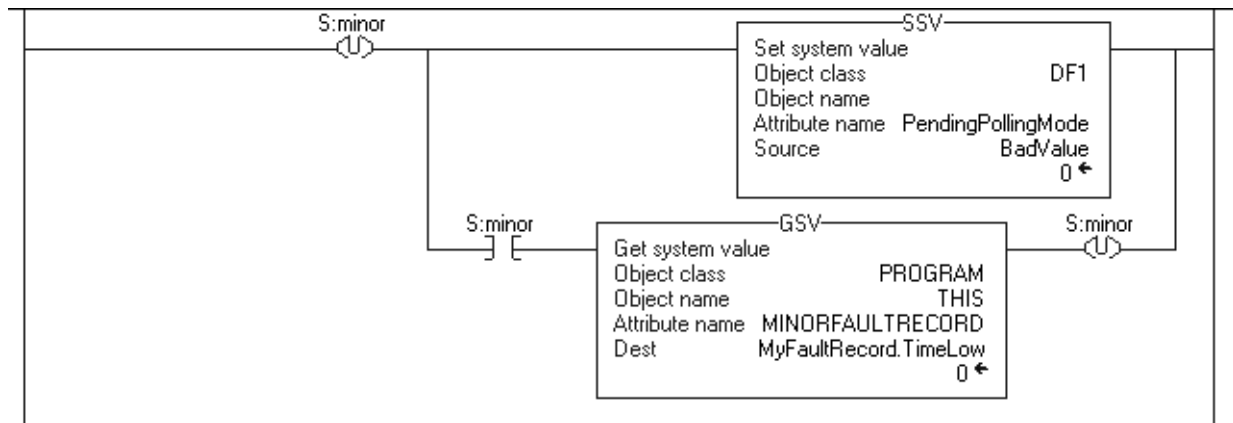
In the GSV instruction, MINORFAULTRECORD is an attribute of the PROGRAM object class. The object name is the name of the PROGRAM. Or you can enter THIS, which specifies the PROGRAM that contains the GSV instruction.

4. Take appropriate action to respond to the minor fault (typically, correct the logic error).

You do not need to clear an instruction-execution minor fault. However, the S:MINOR bit remains set until the end of the logic scan. If you need to detect multiple minor faults in a single scan, reset S:MINOR with an OTU instruction.

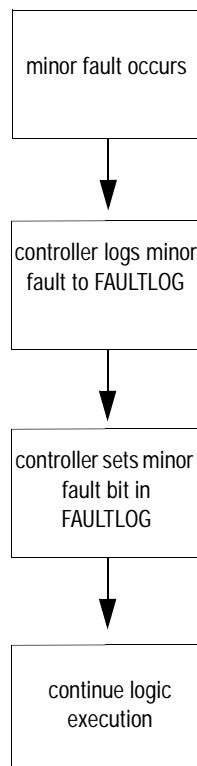
The following logic shows one way to check for an instruction-execution minor fault. Place this logic within a routine in a program (not in the fault routine).

checking for an instruction-execution minor fault



This example monitors S:MINOR to determine if a minor fault occurs with the execution of the SSV instruction. You could replace this SSV instruction with any instruction or operation that you want to check to see whether it generates a minor fault (like checking for an overflow condition with a math instruction). The GSV instruction then retrieves the fault information and stores it in a tag that uses the structure type you defined. The Destination tag must point to the first DINT of the structure (MyFaultRecord.TimeLow in this example).

Processing other minor faults



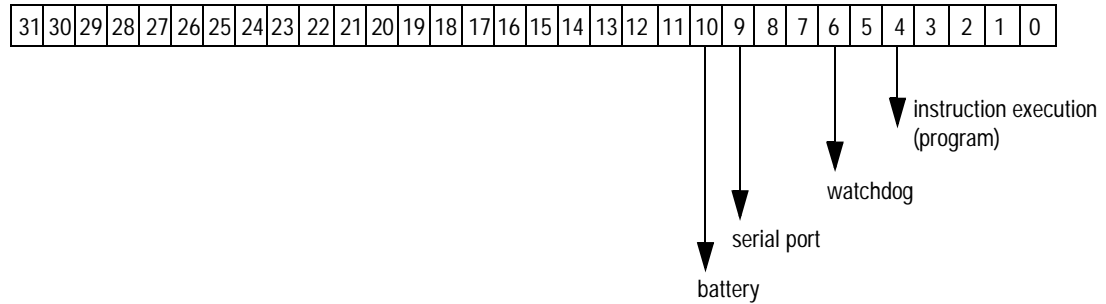
When a minor fault occurs, the controller logs the minor fault information directly to the FAULTLOG object.

Writing logic for other minor faults

To check for other minor faults, follow these steps:

1. Create a DINT to hold the MinorFaultBits record from the FAULTLOG object.

MinorFaultBits record in the FAULTLOG object

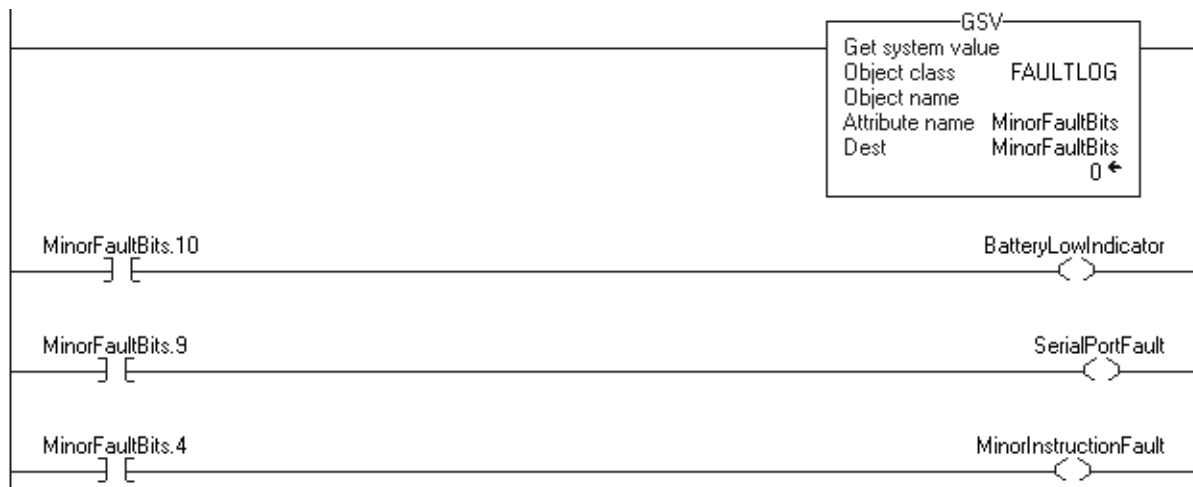


2. Use a GSV instruction to get the MinorFaultBits record of the FAULTLOG object. The Destination should be the DINT tag you created.
3. Examine the fault bits to determine the type of fault and take appropriate action.

You do not need to clear a minor fault.

The following logic shows one way to check for minor faults, other than instruction-execution faults. Place this logic within a routine within the program (not the fault routine).

checking for other minor faults



This example uses a GSV instruction to get a copy of the MinorFaultBits record of the FAULTLOG and store it in a DINT tag *MinorFaultBits*. Then this example examines some of the bits in *MinorFaultBits* to see what type of fault occurred.

Minor Fault Types and Codes

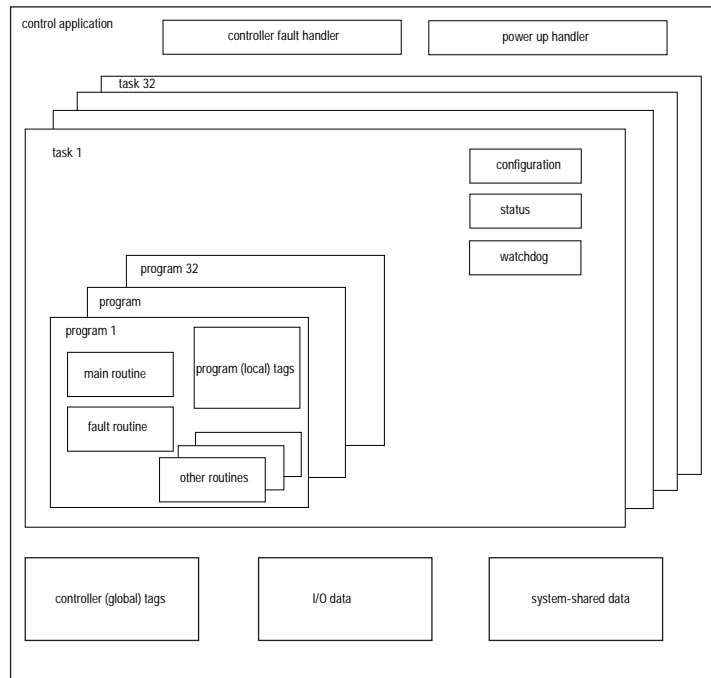
The minor fault list includes:

Type:	Code:	Cause:	Recovery Method:
4	4	An arithmetic overflow occurred in an instruction.	Fix program by examining arithmetic operations (order) or adjusting values.
4	7	The GSV/SSV destination tag was too small to hold all of the data.	Fix the destination so it has enough space.
4	35	PID delta time ≤ 0 .	Adjust the PID delta time so that it is > 0 .
4	36	PID setpoint out of range	Adjust the setpoint so that it is within range.
6	2	Periodic task overlap. Periodic task has not completed before it is time to execute again.	Simplify program(s), or lengthen period, or raise relative priority, etc.
9	0	Unknown error while servicing the serial port.	Contact GTS personnel.
9	1	The CTS line is not correct for the current configuration.	Disconnect and reconnect the serial port cable to the controller. Make sure the cable is wired correctly
9	2	Poll list error. A problem was detected with the DF1 master's poll list, such as specifying more stations than the size of the file, specifying more than 255 stations, trying to index past the end of the list, or polling the broadcast address (STN #255).	Check for the following errors in the poll list: <ul style="list-style-type: none"> • total number of stations is greater than the space in the poll list tag • total number of stations is greater than 255 • current station pointer is greater than the end of the poll list tag • a station number greater than 254 was encountered
9	5	DF1 slave poll timeout. The poll watchdog has timed out for slave. The master has not polled this controller in the specified amount of time.	Determine and correct delay for polling.
9	9	Modem contact was lost. DCD and/or DSR control lines are not being received in proper sequence and/or state.	Correct modem connection to the controller.
10	10	Battery not detected or needs to be replaced.	Install new battery.

Processing Major Faults

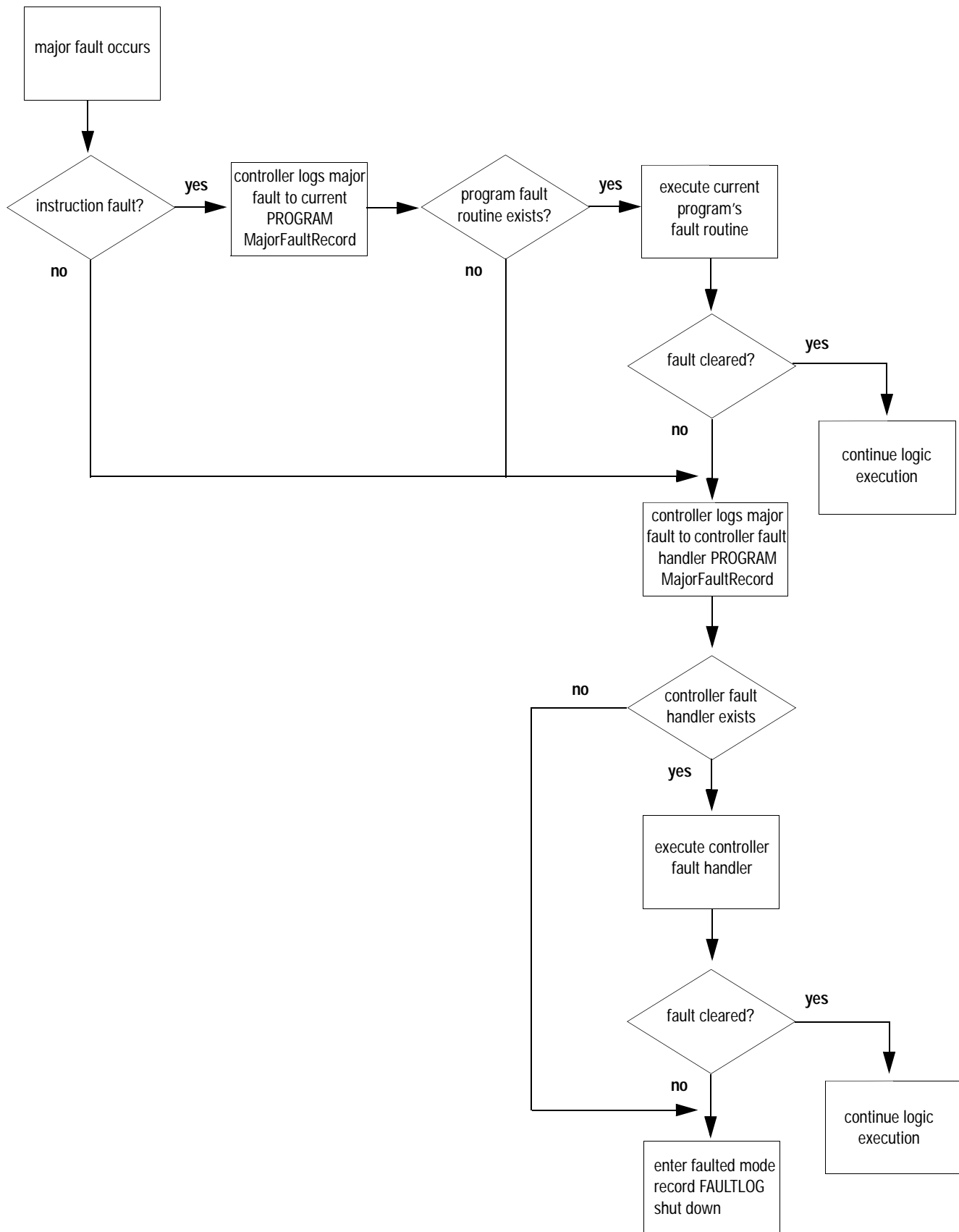
The controller supports two levels for handling major faults:

- program fault routine
- controller fault handler



Each program can have its own fault routine. The controller executes the program's fault routine when an instruction fault occurs. If the programs' fault routine does not clear the fault, or a program fault routine does not exist, the controller proceeds to execute the controller fault handler (if defined). If the controller fault handler does not exist or cannot clear the major fault, the controller enters faulted mode and shuts down. At this point, the FAULTLOG is updated. (See the next page.)

All non-instruction faults (I/O, task watchdog, etc.) execute the controller fault handler directly (no program fault routine is called).



There are two main categories of major faults:

Category:	Description:
instruction execution	problem occurs when executing logic
other	major problem occurs with the: <ul style="list-style-type: none">• power loss• I/O• task watchdog• mode change• motion axis

The multitasking capability of the controller makes it possible for multiple major faults to be reported. For example, multiple task watchdog timeouts can occur at the same time or I/O faults can be reported at the same time as an instruction execution fault occurs. In these cases, major faults are processed in the order that they occurred.

You can use the controller fault handler to clear a watchdog fault. If the same watchdog fault occurs a second time during the same logic scan, the controller enters faulted mode, regardless of whether the controller fault handler clears the watchdog fault.

If any of the multiple reported major faults are not cleared by the controller fault handler, the controller goes to faulted mode. The fault that was not cleared, and up to two additional faults that have not been cleared, are logged in the controller fault log. You can view this fault information via the programming software by using the major fault tab in the controller properties.

The controller can handle as many as 32 simultaneous major faults. If more than 32 major faults occur at the same time, the controller goes to faulted mode and the first three major faults are logged to the controller fault log.

Writing logic for a major fault

To check and clear a major fault, follow these steps:

1. Depending on the type of major fault, do one of the following:

If you are writing logic for this type of major fault:

Do this:

instruction execution

Create a routine within the current program and specify this routine as the fault routine for the program. See page 12-16.

any other

Create a new program and select this program as the controller fault handler program. See page 12-16.

2. Create a user-defined structure to store the fault information. This can be the same structure you use for minor fault information. The format must be as follows (you can change the structure and member names, but the data types and sizes must be the same as shown below):

Name: Size: byte(s)

Description:

Members:

	Name	Data Type	Style	Description
	TimeLow	DINT	Decimal	Low 32 bits of fault timestamp value
	TimeHigh	DINT	Decimal	Upper 32 bits of fault timestamp value
	Type	INT	Decimal	Fault Type (Program, I/O, etc)
	Code	INT	Decimal	Unique (by type) Code for the fault
	Info	DINT[8]	Hex	Fault Specific information - content varies by type and code
*				

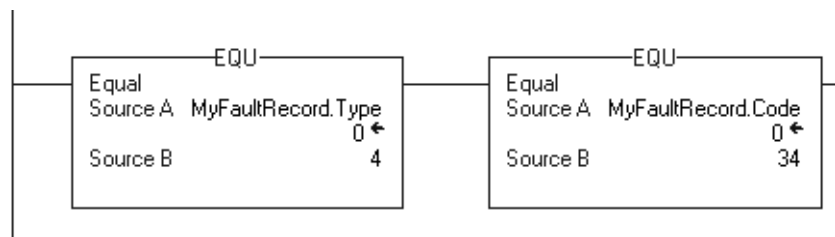
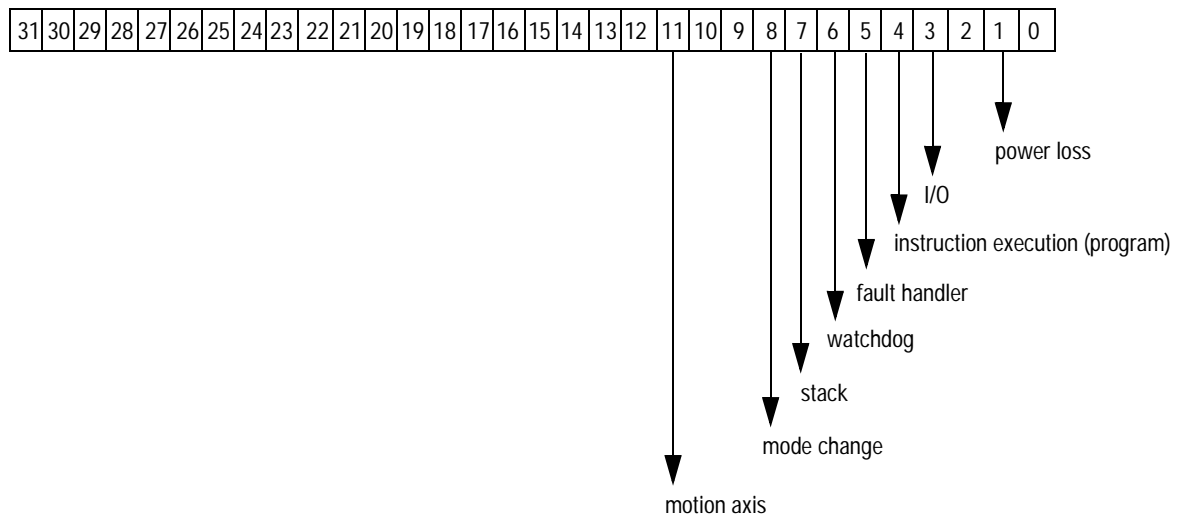
3. Use a GSV instruction to get the MAJORFAULTRECORD of the current program (THIS). The destination should be a tag of the user-defined structure type you created.

GSV	
Get system value	
Object class	PROGRAM
Object name	THIS
Attribute name	MAJORFAULTRECORD
Dest	MyFaultRecord.TimeLow 0 ←

The Destination tag must point to the first member of the structure (*MyFaultRecord.TimeLow*). This tag is of the structure type you define to hold fault information.

4. Examine the fault type and code to determine which fault occurred and take appropriate action.

MajorFaultBits record in the FAULTLOG object

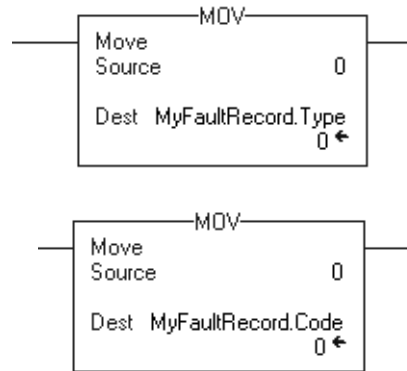


Examine the *MyFaultRecord.Type* and *MyFaultRecord.Code* members to determine the type of major fault. This example looks for specific fault types and codes.

5. Take appropriate action. Develop your own logic to respond to the major fault.

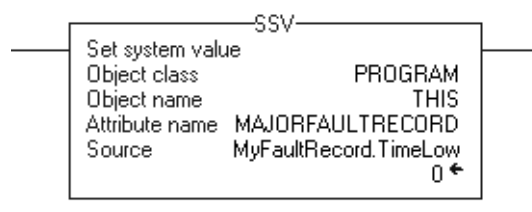
6. If you decide to clear the fault:

- A. Clear the type and code members of the tag (*MyFaultRecord.Type* and *MyFaultRecord.Code* shown above).



Use MOV instructions to clear the type and code values.

- B. Use an SSV instruction to clear the fault by copying the tag (*MyFaultRecord.TimeLow*) to the MajorFaultRecord of the PROGRAM object.



The Source tag must point to the first member of the structure (*MyFaultRecord.TimeLow*). This tag is of the structure type you define to hold fault information.

You can also clear a major fault by using the keyswitch on the controller. Turn the keyswitch to Prog, then to Run, and then back to Prog.

Major Fault Types and Codes

The major fault list includes:

Type:	Code:	Cause:	Recovery Method:
1	1	The controller powered on in Run mode.	Execute the power-loss handler.
3	16	A required I/O module connection failed.	Check that the I/O module is in the chassis. Check electronic keying requirements. View the controller properties Major Fault tab and the module properties Connection tab for more information about the fault.
3	20	Possible problem with the ControlBus chassis.	Not recoverable - replace the chassis.
3	23	At least one required connection was not established before going to Run mode.	Wait for the controller I/O light to turn green before changing to Run mode.

Type:	Code:	Cause:	Recovery Method:
4	16	Unknown instruction encountered.	Remove the unknown instruction. This probably happened due to a program conversion process.
4	20	Array subscript too big, control structure .POS or .LEN is invalid.	Adjust the value to be within the valid range. Don't exceed the array size or go beyond dimensions defined.
4	21	Control structure .LEN or .POS < 0.	Adjust the value so it is > 0.
4	31	The parameters of the JSR instruction do not match those of the associated SBR or RET instruction.	Pass the appropriate number of parameters. If too many parameters are passed, the extra ones are ignored without any error.
4	34	A timer instruction has a negative preset or accumulated value.	Fix the program to not load a negative value into timer preset or accumulated value.
4	42	JMP to a label that did not exist or was deleted.	Correct the JMP target or add the missing label.
4	83	The data tested was not inside the required limits.	Modify value to be within limits.
4	84	Stack overflow.	Reduce the subroutine nesting levels or the number of parameters passed.
6	1	Task watchdog expired. User task has not completed in specified period of time. A program error caused an infinite loop, or the program is too complex to execute as quickly as specified, or a higher priority task is keeping this task from finishing.	Increase the task watchdog, shorten the execution time, make the priority of this task "higher," simplify higher priority tasks, or move some code to another controller.
8	1	Attempted to place controller in Run mode with keyswitch during download.	Wait for the download to complete and clear fault.
11	1	Actual position has exceeded positive overtravel limit.	Move axis in negative direction until position is within overtravel limit and then execute Motion Axis Fault Reset.
11	2	Actual position has exceeded negative overtravel limit.	Move axis in positive direction until position is within overtravel limit and then execute Motion Axis Fault Reset.
11	3	Actual position has exceeded position error tolerance.	Move the position within tolerance and then execute Motion Axis Fault Reset.
11	4	Encoder channel A, B, or Z connection is broken.	Reconnect the encoder channel then execute Motion Axis Fault Reset.
11	5	Encoder noise event detected or the encoder signals are not in quadrature.	Fix encoder cabling then execute Motion Axis Fault Reset.
11	6	Drive Fault input was activated.	Clear Drive Fault then execute Motion Axis Fault Reset.
11	7	Synchronous connection incurred a failure.	First execute Motion Axis Fault Reset. If that doesn't work, pull servo module out and plug back in. If all else fails replace servo module.
11	8	Servo module has detected a serious hardware fault.	Replace the module.
11	9	Asynchronous Connection has incurred a failure.	First execute Motion Axis Fault Reset. If that doesn't work, pull servo module out and plug back in. If all else fails replace servo module.
11	32	The motion task has experienced an overlap.	The group's course update rate is too high to maintain correct operation. Clear the group fault tag, raise the group's update rate, and then clear the major fault.

Creating a Program Fault Routine

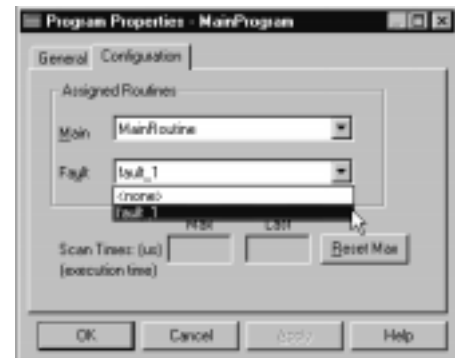
Each program can have one fault routine. You specify the fault routine when you configure the program. You can only change the fault routine by using the programming software to change the program configuration.

1. Select a program ("MainProgram" in this example).
2. Click the right mouse button and select Properties.



To specify a fault routine:

Select the fault routine. →



Creating the Controller Fault Handler

The controller fault handler is an optional task that executes when the major fault is not an instruction-execution fault or the program fault routine:

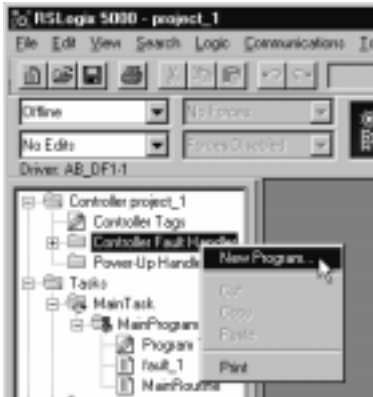
- could not clear the major fault.
- faulted.
- does not exist.

The controller fault handler can have only one program. That one program can have multiple routines.

To configure a controller fault program, either create a program for the controller fault handler or select an unscheduled program. For more information about creating programs and routines, see chapter 5.

Creating a program for the controller fault handler

1. Select the ControllerFaultHandler.
2. Click the right mouse button and select New Program.



To create a program as the controller fault program:



In this field:	Enter:
Name	Enter the name of the program.
Description	Enter a description of the program (optional).
Type	The type defaults to System Fault.

Naming programs

Program names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

Program names are not case sensitive.

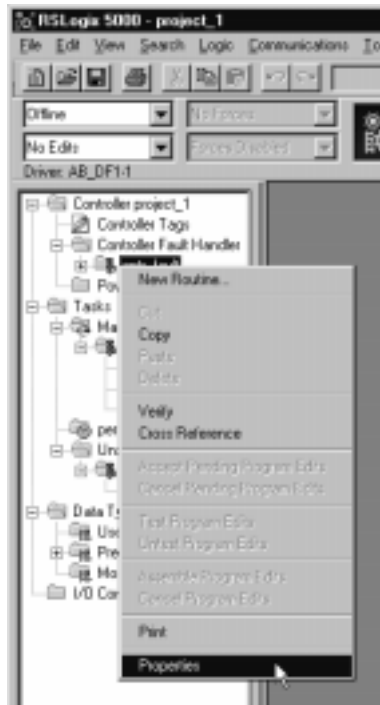
Selecting an unscheduled program for the controller fault handler

1. Select the program ("unscheduled_1 in this example).
2. Drag and drop the program to the controller fault handler.



To select an unscheduled program, drag and drop the unscheduled program into the controller fault handler folder. If a controller fault program already exists, the unscheduled program takes its place. The previous controller fault program moves to the unscheduled programs folder.

1. Select a program ("cntr_fault" in this example).
2. Click the right mouse button and select Properties.

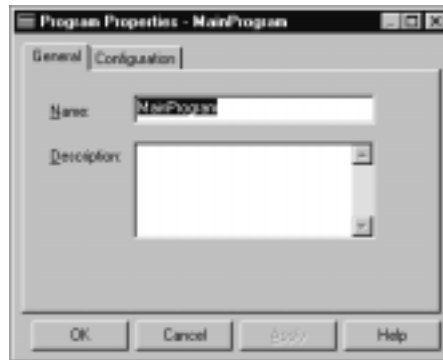


Configuring programs

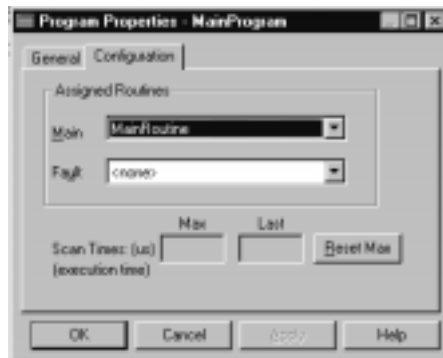
Once you create a controller fault program, there are other properties that you need to configure. You must have a main routine.

The controller fault program does not execute a fault routine. If you specify a fault routine for the controller fault program, the controller never executes that routine.

To configure an existing program:



On this tab:	In this field:	Enter:
General	Name	The programming software displays the current name of the program. Edit the name, if necessary.
	Description	The programming software displays the current description. Edit the description, if necessary.



On this tab:	In this field:	Enter:
Configuration	Assigned Routine	The programming software displays the name of the Main Routine and the Fault Routine, if any. Change the selections, if necessary.
	Scan Time (µs)	While online, the programming software displays the maximum scan time and the last scan time in µsec for the current program. These values are execution times for the program and do not include any time spent waiting for other programs or higher-priority tasks. These values are display only.

Creating routines

1. Select a program ("cntr_fault" in this example).

2. Click the right mouse button and select New Routine.

You can create multiple routines for the controller fault program. One routine must be configured as the main routine for the program. This routine can call other routines.

To create a controller fault routine:



In this field:	Enter:
Name	Enter the name of the routine.
Description	Enter a description of the routine (optional).
Type	Select the programming language used to create the routine. Ladder is the default.
In Program	Leave this selection alone. It automatically defaults to the controller fault program.

Naming routines

Routine names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

Routine names are not case sensitive.

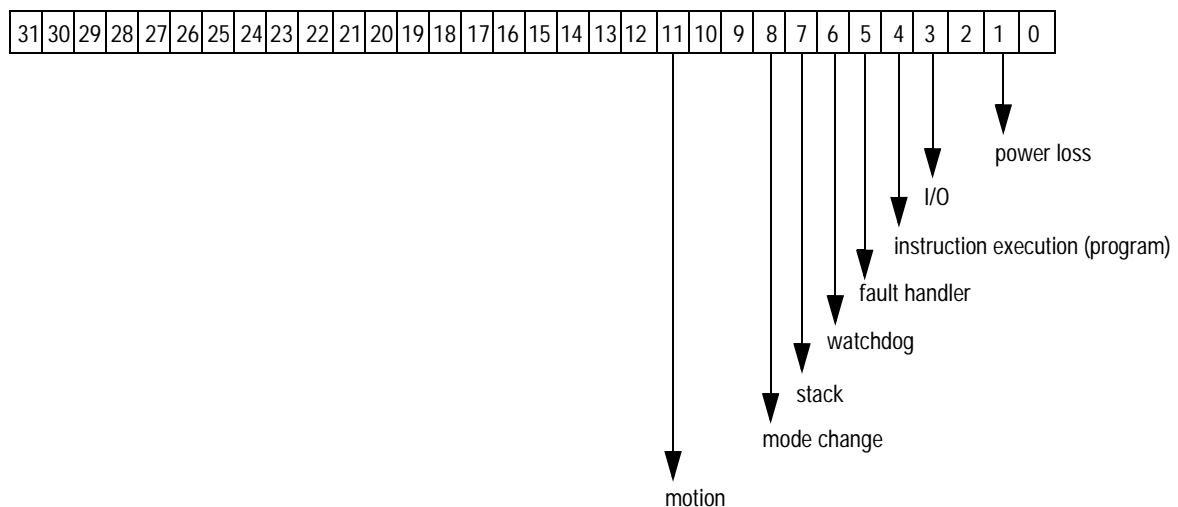
Accessing the FAULTLOG

You access controller status through the GSV/SSV instructions. Status information is stored in objects. One such object is the FAULTLOG. The FAULTLOG contains:

Attribute:	Instruction:	Description:
MajorEvents	GSV SSV	How many major faults have occurred since the last time this counter was reset.
MinorEvents	GSV SSV	How many minor faults have occurred since the last time this counter was reset.
MajorFaultBits	GSV SSV	Individual bits indicate the type of major fault.
MinorFaultBits	GSV SSV	Individual bits indicate the type of minor fault.

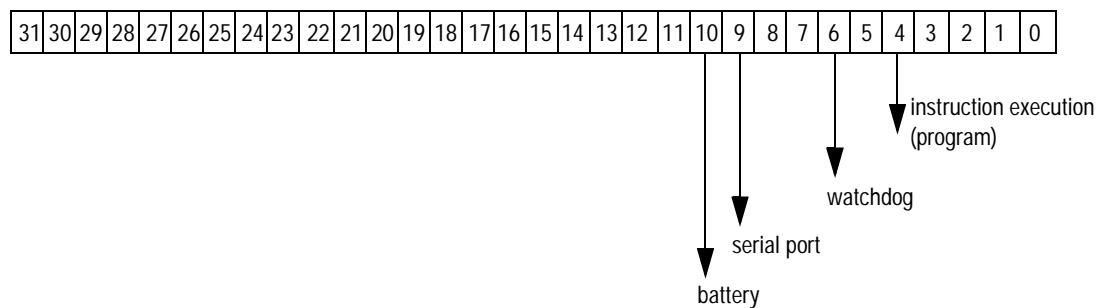
MajorFaultBits structure

The MajorFaultBits record in the FAULTLOG identifies the last major fault by setting the bit that corresponds to the fault type.



MinorFaultBits structure

The MinorFaultBits record in the FAULTLOG identifies the last minor fault by setting the bit that corresponds to the fault type.



Preparing a Power-Up Program

Using This Chapter

For information about:	See page:
How the controller powers up in Run mode	13-1
Creating the power-up handler	13-3
Clearing the major fault	13-6

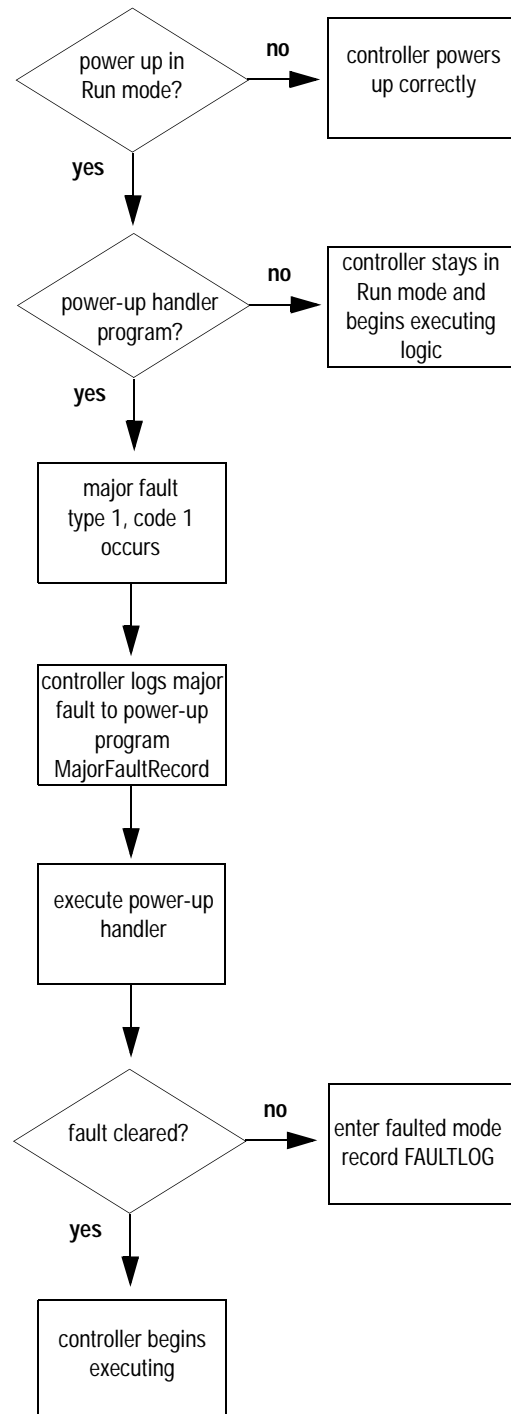
How the Controller Powers Up in Run Mode

You can use the power-up handler to determine how the controller powers up in Run mode.

If you use the power-up handler (a program is defined), and the controller powers up in Run mode, the controller sets major fault type 1, code 1. The power-up handler must clear the major fault for the controller to operate normally. Otherwise, the controller enters its faulted mode (shuts down).

If you do not use the power-up handler (no program is defined), the controller stays in Run mode if it powers up in Run mode. The controller begins executing logic.

Processing the power-up handler



Creating the Power-Up Handler

1. Select the PowerUpHandler.
2. Click the right mouse button and select New Program.



The power-up handler is an optional task that executes when the controller powers up in Run mode. The power-up handler can have only one program. That one program can have multiple routines.

To configure a power-up program, either create a program for the power-up handler or select an unscheduled program. For more information about creating programs and routines, see chapter 4.

Creating a program for the power-up handler

To create a program as the power-up program:



In this field:	Enter:
Name	Enter the name of the program.
Description	Enter a description of the program (optional).
Type	The type defaults to Power Up.

Naming programs

Program names follow IEC 1131-3 identifier rules and:

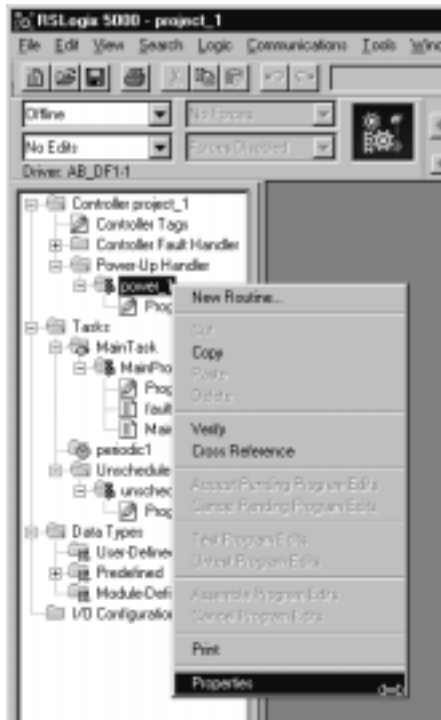
- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (__)

Program names are not case sensitive.

- 1. Select the program ("unscheduled_1 in this example).
- 2. Drag and drop the program to the power-up handler.



- 1. Select a program ("power_1" in this example).
- 2. Click the right mouse button and select Properties.



Selecting an unscheduled program for the power-up handler

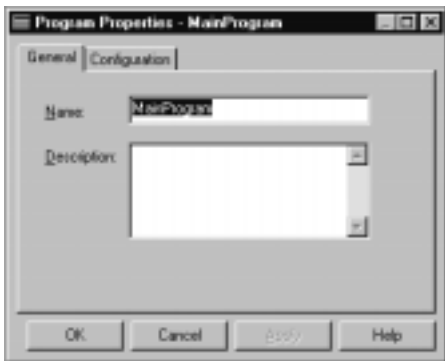
To select an unscheduled program, drag and drop the unscheduled program into the power-up handler folder. If a power-up program already exists, the unscheduled program takes its place. The previous power-up program moves to the unscheduled programs folder.

Configuring programs

Once you create a power-up program, there are other properties that you need to configure. You must have a main routine.

The power-up program does not execute a fault routine. If you specify a fault routine for the power-up program, the controller never executes that routine or the controller fault handler.

To configure an existing program:



On this tab:	In this field:	Enter:
General	Name	The programming software displays the current name of the program. Edit the name, if necessary.
	Description	The programming software displays the current description. Edit the description, if necessary.



On this tab:	In this field:	Enter:
Configuration	Assigned Routine	The programming software displays the name of the Main Routine and the Fault Routine, if any. Change the selections, if necessary.
	Scan Time (μs)	While online, the programming software displays the maximum scan time and the last scan time in μsec for the current program. These values are execution times for the program and do not include any time spent waiting for other programs or higher-priority tasks. These values are display only.

Creating routines

1. Select a program ("power_1" in this example).
2. Click the right mouse button and select New



You can create multiple routines for the power-up program. One routine must be configured as the main routine for the program. This routine can call other routines.

To create a power-up routine:



In this field:	Enter:
Name	Enter the name of the routine.
Description	Enter a description of the routine (optional).
Type	Select the programming language used to create the routine. Ladder is the default.
In Program	Leave this selection alone. It automatically defaults to the power-up program.

Naming routines

Routine names follow IEC 1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)

Routine names are not case sensitive.

Clearing the Major Fault

To check and clear the major fault, follow these steps:

1. Create a user-defined structure to store the fault information. The format must be as follows (you can change the structure and member names, but the data types and sizes must be the same as shown below):

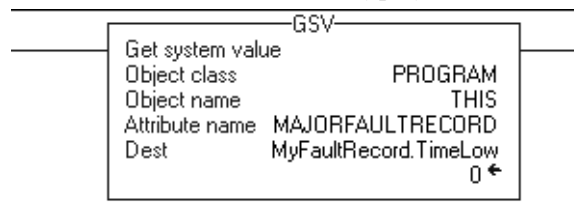
Name: Size: byte(s)

Description:

Members:

	Name	Data Type	Style	Description
	TimeLow	DINT	Decimal	Low 32 bits of fault timestamp value
	TimeHigh	DINT	Decimal	Upper 32 bits of fault timestamp value
	Type	INT	Decimal	Fault Type (Program, I/O, etc)
	Code	INT	Decimal	Unique (by type) Code for the fault
	Info	DINT[8]	Hex	Fault Specific information - content varies by type and code
*				

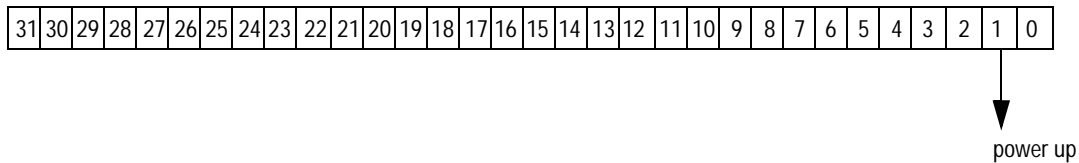
2. Use a GSV instruction to get the MAJORFAULTRECORD of the power-up program (THIS). The destination should be a tag of the user-defined structure type you created.



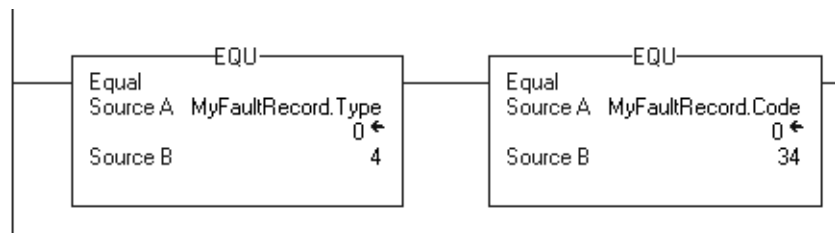
The Destination tag must point to the first member of the structure (*MyFaultRecord.TimeLow*). This tag is of the structure type you define to hold fault information.

3. Examine the fault type and code to determine which fault occurred and take appropriate action.

MajorFaultBits record in the FAULTLOG object

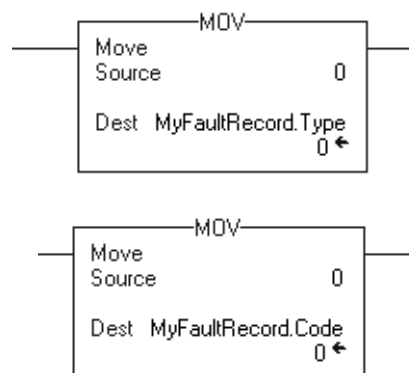


Because you are executing the power-up handler, both the *MyFaultRecord.Type* and *MyFaultRecord.Code* members are 1.



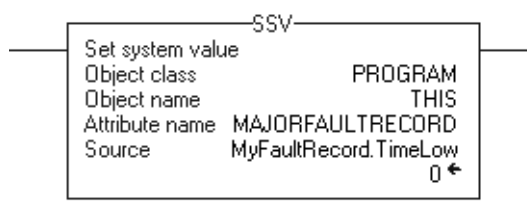
Examine the *MyFaultRecord.Type* and *MyFaultRecord.Code* members to determine the type of major fault. This example looks for specific fault types and codes.

4. Take appropriate action. Develop your own logic to respond to the major fault.
5. If you decide to clear the fault:
 - A. Clear the type and code members of the tag (*MyFaultRecord.Type* and *MyFaultRecord.Code* shown above).



Use MOV instructions to clear the type and code values.

- B.** Use an SSV instruction to clear the fault by copying the tag (MyFaultRecord.TimeLow) to the MajorFaultRecord of the PROGRAM object.



The Source tag must point to the first member of the structure (*MyFaultRecord.TimeLow*). This tag is of the structure type you define to hold fault information.

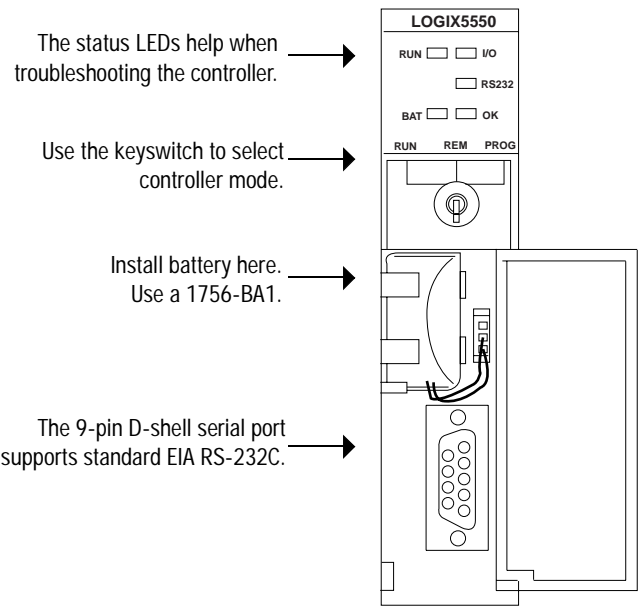
Troubleshooting

Using This Appendix

For information about:	See page:
Identifying controller components	A-1
Monitoring controller status LEDs	A-2
Monitoring controller status	A-5
Changing controller mode	A-8
Examining controller prescan operations	A-9

Identifying Controller Components

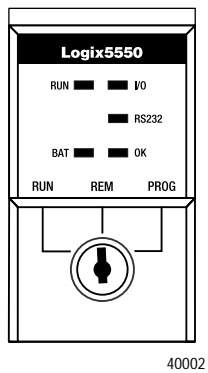
The controller has these components on the front panel:



40001

Monitoring Controller Status LEDs

The LEDs on the front panel show these states:




Indicator:	Color:	Description:
RUN	off	• controller in Program or Test mode
	green	• controller is in Run mode
I/O	off ¹	• no I/O or communications configured
	green	• communicating to all configured devices
	green flashing	• one or more configured devices are not responding
	red flashing	• not communicating to any devices • controller faulted
RS232	off	no activity
	green flashing	• data being received or transmitted
	red	• controller faulted
BAT	off	• battery will support memory • battery will not support memory
	red	• no battery present <i>replace the battery</i>
	off	• no power applied
OK	red flashing	• recoverable fault
	red	• controller faulted <i>clear faults, clear memory, or replace the controller</i>
	green	• controller OK

1. If the controller does not contain a project (controller memory is empty), the I/O indicator will be off.

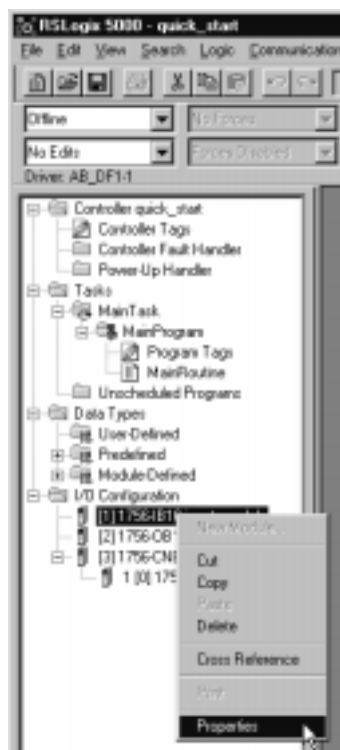
Determining which modules are not responding

If the I/O status indicator is flashing green or flashing red, you can use the programming software to determine which I/O devices are not responding.

Each I/O module provides an indication when a fault occurs. The programming software displays this fault information:

In this location:	The software displays:
Controller organizer	The I/O configuration portion displays the modules configured for the controller. If the controller detects an error condition or fault with one of these modules, the controller organizer displays a yellow attention symbol () over the device.
Connection tab from module properties	<p>The module fault field displays the fault code returned to the controller (related to the module) and the text detailing the fault.</p> <p>Common categories for module errors are:</p> <p>Connection request error The controller is attempting to make a connection to the module and has received an error. The connection was not made.</p> <p>Service request error The controller is attempting to request a service from the module and has received an error. The service was not performed successfully.</p> <p>Module configuration rejected The configuration in the module is invalid. This commonly happens when a second controller tries to share ownership of an input module, but the module configuration does not match the configuration already in the input module.</p> <p>Module key mismatch Electronic keying is enabled and some part of the keying information differs between the software and the module.</p>
Module Info tab from module properties	This tab displays module and status information about the module. You can also reset a module to its power-up state. You must be online to use this tab.
Backplane tab from module properties	This tab displays diagnostic information about the module's communications over the backplane and the chassis in which it is located. You can also clear module faults and reset the transmit retry limit.

1. Select a module ("1756-IB16" in this example)
2. Click the right mouse button and select Properties



From the Module Properties tabs you can view and edit:

On this tab:	In this field:	Enter:
General	Name	The programming software displays the current name of the module.
	Description	The programming software displays the current description.
	Slot Number	The programming software automatically displays the current slot number.
	Communication Format	The programming software displays the current communication format.
	Electronic Keying	The programming software displays the current electronic keying requirement.
Connection	This tab provides module connection fault data. Use this tab to get more detailed information about why a particular module connection is faulted.	
	Requested Packet Interval	The programming software displays the current RPI setting.
	Inhibit Module	The programming software displays whether or not the module is inhibited.
	Major Fault	The programming software displays whether or not the controller generates a major fault if the connection to this module fails.
	The programming software displays product and status information about the module. There are no fields to select or enter data.	
Module Info	The identification information and the match status are useful when diagnosing electronic keying problems.	
	The configured and owned status is useful for diagnosing multiple owner or multiple listener problems.	
	The error and status information provides information from the point of view of the I/O modules.	
Configuration	Contains module-specific configuration information. The available fields depend on the module. For example, this tab for the 1756-IB16 module has fields for enable change of state and input filter time settings.	
Backplane	The programming software displays backplane status information. There are no fields to select or enter data. You can clear faults and reset the status counters.	

Monitoring Controller Status

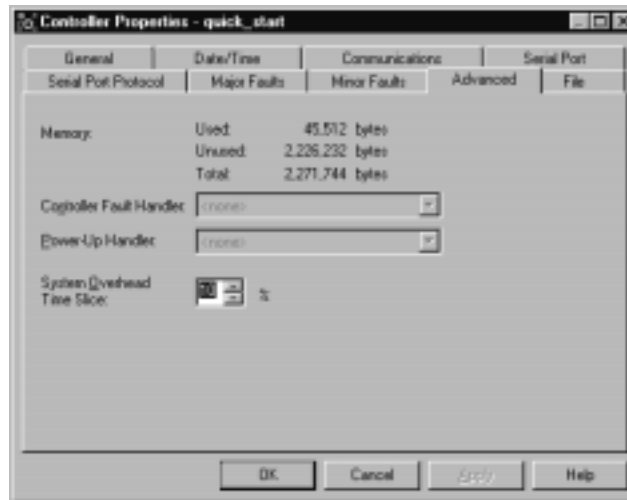
1. Place the cursor over the controller folder (Controller quick_start in this example).
2. Click the right mouse button and select Properties.



The Logix5550 controller offers different levels of status information that you can view through the programming software and access through logic.

Viewing status through the programming software

The controller properties dialog displays controller configuration information for the open project, and when online, for the attached controller.



From the controller properties, you can edit and view this information:

From this tab:	You can:
General	modify the controller name, description, and controller properties for the current project
Date/Time	view and edit the controller's wall clock time and the coordinated system time status.
Communications	configure communication information that is stored with the project file
Serial Port	view and configure the serial port on the controller
Serial Port Protocol	configure the serial port for: <ul style="list-style-type: none"> • DF1 point-to-point • DF1 slave • DF1 master
Major Faults	view any major faults that have occurred on the controller
Minor Faults	view any minor faults that have occurred on the controller
Advanced	view and edit advanced controller properties, which include the system fault program, the power loss program, and system overhead time slice
File	view information about the project file

Monitoring status flags

The controller supports status keywords you can use in your logic to monitor specific events.

Keyword:	Status Flag:	Description:
S: V	overflow	Overflow is set if the value you are storing cannot fit into the destination. Either the value is greater than the maximum value for the destination or the value is less than the minimum value for the destination. Important: Each time S:V goes from cleared to set, it generates a minor fault (type 4, code 4)
S:Z	zero	Zero is set if the instruction's destination value is 0.
S:N	sign (result is negative)	Sign is set if the instruction's destination value is negative.
S:C	carry	The carry flag is not actually a part of the data type. The carry flag represents the bit that would be in the data type if it were stored to a larger data type.
S:FS	first scan	The first scan bit is set if this is the first, normal scan of the routines in the current program.
S:MINOR	minor fault	The minor fault bit is set if at least one minor fault has been generated. The controller sets this bit when a minor fault occurs due to program execution. The controller does not set this bit for minor faults that are not related to program execution, such as battery low.
THIS	current item	The THIS statement is only valid with the GSV and SSV instructions that refer to a TASK, PROGRAM, or ROUTINE. Use THIS to specify the currently executing TASK, PROGRAM, or ROUTINE.

The status keywords are not case sensitive.

Because the status flags can change so quickly, the status keywords are not animated in the programming software to actual show status.

You **cannot** define a tag alias to a keyword.

Using GSV/SSV instructions

The GSV/SSV instructions get and set controller system data that is stored in objects. The controller stores system data in objects. There is no status file, as in the PLC-5 processor.

When enabled, the GSV instruction retrieves the specified information and places it in the destination. When enabled, the SSV instruction sets the specified attribute with data from the source.

When you enter a GSV/SSV instruction, the programming software displays the valid object classes, object names, and attribute names for each instruction. For the GSV instruction, you can get values for all the available attributes. For the SSV instruction, the software displays only those attributes you can modify.



ATTENTION: Use the SSV instruction carefully. Making changes to objects can cause unexpected controller operation or injury to personnel.

You can access these objects:

This object	Provides status information about:
AXIS	a servo module axis
CONTROLLER	a controller's execution
CONTROLLERDEVICE	the physical hardware of a controller
CST	coordinated system time for the devices in one chassis
DF1	the DF1 communication driver for the serial port
FAULTLOG	fault information for a controller
MESSAGE	peer-to-peer communications
MODULE	a module
MOTIONGROUP	a group of axes for the servo module
PROGRAM	a program
ROUTINE	a routine
SERIALPORT	the serial communication port
TASK	a task
WALLCLOCKTIME	a timestamp the controller can use for scheduling

For more information, see the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

Changing Controller Mode

Use the keyswitch to change the mode in which the controller operates:

If you want to:

- Run your program

Outputs are enabled. Equipment being controlled by the I/O addressed in the program begins operating.

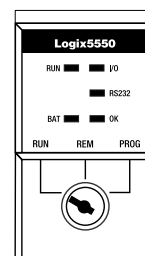
- Enable outputs

You cannot create or delete tasks, programs, or routines. You cannot create or delete tags or edit online while in Run mode.

You cannot change the controller mode through the programming software while the keyswitch is in the RUN position.

Turn the keyswitch to:

RUN



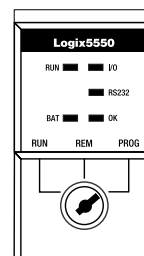
40003

- Disable outputs (outputs are set to their configured program states)
- Create, modify, and delete tasks, programs, or routines
- Download projects

The controller does not execute (scan) tasks.

You cannot change the controller mode through the programming software while the keyswitch is in the PROG position.

PROG



40004

Change between Remote Program, Remote Test, and Remote Run modes through the programming software.

REM

Remote Run

- Enable outputs
- Edit online (limited)

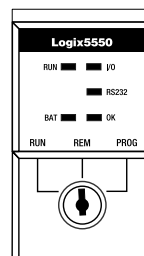
Remote Program

- Disable outputs (outputs are set to their configured states)
- Create, modify, and delete tasks, programs, or routines
- Download projects
- Edit online

The controller does not execute (scan) tasks.

Remote Test

- Execute tasks with outputs disabled
- Edit online (limited)



40002

Examining Controller Prescan Operations

If unexpected operation occurs when the controller enters Run mode, make sure to examine the prescan operation of the instructions. Some instructions execute differently during prescan than they do during a normal scan. For details on how each instruction operates during prescan, see the *Logix5550 Controller Instruction Set Reference Manual*, publication 1756-6.4.1.

The prescan operation is an intermediate scan between the transition from Program mode to Run mode. The prescan examines all programs and instructions and initializes data based on the results.

For example, a routine that is called infrequently might have a bad indirect address that generates a major fault. It might take several normal program scans before this fault occurs. Prescan provides the opportunity for the controller to examine the program for errors such as this before changing to Run mode.

Instructions with unique prescan operations

During prescan, the controller modifies data associated with some instructions (see the following table). The following table describes prescan operations that deviate from normal instruction operation.

Instruction:	Executes these actions during prescan:
CTU	The .CU/.CD bit is set to prevent a false count when the first Run-mode scan begins.
CTD	
DTR	The reference value is updated (regardless of the rung condition).
FFL	The .EL bit is set to prevent a false load when the first Run-mode scan begins.
LFL	
FFU	The .EU bit is set to prevent a false unload when the first Run-mode scan begins.
LFU	
FOR	Ladder instructions within the loop are prescanned.
JSR	The subroutine is invoked and prescanned. If recursive calls are made to the subroutine, the subroutine is only prescanned the first time it is called.
ONS	The programmed bit address of the instruction is set to inhibit false triggering when the first Run-mode scan begins.
OSR	
OSF	The programmed bit address of the instructions is reset to inhibit false triggering when the first Run-mode scan begins.
SQL	The .EN bit is set to prevent a false increment of the position when the first Run-mode scan begins.
SQO	
TOF	The .TT bit is reset and the .ACC is set to equal the .PRE.

See the *Logix5550 Controller Instruction Set Reference*, publication 1756-6.4.1 for specific details on how each instruction is prescanned.

Recovering from prescan errors

If an indirect reference is used by one of these instructions and the pointer to this reference is initialized at run time, there is a chance that an error might occur during prescan.

Use a program fault routine to trap the prescan error and reset the error so the controller can continue with the prescan process. The following example shows a sample program and fault handler. The example logic uses this fault record structure:

Data Type: FaultRecordType

Warning: This structure is being referenced. Modifications will result in loss of data.

Name: Size: byte(s)

Description:

Members:

	Name	Data Type	Style	Description
<input type="checkbox"/>	TimeLow	DINT	Decimal	
<input type="checkbox"/>	TimeHigh	DINT	Decimal	
<input type="checkbox"/>	Type	INT	Decimal	
<input type="checkbox"/>	Code	INT	Decimal	
<input type="checkbox"/>	Info	DINT[8]	Decimal	
<input checked="" type="checkbox"/>	*			

The example logic also uses a DINT array named TABLE with 10 elements (TABLE[10]).

Main Routine:

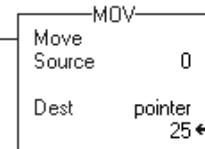
****ATTENTION: THIS RUNG MUST BE PRESENT TO ENSURE THE FAULT ROUTINE ONLY TRAPS THE CONDITION DURING PRESCAN****
 Because this rung is unconditional, the bit will always be set while the program is running. When the CPU is switched from program to run, the prescan resets all bits referenced by OTE instructions. The status from this bit is used by the fault routine to determine if the fault occurred during prescan or during the normal program scan (as long as the fault routine is set up properly). This is useful for recovering from prescan errors that can occur when tags used as pointers indirectly reference arrays that have not been initialized yet.

CPU_Scanning

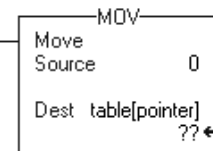


After the first scan you can force this value beyond the array limits and cause the controller to major fault and shutdown.

S:FS



This rung performs an indirect reference using the pointer above. Note that this rung does not fault the controller during prescan because instructions like this do not change tag values during prescan. Once the controller has completed the prescan, this instruction will cause a fault (as long as the fault routine is set up properly).

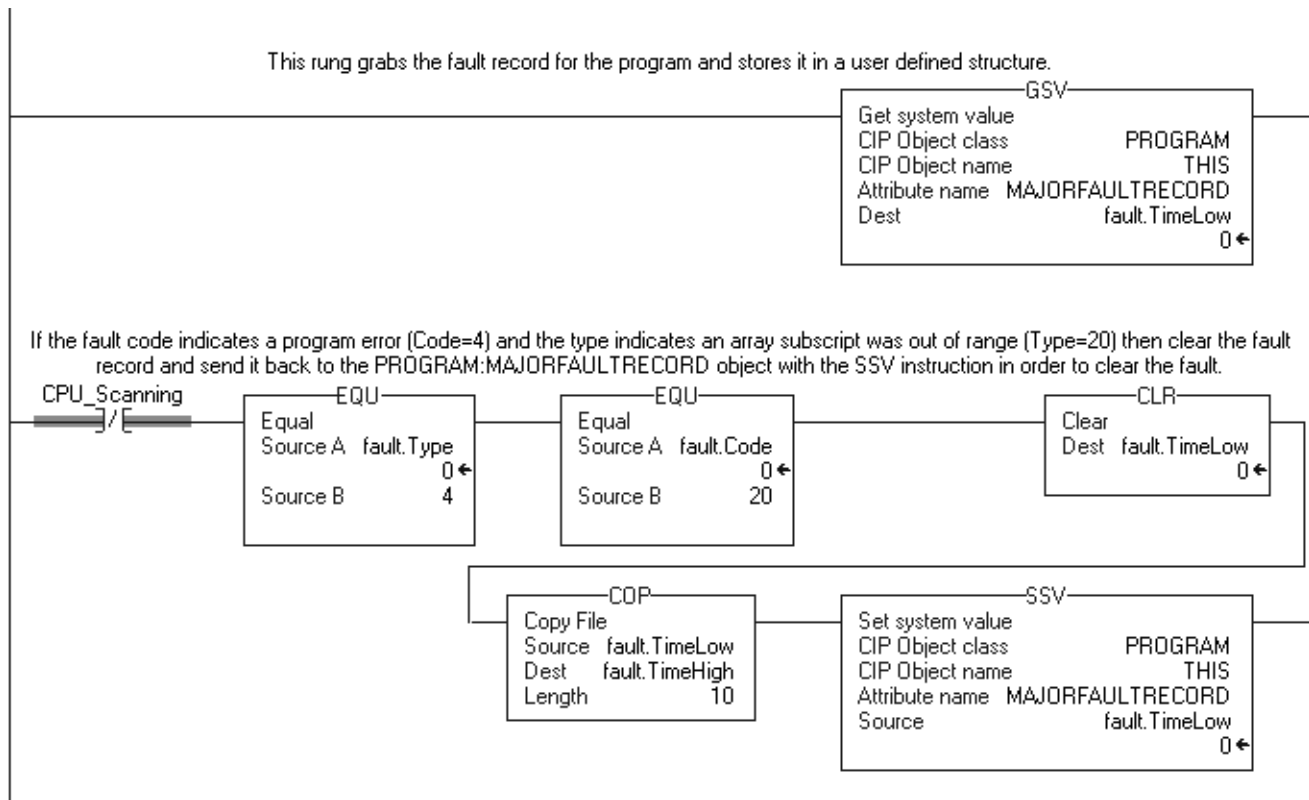


This rung will cause the controller to major fault during prescan if the pointer is not within the size of the array.
 Set up code in a program fault handler to prevent this condition from occurring.

table[pointer].0



Program Fault Routine:



IEC1131-3 Compliance

Using This Appendix

For information about:	See page:
Introduction	B-1
Operating system	B-2
Data definitions	B-2
Programming languages	B-3
Instruction set	B-3
IEC1131-3 program portability	B-4
IEC compliance tables	B-4

Introduction

The International Electrotechnical Commission (IEC) has developed a series of specifications for programmable controllers. These specifications are intended to promote international unification of equipment and programming languages for use in the controls industry. These standards provide the foundation for the Logix5550 controller and RSLogix 5000 programming software.

The IEC programmable controller specification is broken down into five separate parts each focusing on a different aspect of the control system:

- Part 1: General Information
- Part 2: Equipment and Requirements Test
- Part 3: Programming Languages
- Part 4: User Guidelines
- Part 5: Messaging Service Specification

The controls industry as a whole has focused on part 3 (IEC1131-3), Programming Languages, because it provides the cornerstone for implementing the other standards and provides the most significant end user benefit by reducing training cost. Because of this, only IEC1131-3 is addressed here.

The IEC1131-3 programming language specification addresses numerous aspects of programmable controller including the operating system execution, data definitions, programming languages, and instruction set. Components of the IEC113-3 specification are categorized as required by the specification, optional or extensions. By so doing, the IEC113-3 specification provides a minimum set of functionality that can be extended to meet end user application needs. The downside of this approach is that each programmable control system vendor may implement different components of the specification or provide different extensions.

Operating System

The controller's preemptive multitasking operating system (OS) is compliant with the IEC1131-3 definition. In IEC1131-3, the programmable controllers OS can contain zero or more tasks, that can execute one or more programs each containing one or more functions or routines. According to IEC1131-3, the number of each of these components is implementation dependent. The Logix5550 provides 32 task each containing 32 programs and an unlimited number of functions or routines.

IEC1131-3 provides an option for creating different task execution classifications. Task may be configured as continuous, periodic or event based. A continuous task does not need to be scheduled in that it will utilize any left over processing time when other tasks are dormant. Periodic tasks are scheduled to operate based on a reoccurring time period. The IEC1131-3 specification does not specify a time base for periodic task configuration. An IEC1131-3 event based task is triggered upon detection of the rising edge of a configured input. The Logix5550 provides support for both continuous and periodic task options. Additionally, the period for a periodic task, is configurable starting as low as 1 millisecond (ms).

Data Definitions

The IEC1131-3 specification provides access to memory through the creation of named variables. IEC1131-2 names for variables consist of a minimum of six characters (RSLogix5000 programming software supports a minimum of 1 character) starting with an underscore "_" or an alpha character (A-Z), followed by one or more characters consisting of an underscore "_", alpha character (A-Z) or a number (0-9). Optionally, lower case alpha characters (a-z) can be supported as long as they are case insensitive (A = a, B = b, C = c ...). The controller provides full compliance with this definition, supports the lower case option and extends the name to support up to 40 character names.

Data variables in IEC1131-3 may be defined such that they are accessible to all programs within a resource or controller, or limited access is provided only to the functions or routines within a single program. To pass data between multiple resources or controllers, access paths may be configured to define the location of the data within a system. The Logix5550 provides compliance by providing program scoped, controller scoped data and permits the configuration of access paths using produced/consumed data.

The memory interpretation of a variable within IEC1131-3 is defined through the use of either an elementary data type or an optional derived data type that is created from a group of multiple data types. The Logix5550 supports the use of the BOOL (1 bit), SINT (8 bit integer), INT (16 bit integer), DINT (32 bit integer) and REAL (IEEE floating point number) elementary data types. Additionally, the optional derived data types are supported through the creation of user defined structures and arrays.

Programming Languages

The IEC113-3 specification defines five (5) different programming languages and a set of common elements. All languages are defined as optional but at least one must be supported in order to claim compliance with the specification. The IEC1131-3 programming language components are defined as follows:

- Common Language Elements
- Common Graphical Elements
- Instruction List (IL) Language Elements
- Structured Text Language (ST) Elements
- Ladder Diagram (LD) Language Elements
- Sequential Function Chart (SFC) Language Elements
- Function Block Diagram (FBD) Language Elements

The controller and RSLogix5000 provide support for the common language elements and the Ladder Diagram language options. Additionally, the environment utilizes an ASCII import/export format based on the Structured Text language. The instruction set and program file exchange features are discussed in detail in the sections that follow.

Instruction Set

The instruction set specified by IEC1131-3 is entirely optional. The specification lists a limited set of instructions that if implemented must conform to the stated execution and visual representation. IEC1131-3 however, does not limit the instructions set to those listed within the specification. Each PLC vendor is free to implement additional functionality in the form of instructions over and above those listed by the specification. Examples of such extended instructions are those needed to perform diagnostics, PID loop control, motion control and data file manipulation. Because extended instructions are not defined by the IEC1131-3 specification, there is no guarantee that the implementation between different PLC vendors will be compatible. Thus utilization of these instructions may preclude the movement of logic between vendors.

The controller and RSLogix5000 provide a suite of instructions that execute as defined by the IEC1131-3 specification. The physical representation of these instructions maintain their look and feel with existing systems so as to reduce the training cost associated with working with the environment. In addition to the IEC1131-3 compliant instructions, a full range of instructions from existing products have been brought forward into the environment so that no functionality is lost.

IEC1131-3 Program Portability

One of the goals of end-users creating programs in an IEC1131-3 compliant environment is the movement or portability of programs between controllers developed by different vendors. This area is a weakness of IEC113-3 because no file exchange format is defined by the specification. This means that if any program created in one vendor's environment will require manipulation to move it to another vendor's system.

In order to minimize the effort involved in performing cross-vendor portability, the RSLogix 5000 programming software for the controller includes a full ASCII export and import utility. Additionally, the file format that is utilized by this tool is based on a hybrid of the IEC1131-3 Structured Text language definition. Controller operating system and data definitions follow the appropriate IEC1131-3 formats. Extensions were implemented in order to convert Ladder Diagram logic into ASCII text since this is not defined by IEC1131-3.

IEC Compliance Tables

The controller and RSLogix5000 complies with the requirements of IEC1131-3 for the following language features:

Table Number: ¹	Feature Number:	Feature Description:	Implementation Notes:
1	1	Required character set	none
1	2	Lower case letters	none
1	3a	Number sign (#)	Used for immediate value data type designation
1	4a	Dollar sign (\$)	Used for description and string control character
1	6a	Subscript delimiters ([])	Array subscripts
2	1	Identifiers using upper case and numbers	Task, program, routine, structure and tag names
2	2	Identifiers using upper case, numbers, and embedded underlines	Task, program, routine, structure and tag names
2	3	Identifiers using upper and lower case, numbers and embedded underlines	Task, program, routine, structure and tag names
4	1	Integer literal	12, 0, -12
4	2	Real literal	12.5, -12.5
4	3	Real literal with exponents	-1.34E ⁻¹² , 1.234E ⁶
4	4	Base 2 literal	2#0101_0101
4	5	Base 8 literal	8#377
4	6	Base 16 literal	16#FFEO
4	7	Boolean zero and one	0, 1
5	1	Empty String ''	Descriptions
5	2	String of length one containing a character 'A'	Descriptions
5	3	String of length one containing a space ' '	Descriptions
5	4	String of length one containing a single quote character '\$'	Descriptions
5	5	String of length two containing CR and LF '\$R\$L'	Descriptions
6	2	String dollar sign '\$\$'	Descriptions
6	3	String single quote '\$'	Descriptions
6	4	String Line Feed '\$L' or '\$I'	Descriptions
6	5	String New-line '\$N' or '\$n'	Descriptions
6	6	String From Feed (page) '\$P' or '\$p'	Descriptions
6	7	String Carriage return '\$R' or '\$r'	Descriptions





Table Number: ¹	Feature Number:	Feature Description:	Implementation Notes:
6	8	String Tab '\$T' or '\$t'	Descriptions
10	1	BOOL Data Type	Tag variable definition
10	2	SINT Data Type	Tag variable definition
10	3	INT Data Type	Tag variable definition
10	4	DINT Data Type	Tag variable definition
10	10	REAL Data Type	Tag variable definition
10	12	Time	Tag variable definition, TIMER Structure
11	1	Data type Hierarchy	none
12	1	Direct Derivation from elementary types	User Defined data type structures
12	4	Array data types	Tag variable definition
12	5	Structured Data types	User defined data type structures
13	1	BOOL, SINT, INT, DINT initial value of 0	Tag variable definition
13	4	REAL, LREAL initial value of 0.0	Tag variable definition
13	5	Time initial value of T#0s	Tag variable definition, reset (RES) instruction
13	9	Empty String ''	Descriptions
14	1	Initialization of directly derived types	Import/export
14	4	Initialization of array data types	Import/export
14	5	Initialization of structured type elements	Import/export
14	6	Initialization of derived structured data types	Import/export
20	1	Use of EN and ENO for LD	Function present in ladder but not labeled
21	1	Overloaded functions ADD(INT, DINT) or ADD(DINT, REAL)	All instructions overloaded types that are supported documented with each instruction
22	1	_TO_ conversion function	RAD, DEG instructions Radians to/from Decimal. Others not needed because of instruction overloading
22	3	BCD to INT Convert	FRD instruction
22	4	INT to BCD Convert	TOD instruction
23	2	Square root	SQR instruction
23	3	Natural log	LN instruction
23	4	Log base 10	LOG instruction
23	6	Sine in radians	SIN instruction
23	7	Cosine in radians	COS instruction
23	8	Tangent in radians	TAN instruction
23	9	Principal arc sine	ASN instruction
23	10	Principal arc cosine	ACS instruction
23	11	Principal arc tangent	ATN instruction
24	12	Arithmetic add	ADD instruction
24	13	Arithmetic multiplication	MUL instruction
24	14	Arithmetic subtraction	SUB instruction
24	15	Arithmetic divide	DIV instruction
24	17	Exponentiation	XPY instruction
24	18	Value move	MOV instruction
26	5	Bitwise AND	AND instruction
26	6	Bitwise OR	OR instruction
26	7	Bitwise XOR	XOR instruction
26	8	Bitwise NOT	NOT instruction
28	5	Comparison greater-than	GRT instruction
28	6	Comparison greater-than or equal	GRE instruction
28	7	Comparison equal	EQU instruction
28	8	Comparison less-than	LES instruction

Table Number: ¹	Feature Number:	Feature Description:	Implementation Notes:
28	9	Comparison less-than or equal	LEQ instruction
28	10	Comparison not equal	NEQ instruction
57	1, 2	Horizontal line for rung	Ladder editor
57	3, 4	Vertical line	Ladder editor
57	5, 6	Horizontal / Vertical connection	Ladder editor
57	9, 10	Connection and non-connection corners	Ladder editor
57	11, 12	Blocks with connections	Ladder editor
58	2	Unconditional jump	JMP instruction
58	3	Jump target	LBL instruction
58	4	Conditional jump	JMP instruction
58	5	Conditional return	RET instruction
58	8	Unconditional return	RET instruction
59	1	Left hand power rail	Ladder editor
59	2	Right hand power rail	Ladder editor
60	1	Horizontal link	Ladder editor
60	2	Vertical link	Ladder editor
61	1, 2	Normally open contact -- --	XIC instruction
61	3, 4	Normally close contact -- / --	XIO instruction
61	5, 6	Positive transition sensing contact - P -	ONS instruction
62	1	Coil --()--	OTE instruction
62	6	Set retentive memory coil -(SM)-	OTL instruction
62	7	Reset retentive memory coil -(RM)-	OTU instruction
62	8	Positive transition sensing coil	OSR instruction
62	9	Negative transition sensing coil	OSF instruction

1. Table associated with languages other than ladder diagram have been skipped.

Specifications

Logix5550 Controller

Description:	Value:		
backplane current		+5V dc	+24V dc
	1756-L1	0.65A	0.02A
	1756-L1M1	0.95A	0.02A
	1756-L1M2	1.05A	0.02A
	1756-L1M3	1.20A	0.02A
temperature	operating	0° to 60° C (32 to 140° F)	
	storage	-40° to 85° C (-40 to 185° F)	
relative humidity	5% to 95% noncondensing		
vibration	10 to 500 Hz 2.0 G maximum peak acceleration		
shock	operating	30G peak for 11ms	
	storage	50G peak for 11ms	
weight	1756-L1	10.0 oz.	
	1756-L1M1	12.5 oz.	
	1756-L1M2	12.5 oz.	
	1756-L1M3	12.7 oz.	
battery	1756-BA1 (PROMARK Electronics 94194801) 0.59g lithium		
programming cable	1756-CP3 serial cable category 3 ^{1,2}		
agency certification ³ (when product or packaging is marked)	<div></div> <div> Class I Division 2 Hazardous</div> <div> marked for all applicable directives</div>		





1. Use this conductor category information for planning conductor routing as described in the system level documentation.
2. Refer to *Programmable Controller Wiring and Grounding Guidelines*, publication 1770-4.1
3. CSA certification - Class I Division 2, Group A, B, C, D or nonhazardous locations

Logix5550 Memory Board

You can install one of these memory boards in the controller:

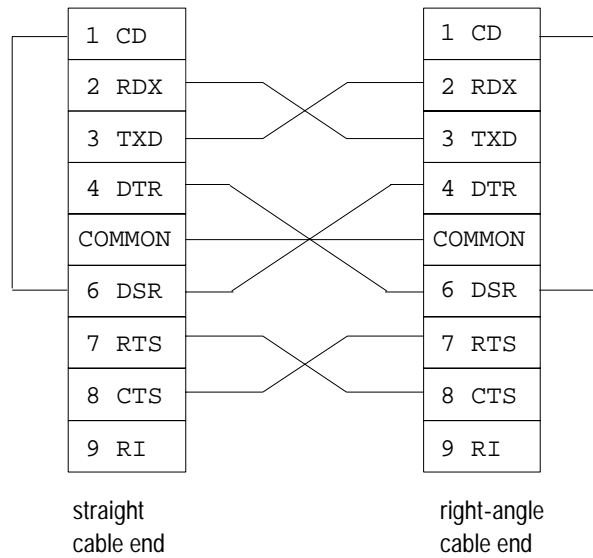
- 1756-M1 (512 Kbytes expansion memory)
- 1756-M2 (1 Mbytes expansion memory)
- 1756-M3 (2 Mbytes expansion memory)

The 1756-M_x memory boards are designed to work only with the 1756-L1 Logix5550 controller.

Description:	Value:	
backplane current	+5V dc	
Add this current demand to that of the Logix5550 controller (1756-L1).	1756-M1	0.30A
	1756-M2	0.40A
	1756-M3	0.55A
temperature		
operating	0° to 60° C	(32 to 140° F)
storage	-40° to 85° C	(-40 to 185° F)
relative humidity	5% to 95% noncondensing	
vibration	10 to 500 Hz 2.0 G maximum peak acceleration	
shock		
operating	30G peak for 11ms	
storage	50G peak for 11ms	
weight	1756-M1	2.5 oz.
	1756-M2	2.5 oz.
	1756-M3	2.7 oz.
agency certification (when product or packaging is marked) ¹	   Class I Division 2 Hazardous  marked for all applicable directives	

1. CSA certification - Class I Division 2, Group A, B, C, D or nonhazardous locations

1756-CP3 Serial Cable Pinouts



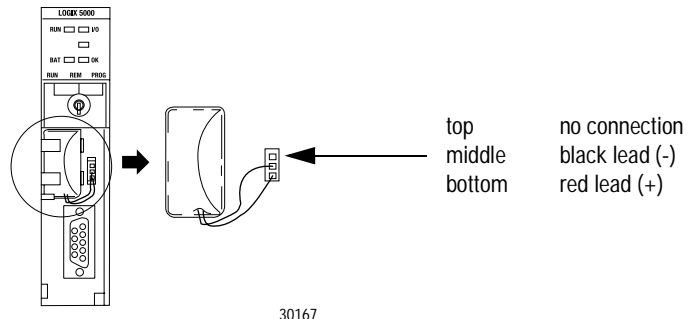
40046

The cable is shielded and tied to the connector housing at both ends.

1756-BA1 Battery

The Logix5550 controller uses the 1756-BA1 battery:

Battery	1756-BA1 0.59g lithium
---------	---------------------------






30167

Store the lithium battery in a cool, dry environment, typically 20° C to 25° C (68° F to 77° F) and 40% to 60% relative humidity. When not installed in the controller, store the battery in the original container, away from flammable materials.

1756-M0A2E Motion Module

Description:	Value:
number of axes per Logix5550 controller	32 axes maximum
maximum number of axes per coarse update rate	coarse update rate: max. number of axes:
The coarse update rates assume that the servo is on for each axis and that each axis has an active trapezoidal move. For more information, see the <i>ControlLogix Motion Module User Manual</i> , publication 1756-6.5.16.	2 ms 2
	3 ms 3
	4 ms 4
	5 ms 6
	6 ms 7
	7 ms 8
	8 ms 10
	9 ms 11
	10 ms 13
	11 ms 14
	12 ms 15
	13 ms 17
	14 ms 18
	15 ms 20
	16 ms 21
	17 ms 22
	18 ms 24
	19 ms 25
	20 ms 26
	21 ms 28
	22 ms 29
	23 ms 30
	24 ms 32
number of axes per module	2 axes maximum
module keying	electronic
servo loop	
type	nested PI digital position and velocity servo
gain resolution	32-bit floating point
absolute position range	±1,000,000,000 encoder counts
rate	5 kHz
power dissipation	5.5W maximum
backplane current	5V dc @ 700 mA 24V dc @ 2.5 mA
encoder input	
type	incremental AB quadrature with marker
mode	4X quadrature
rate	4 MHz counts per second maximum
electrical interface	optically isolated 5V differential
voltage range	3.4V to 5.0V differential
input impedance	531 Ohms differential

Description:	Value:
registration inputs	
type	optically isolated, current sourcing input
24V input voltage	+24V dc nominal
maximum	26.4V
minimum on	18.5V
maximum off	3.5
5V input voltage	+5V dc nominal
maximum	5.5V
minimum on	3.7V
maximum off	1.5V
input impedance	
24V input	1.2 kOhms
5V input	9.5 kOhms
response time	1 μ s
all other inputs	
type	optically isolated, current sinking input
input voltage	+24V dc nominal
maximum	26.4V
minimum on	17.0V
maximum off	8.5V
input impedance	7.5 kOhms
servo output	
type	analog voltage
isolation	200 kOhms
voltage range	± 10 V
voltage resolution	16 bits
load	5.6 kOhms resistive minimum
maximum offset	25 mV
gain error	$\pm 4\%$
all other outputs	
type	solid-state isolated relay contacts
operating voltage	+24V dc nominal (Class 2 source)
maximum	26.4V
operating current	75 mA
RTB screw torque (cage clamp)	5lb-in. (0.5 Nm) maximum

Description:	Value:
conductors	
wire size	22 gauge (3.1mm ²) minimum to copper ¹ 3/64 inch (1.2 mm) insulation maximum
category	1 ^{2,3}
temperature	
operating	0° to 60° C (32° to 140° F)
storage	-40° to 85° C (-40° to 185° F)
relative humidity	5% to 95% noncondensing
agency certification (when product or packaging is marked) ⁴	  Class I Division 2 Hazardous  marked for all applicable directives

1 Maximum wire size will require the extended depth RTB housing (1756-TBE).

2 Use this conductor category information for planning conductor routing as described in the system level installation manual.

3 Refer to *Programmable Controller Wiring and Grounding Guidelines*, publication number 1770-4.1.

4 CSA certification - Class I Division 2, Group A, B, C, D or nonhazardous locations.

This glossary is specific to ControlLogix terms. For a comprehensive glossary, see the *Industrial Automation Glossary*, publication AG-7.1.

A

- alias tag** A tag that references another tag. An alias tag can refer to another alias tag or a base tag. An alias tag can also refer to a component of another tag by referencing a member of a structure, an array element, or a bit within a tag or member. See *base tag*.
- atomic data type** The basic definition used to allocate bits, bytes or words of memory and define their numeric interpretation, this includes BOOL, SINT, INT, DINT, and REAL data types. See *array, structure*.
- array** A numerically indexed sequence of elements, each of the same data type. In the Logix5550 controller, an index starts at 0 and extends to the number of elements minus 1 (zero based). An array can have as many as three dimensions, unless it is a member of a structure where it can have only one dimension. An array tag occupies a contiguous block of memory in the controller, each element in sequence. See *atomic data type, structure*.
- application** The combination of routines, programs, tasks, and I/O configuration used to define the operation of a single controller. See *project*.

B

- base tag** A tag that actually defines the memory where a data element is stored. See *alias tag*.
- bidirectional connection** A connection in which data flows in both directions: from the originator to the receiver and from the receiver to the originator. See *connection, unidirectional connection*.
- binary** Integer values displayed and entered in base 2 (each digit represents a single bit). Prefixed with 2#. Padded out to the length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of four digits is separated by an underscore for legibility. See *decimal, hexadecimal, octal*.
- bit** Binary digit. The smallest unit of memory. Represented by the digits 0 (cleared) and 1 (set).

BOOL An atomic data type that stores the state of a single bit (0 or 1).

byte A unit of memory consisting of 8 bits.

C ---

cached connection With the MSG instruction, a cached connection instructs the controller to maintain the connection even after the MSG instruction completes. This is useful if you repeatedly execute the MSG instruction because initiating the connection each time increases scan time. See *connection*, *uncached connection*.

change of state (COS) Any change in the status of a point or group of points on an I/O module.

CIP See Control and Information Protocol.

communication format Defines how an I/O module communicates with the controller. Choosing a communication format defines:

- what configuration tabs are available through the programming software
- the tag structure and configuration method

compatible module An electronic keying protection mode that requires that the vendor, catalog number, and major revision attributes of the physical module and the module configured in the software match in order to establish a connection to the module. See *disable keying*, *exact match*.

connection The communication mechanism from the controller to another module in the control system. The number of connections that a single controller can have is limited. Communications with I/O modules, consumed tags, produced tags, and MSG instructions use connections to transfer data.

consumed tag A tag that receives its data from another controller. Consumed tags are always at controller scope. See *produced tag*.

continuous task A task that runs continuously, restarting the execution of its programs when the last one finishes. There can be only one continuous task, although there does not have to be any. See *periodic task*.

Control and Information Protocol Messaging protocol used by Allen-Bradley's series ControlLogix line of control equipment. Native communications protocol used on the ControlNet network.

ControlBus The backplane used by the 1756 chassis. Acts as a network.

controller scope Data accessible anywhere in the controller. The controller contains a collection of tags that can be referenced by the routines and alias tags in any program, as well as other aliases in the controller scope. See *program scope*.

Coordinated System Time (CST) A synchronized time value for all the modules within a single ControlBus chassis. Data timestamped with CST data from modules within a single ControlBus chassis can safely be compared to determine the relative time between data samples.

COUNTER Structure data type that contains status and control information for counter instructions

D

data type A definition of the memory size and the layout of memory that will be allocated when a tag of the data type is created. Data types can be atomic, structures, or arrays.

decimal Integer values displayed and entered in base 10. No prefix. Not padded to the length of the integer. See *binary*, *hexadecimal*, *octal*.

description Descriptions for tags are as many as 120 characters long; descriptions for other objects are as many as 128 characters long. Any printable character can be used, including carriage return, tab, and space.

dimension Specification of the size of an array. Arrays can have as many as three dimensions.

DINT An atomic data type that stores a 32-bit signed integer value (-2,147,483,648 to +2,147,483,647).

direct An I/O connection where the controller establishes an individual connections with an I/O module. See *rack optimized*.

disable keying An electronic keying protection mode that requires no attributes of the physical module and the module configured in the software to match and still establishes a connection to the module. See *compatible module*, *exact match*.

download The process of transferring the contents of a project on the workstation into the controller. See *upload*.

E

elapsed time	The total time required for the execution of all operations configured within a single task. If the controller is configured to run multiple tasks, elapsed time includes any time used/shared by other tasks performing other operations. See <i>execution time</i> .
electronic keying	A feature of the 1756 I/O line where modules can be requested to perform an electronic check to insure that the physical module is consistent with what was configured by the software. Enables the user via the software to prevent incorrect modules or incorrect revisions of modules from being inadvertently used. See <i>compatible module</i> , <i>disable keying</i> , <i>exact match</i> .
element	An addressable unit of data that is a sub-unit of a larger unit of data. A single unit of an array. See <i>array</i> .
exact match	An electronic keying protection mode that requires that all attributes (vendor, catalog number, major revision, and minor revision) of the physical module and the module configured in the software match in order to establish a connection to the module.
execution time	The total time required for the execution of a single program. Execution time includes only the time used by that single program, and excludes any time shared/used by programs in other tasks performing other operations. See <i>elapsed time</i> .
exponential	Real values displayed and entered in scientific or exponential format. The number is always displayed with one digit to the left of the decimal point, followed by the decimal portion, and then by an exponent. See <i>style</i> .

F

faulted mode	The controller generated a major fault, could not clear the fault, and has shut down.
float	Real values displayed and entered in floating point format. The number of digits to the left of the decimal point varies according to the magnitude of the number. See <i>style</i> .

H

hexadecimal Integer values displayed and entered in base 16 (each digit represents four bits). Prefixed with 16#. Padded out to length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of four digits is separated by an underscore for legibility. See *binary*, *decimal*, *octal*.

I

immediate value An actual 32-bit signed real or integer value. Not a tag that stores a value.

index A reference used to specify an element within an array.

INT An atomic data type that stores a 16-bit integer value (-32,768 to +32,767).

interface module (IFM) A prewired I/O field wiring arm.

L

listen-only connection An I/O connection where another controller owns/provides the configuration data for the I/O module. A controller using a listen-only connection does not write configuration data and can only maintain a connection to the I/O module when the owner controller is actively controlling the I/O module. See *owner controller*.

M

major fault A malfunction, either hardware or instruction, that sets a major fault bit and processes fault logic to try to clear the fault condition. If the fault logic cannot clear the fault, logic execution stops, the controller shuts down, and outputs go to their configured state. See *faulted state*, *minor fault*.

major revision The 1756 line of modules have major and minor revision indicators. The major revision is updated any time there is a functional change to the module. See *electronic keying*, *minor revision*.

master (CST)	Within a single chassis, one and only one, controller must be designated as the Coordinated System Time (CST) master. All other modules in the chassis synchronize their CST values to the CST master.
member	An element of a structure that has its own data type and name. Members can be structures as well, creating nested structure data types. Each member within a structure can be a different data type. See <i>structure</i> .
memory	Electronic storage media built into a controller, used to hold programs and data.
minor fault	A malfunction, either hardware or instruction, that sets a minor fault bit, but allows the logic scan to continue. See <i>major fault</i> .
minor revision	The 1756 line of modules have major and minor revision indicators. The minor revision is updated any time there is a change to a module that does not affect its function or interface. See <i>electronic keying</i> , <i>major revision</i> .
multicast	A mechanism where a module can send data on a network that is simultaneously received by more than one listener. Describes the feature of the ControlLogix I/O line which supports multiple controllers receiving input data from the same I/O module at the same time.
multiple owners	A configuration setup where more than one controller has exactly the same configuration information to simultaneously own the same input module.

N

name	Names identify tags and modules. The naming conventions are IEC-1131-3 compliant. A name: <ul style="list-style-type: none"> • must begin with an alphabetic character (A-Z or a-z) or an underscore (_) • can contain only alphabetic characters, numeric characters, and underscores • can have as many as 40 characters • must not have consecutive or trailing underscore characters (_)
network update time (NUT)	The repetitive time interval in which data can be sent on a ControlNet network. The network update time ranges from 2ms-100ms.

O

object A structure of data that stores status information. When you enter a GSV/SSV instruction, you specify the object and its attribute that you want to access. In some cases, there are more than one instance of the same type of object, so you might also have to specify the object name. For example, there can be several tasks in your application. Each task has its own TASK object that you access by the task name.

octal Integer values displayed and entered in base 8 (each digit represents three bits). Prefixed with 8#. Padded out to the length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of three digits is separated by an underscore for legibility. See *binary*, *decimal*, *hexadecimal*.

owner controller The controller that creates the primary configuration and communication connection to a module. The owner controller writes configuration data and can establish a connection to the module. See *listen-only connection*.

P

path A description of the devices and networks between one device and another. A connection from one device to another follows the specified path. See *connection*.

periodic task A task that is triggered at a specific time interval. Whenever the time interval expires, the task is triggered and its programs are executed. There can be as many as 32 periodic tasks in the controller. See *continuous task*.

periodic task overlap A condition that occurs when an instance of one task is executing and the same task is triggered again. The execution time of the task is greater than the periodic rate configured for the task. See *periodic task*.

predefined structure A structure data type that stores related information for a specific instruction, such as the TIMER structure for timer instructions. Predefined structures are always available, regardless of the system hardware configuration. See *product defined structure*.

prescan A function of the controller where the logic is examined prior to execution in order initialize instructions and data.

The controller performs prescan when you change the controller from Program mode to Run mode.

priority	Specified precedence of task execution. If two tasks are triggered at the same time, the task with the higher priority executes first. Priorities range from 1-15, with 1 being the highest priority. If two tasks with the same priority are triggered at the same time, the controller switches between the tasks every millisecond. A continuous task runs at a fixed priority level that is lower than all the other tasks in the controller.
postscan	A function of the controller where the logic within a program is examined before disabling the program in order reset instructions and data.
produced tag	A tag that a controller is making available for use by other controllers. Produced tags are always at controller scope. See <i>consumed tag</i> .
product defined structure	A structure data type that is automatically defined by the software and controller. By configuring an I/O module you add the product defined structure for that module.
program	A program contains a set of related routines and a collection of tags. When a program is executed by a task, execution of logic starts at the configured main routine. That routine can, in turn, execute subroutines using the JSR instruction. If a program fault occurs, execution jumps to a configured fault routine for the program. Any of these routines can access the program tags, but routines in other programs cannot access these program tags. See <i>routine</i> , <i>task</i> .
program scope	Data accessible only within the current program. Each program contains a collection of tags that can only be referenced by the routines and alias tags in that program. See <i>controller scope</i> .
project	The file that the programming software uses to store a controller's logic and configuration. See <i>application</i> .

R

rack optimized	An I/O connection where the 1756-CNB module collects digital I/O words into a rack image (similar to 1771-ASB). A rack optimized connection conserves ControlNet connections and bandwidth, however, limited status and diagnostic information is available when using this connection type. See <i>direct</i> .
REAL	An atomic data type that stores a 32-bit IEEE floating-point value.
removal and insertion under power (RIUP)	A ControlLogix feature that allows a user to install or remove a module while chassis power is applied.

requested packet interval (RPI) When communicating over a the network, this is the maximum amount of time between subsequent production of input data. Typically, this interval is configured in microseconds. The actual production of data is constrained to the largest multiple of the network update time that is smaller than the selected RPI. The selected RPI must be greater than or equal to the network update time.

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

S

scan time See *elapsed time*, *execution time*.

scope Defines where you can access a particular set of tags. See *controller scope*, *program scope*.

SINT An atomic data type that stores an 8-bit signed integer value (-128 to +127).

structure A structure stores a group of data, each of which can be a different data type. The controller has its own predefined structures. Each I/O module you can configure for the controller has its own predefined structures. And you can create specialized user-defined structures, using any combination of individual tags and most other structures. See *member*, *user-defined structure*.

style The format that numeric values are displayed in. See *binary*, *decimal*, *hexadecimal*, *octal*, *float*, *exponential*.

system overhead timeslice The percentage of time the controller allocates to perform communication and background functions.

T

tag A named area of the controller's memory where data is stored. Tags are the basic mechanism for allocating memory, referencing data from logic, and monitoring data. See *alias tag*, *base tag*, *consumed tag*.

task A scheduling mechanism for executing a program. As many as 32 programs can be scheduled to execute when a task is triggered. A task can be configured to run as a continuous task or a periodic task. As many as 32 tasks can be created to schedule programs. See *continuous task, periodic task*.

timestamp A ControlLogix process that records a change in input data with a relative time reference of when that change occurred.

U ---

uncached connection With the MSG instruction, an uncached connection instructs the controller to close the connection upon completion of the MSG instruction. Clearing the connection leaves it available for other controller uses. See *connection, cached connection*.

unidirectional connection A connection in which data flows in only one direction: from the originator to the receiver. See *connection, bidirectional connection*.

upload The process of transferring the contents of the controller into a project file on the workstation. See *download*.

user defined structure A user-defined structure groups different types of data into a single named entity. A user-defined structure contains one or more data definitions called members. Creating a member in a user-defined structure is just like creating an individual tag. The data type for each member determines the amount of memory allocated for the member. The data type for each member can be a/an:

- atomic data type
- product-defined structure
- user-defined structure
- single dimension array of an atomic data type
- single dimension array of a product-defined structure
- single dimension array of a user-defined structure

W ---

watchdog Specifies how long a task can run before triggering a major controller fault.

Numerics

- 1756-BA1 C-3
- 1756-CP3 cable C-3
- 1756-M02AE servo module 10-1

A

- accessing
 - data 4-2
 - FAULTLOG 12-20
 - I/O 3-16
- ACD file extension 2-1
- alias
 - defining for I/O data 3-19
 - getting started 1-14, 1-15
 - tag type 4-2
- array
 - introduction 4-13
 - memory allocation 4-17
- array concepts
 - indexing 4-14
 - specifying bits 4-15
- atomic data type 4-3, 4-6
- axis 10-4

B

- background function 5-15
- base tag 4-2
- battery 1-3, C-3
- bits within arrays 4-15
- branch 5-14

C

- change-of-state data exchange 3-2, 3-3
- changing
 - controller mode A-8
 - I/O configuration 3-15
 - module properties 1-11
 - project properties 2-2
- clearing a major fault 13-6
- communicating
 - defining connection path 9-2
 - mapping addresses 6-4
 - other controllers 6-1
 - serial 8-1
 - using MSG instructions 6-1
 - with another Logix5550 controller 6-1
 - with PLC and SLC controllers 6-2
 - with workstation 9-1
- communication connection 7-1
- compliance tables B-4
- components A-1
- configuring
 - communication with workstation 9-1
 - controller fault handler program 12-18
 - controller fault handler routine 12-19
 - DF1 master 8-8
 - DF1 point-to-point 8-6
 - DF1 protocol 8-5
 - DF1 slave 8-7

- I/O modules
 - alias 3-19
 - changing configuration 3-15
 - controller ownership 3-7
 - COS 3-3
 - electronic keying 3-6
 - inhibit operation 3-9
 - local 3-4
 - logic scan 3-2
 - naming 3-5
 - operation 3-3
 - remote 3-11
 - RPI 3-3
 - update 3-2
 - power-up handler 13-4
 - power-up handler program 13-4
 - power-up handler routine 13-5
 - program 5-10
 - routine 5-12
 - serial port 8-3
 - task 5-6
 - watchdog 5-8
- connecting to controller serial port 8-2
- connection
- allocating 7-1
 - consumed tag 7-7
 - direct connection 7-2
 - for I/O module 7-2
 - messaging 7-7
 - produced tag 7-6
 - rack-optimized 7-4
 - requirements 7-8
- connection path 9-2
- consumer
- connection 7-7
 - maximum number of produced and consumed tags 6-8
 - processing 6-7
 - system-shared tag 6-6
- continuous task 5-2, 5-3
- controller
- fault handler 12-16
 - faults 12-1
- controller fault handler 12-9
- controller memory 4-1
- controller mode A-8
- controller organizer 2-3, 3-19
- controller ownership 3-7
- controller scope 4-20
- converting data types 4-8
- coordinated system time 2-5, 10-2
- COS 3-3
- creating
- controller fault handler 12-16
 - controller fault handler program 12-17
 - power-up handler program 13-3
 - program 5-9
 - program fault routine 12-16
 - project 1-4, 2-1
 - routine 5-11
 - sample I/O module 1-7, 1-9
 - sample project 1-5
 - tags 1-13, 4-2
 - task 5-5
- CST. See coordinated system time
- D**
- data
- accessing 4-2
 - array 4-13
 - atomic type 4-3, 4-6
 - definitions B-2
 - forcing 11-2
 - how stored 4-1
 - organizing 4-1
 - predefined structures 4-4
 - specifying bits 4-8

- structure
 - I/O 3-16
 - introduction 4-9
 - member 4-12
 - module-defined 4-10
 - predefined 4-10
 - user-defined 4-10
- type conversion 4-8
- types 4-3
- data exchange
 - change of state 3-2
- devices not responding A-3
- DF1 protocol
 - configuring 8-5
 - introduction 8-4
 - master 8-4, 8-8
 - master/slave methods 8-5
 - point-to-point 8-4, 8-6
 - slave 8-4, 8-7
- direct connection 7-2
- documenting I/O 1-14, 1-15
- downloading
 - project 1-4, 5-16
 - sample project 1-18

E

- electronic keying 3-6
- entering
 - branch 5-14
 - logic 5-13
 - sample logic 1-16
- example
 - connection path 9-4
 - coordinated system time 2-6
 - getting started 1-16
 - major fault 12-12
 - minor fault 12-5
 - motion 10-2
 - power-up 13-6
 - viewing I/O module faults 3-22

F

- fault
 - controller 12-1
 - controller fault handler 12-9, 12-16
 - I/O module 3-19
 - logic for major 12-12
 - logic for minor 12-5, 12-7
 - major types and codes 12-14
 - minor types and codes 12-8
 - monitoring I/O 12-2
 - processing major 12-9
 - processing minor 12-3
 - program fault routine 12-9, 12-16
 - types 12-1
- forcing
 - description 11-1
 - disabling 11-5
 - enabling 11-4
 - entering 11-2
 - I/O tags 11-1
 - monitoring 11-6
 - removing 11-5
- front plate A-1

G

- getting started
 - adding an input module 1-7
 - adding an output module 1-9
 - changing module properties 1-11
 - changing project properties 1-6
 - creating a project 1-5
 - creating other tags 1-13
 - documenting I/O with alias tags 1-14, 1-15
 - downloading a project 1-18
 - enter logic 1-16
 - installing 1-2
 - introduction 1-1
 - project 1-4
 - viewing controller memory usage 1-22
 - viewing I/O tags 1-12
 - viewing program scan time 1-21

H

hardware fault 12-1

I

I/O module

- alias 3-19
- change-of-state data exchange 3-2, 3-3
- changing configuration 3-15
- configuring local 3-4
- configuring remote 3-11
- connection 7-2
- controller ownership 3-7
- creating sample 1-7, 1-9
- data structure 4-10
- determining if not responding A-3
- direct connection 7-2
- electronic keying 3-6
- inhibit operation 3-9
- monitoring fault 12-2
- naming 3-5
- operation 3-3
- properties 1-11
- rack-optimized connection 7-4
- RPI 3-3
- updates 3-2
- viewing fault information 3-19

I/O tags

- forcing 11-1

IEC 1131-3 compliance

- data definitions B-2
- instruction set B-3
- introduction B-1
- operating system B-2
- program portability B-4
- programming language B-3
- tables B-4

indexing 4-14

individual tag 4-6

inhibit I/O operation 3-9

installing

- controller module 1-3
- ESD precautions 1-2
- preparing the controller 1-3

instruction set B-3

integrating motion 10-1

introduction 1-1

K

keying, electronic 3-6

keyswitch A-8

L

LED states A-2

listen-only 3-7

local

- example I/O addressing 3-17
- I/O module 3-4

logic

- branch 5-14
- entering 1-16, 5-13
- major fault 12-12
- minor fault 12-5, 12-7
- scan 3-2

M

major fault

- clearing 13-6
- controller
 - fault handler 12-9
- controller fault handler 12-16
- description 12-1
- logic 12-12
- power-up handler 13-1
- processing 12-9
- program fault routine 12-9, 12-16
- types and codes 12-14

mapping an address 6-4

master/slave communication 8-5

member 4-12

- memory
 - controller 4-1
 - usage 1-22
- memory allocation
 - array 4-17
 - base tag 4-6
 - structure 4-11
- memory expansion board 1-3, 4-1, C-2
- messaging connection 7-7
- minor fault
 - description 12-1
 - logic 12-5, 12-7
 - processing 12-3
 - types and codes 12-8
- mode A-8
- module-defined structure 4-10
- motion
 - adding a module 10-3
 - autotuning 10-11
 - configuring an axis 10-5
 - hookup diagnostics 10-11
 - integrating 10-1
 - naming an axis 10-4
 - selecting CST master 10-2
- motion example 10-2
- MSG instruction 6-1
- multiple controllers 1-3, 3-7

N

- naming
 - controller 2-2
 - I/O module 3-5
 - program 5-9, 12-17, 13-3
 - routine 5-12, 12-19, 13-6
 - tag 4-4
 - task 5-6

O

- operating system B-2
- organizing data 4-1
- organizing project 5-1
- owner controller 3-7
- ownership 3-7

P

- periodic task 5-2, 5-3
- pinouts C-3
- power up in Run mode 13-1
- power-up handler 13-1
- predefined structure 4-4, 4-10
- prescan operations A-9
- producer
 - connection 7-6
 - maximum number of produced and consumed tags 6-8
 - processing 6-7
 - system-shared tag 6-6
- program 13-4
 - configuring 5-10
 - configuring controller fault handler 12-18
 - controller fault handler 12-17
 - creating 5-9
 - defining 5-8
 - developing 5-1
 - naming 5-9, 12-17, 13-3
 - portability B-4
 - power-up handler 13-1, 13-3
 - scan time 1-21
 - scope 4-20
 - unscheduled 12-17, 13-4

- programming example
 - coordinated system time 2-6
 - getting started 1-16
 - major fault 12-12
 - minor fault 12-5
 - monitoring forces 11-6
 - motion 10-2
 - power-up 13-6
 - viewing I/O module faults 3-22
- programming language B-3
- project
 - changing properties 2-2
 - controller organizer 2-3
 - creating 2-1
 - developing 5-1
 - downloading 1-18, 5-16
 - file extension 2-1
 - getting started 1-5
 - organizing 5-1
 - properties 1-6
 - saving 2-4
 - uploading 2-4
- R**
- rack-optimized connection 7-4
- referencing members 4-12
- remote
 - example I/O addressing 3-18
 - I/O module 3-11
- removal and insertion under power 1-2
- requested packet interval 3-3
- RIUP 1-2
- routine
 - configuring 5-12
 - configuring controller fault handler 12-19
 - configuring power-up handler 13-5
 - creating 5-11
 - defining 5-11
 - naming 5-12, 12-19, 13-6
 - program fault 12-9, 12-16
- RPI 3-3

S

- saving
 - project 2-4
 - save vs. save as 2-4
- scan time
 - program 1-21
- scope 4-20
- serial
 - 1756-CP3 cable C-3
 - cable pinouts C-3
 - communicating 8-1
 - configuring DF1 protocol 8-5
 - configuring port 8-3
 - connecting to controller serial port 8-2
 - DF1 protocol 8-4
 - master 8-8
 - point-to-point 8-6
 - RS-232 8-1
 - slave 8-7
- slave/master communication 8-5
- specifications
 - 1756-BA1 C-3
 - 1756-CP3 cable C-3
 - battery C-3
 - controller C-1
 - memory expansion board C-2
 - serial cable pinouts C-3
- specifying bits 4-8
- storing data 4-1
- structure
 - introduction 4-9
 - memory allocation 4-11
 - module-defined 4-10
 - predefined 4-10
 - referencing members 4-12
 - user-defined 4-10
- system overhead 5-15

- system-shared tag
 - connection for consumed tag 7-7
 - connection for produced tag 7-6
 - introduction 6-6
 - maximum number of produced and consumed tags 6-8
 - processing 6-7

T

- tag
 - connection for consumed 7-7
 - connection for produced 7-6
 - consumed 6-6
 - individual 4-6
 - naming 4-4
 - predefined I/O structure 3-16
 - produced 6-6
 - sample alias 1-14, 1-15
 - scope 4-20
 - system shared 6-6
 - types 4-2
 - viewing I/O 1-12
- tags
 - creating 4-2
- task
 - configuring 5-6
 - continuous 5-3
 - controller fault handler 12-9, 12-16
 - creating 5-5
 - defining 5-2
 - naming 5-6
 - periodic 5-3
 - system overhead 5-15
 - watchdog 5-8
- time 2-6
- troubleshooting
 - controller components A-1
 - LED states A-2
 - prescan operations A-9

U

- unscheduled program 12-17, 13-4
- update 3-2
- uploading 2-4
- user-defined structure 4-10
- using MSG instructions 6-1

V

- viewing
 - controller memory usage 1-22
 - I/O module fault information 3-19
 - I/O tags 1-12
 - program scan time 1-21

W

- WALLCLOCKTIME 2-6
- watchdog 5-8

Notes:

ControlLogix, Logix5550, PLC-5, PLC-3, PLC-2, SLC, DH+, Allen-Bradley, RSLinx, RSNetworx, and Rockwell Software are trademarks of Rockwell Automation.

ControlNet is a trademark of ControlNet International, Ltd.

DeviceNet is a trademark of the Open DeviceNet Vendor Association.

Ethernet is a trademark of Digital Equipment Corporation, Intel, and Xerox Corporation.

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

**Rockwell
Automation**