



Allen-Bradley

DriveLogix System

5720

User Manual

**Rockwell
Automation**

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 available from your local Rockwell Automation Sales Office or online at <http://www.ab.com/manuals/gi>) describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

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

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

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc. 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.

IMPORTANT

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

Rockwell Automation Support

Before you contact Rockwell Automation for technical assistance, we suggest you please review the troubleshooting information contained in this publication first.

If the problem persists, call your local distributor or contact Rockwell Automation in one of the following ways:

Phone	United States/Canada	1.262.512.8176 (7 AM - 6 PM CST) 1.440.646.5800 (24 hour support)
	Outside United States/Canada	You can access the phone number for your country via the Internet: 1. Go to http://www.ab.com 2. Click on <i>Support</i> (http://support.automation.rockwell.com) 3. Under <i>Contact Customer Support</i> , click on <i>Phone Support</i>
Internet	⇒	Go to http://ab.com/support/abdrives
Email	⇒	support@drives.ra.rockwell.com

Be prepared to furnish the following information when you contact support:

- Product Catalog Number
- Product Serial Number
- Firmware Revision Level

Your Questions or Comments on this Manual

If you find a problem with this manual, please notify us of it on the enclosed How Are We Doing form.

	Rockwell Automation Support	3
	Summary of Changes	
	Introduction	1
	Preface	
	Purpose of this Manual	1
	Who Should Use This Manual	2
	When to Use This Manual.	2
	How to Use this Manual	2
	Controller Firmware Revision.	2
Chapter 1	Getting Started	
	Introduction	1
	Connecting Battery	2
	Creating and Downloading a Project	3
	Creating a project	4
	Changing project properties	5
	Configuring the host PowerFlex 700S Drive	6
	Configuring the host PowerFlex 700S Drive (continued)	7
	Adding a local input module1	8
	Adding a local input module (continued)	9
	Adding a local output module	10
	Adding a local output module (continued)	11
	Adding a local analog module.	12
	Adding a local analog module (continued)	13
	Changing module properties	14
	Viewing I/O tags	15
	Creating other tags	16
	Documenting I/O with alias tags.	17
	Entering logic	18
	Entering logic (continued).	19
	Downloading a project	20
	Viewing program scan time.	21
	Viewing controller memory usage	22
	What To Do Next	22
Chapter 2	What Is DriveLogix?	
	Using This Chapter	1
	Developing programs	2
	Using the Event Task.	6
	How the DriveLogix System Uses Connections	9
	Determining Connections for Produced and Consumed Tags	10
	Determining Connections for Messages	11
	Determining Connections for I/O Modules	12
	Determining Total Connection Requirements.	17
	Downloading Projects	20
	Selecting a System Overhead Percentage	22

Chapter 3	Placing and Configuring the Drive	
	Using This Chapter	1
	Understanding the Interface to the Drive.	1
	Determining When the Controller Updates the Drive	3
	Placing and Configuring the Drive	4
	Inhibiting the Drive Connection	13
	Using DriveExecutive Lite.	15
	Accessing Drive Data	23
	Monitoring Drive Data.	23
Chapter 4	Placing and Configuring Local I/O	
	Using This Chapter	1
	Placing Local I/O Modules	2
	Determining When the Controller Updates I/O.	3
	Configuring a DIN Rail	5
	Configuring Local I/O Modules	6
	Inhibiting I/O Module Operation.	10
	Accessing I/O Data	13
	Monitoring I/O Modules	16
Chapter5	Configuring DriveLogix Motion	
	Using This Chapter	1
	Configuring the Drive	2
	Programming the Controller	5
	Supported Motion Commands	12
Chapter 6	Communicating with Devices on an EtherNet/IP Link	
	Using This Chapter	1
	Configuring Your System for a EtherNet/IP Link	1
	Configuring Remote I/O	8
	Sending Messages	13
	Producing and Consuming Data	20
	Guidelines for Configuring Connections.	23
	Example 1: DriveLogix Controller and Remote I/O	23
	Example 2: DriveLogix Controller to DriveLogix Controller	25
	Example 3: DriveLogix Controller to Other Devices	29
Chapter 7	Communicating with Devices on a ControlNet Link	
	Using This Chapter	1
	Configuring Your System for a ControlNet Link	1
	Configuring Remote I/O	5
	Scheduling the ControlNet Network	10
	Sending Messages	11
	Producing and Consuming Data	17
	Guidelines for Configuring Connections.	21
	Example 1: DriveLogix Controller and Remote I/O	22
	Example 2: DriveLogix Controller to DriveLogix Controller	24
	Example 3: DriveLogix Controller to Other Devices	28

Chapter 8	Communicating with Devices on a DeviceNet Link	
	Using This Chapter	1
	Configuring Your System for a DeviceNet Link	1
	Placing DeviceNet Devices	5
	Accessing DeviceNet Devices	6
	Placing the Communication Card in Run Mode	9
	Example 1: DriveLogix Controller and DeviceNet Devices	10
	Example 2: Using a 1788-CN2DN Linking Device	11
Chapter 9	Communicating with Devices on a Serial Link	
	Using This Chapter	1
	Configuring Your System for a Serial Link	1
	Example 1: Workstation Directly Connected to a DriveLogix Controller	8
	Example 2: Workstation Remotely Connected to a DriveLogix Controller	9
	Example 3: DriveLogix Controller to a Bar Code Reader	14
Chapter 10	Communicating with Devices on a DH-485 Link	
	Using This Chapter	1
	Configuring Your System for a DH-485 Link	1
	Planning a DH-485 Network	4
	Installing a DH-485 Network	6
	Example: DriveLogix Controller, ControlLogix Controller, and SLC Controller on the Same DH-485 Network	9
Chapter 11	Communicating with Devices on a Third-Party Link	
	Using This Chapter	1
	Configuring Your System for a Third-Party Link	1
Chapter 12	DriveLogix Back-Up on DeviceNet	
	Using This Chapter	1
	How the Back-up Works	2
	Power-Up and System Start-up	4
	Developing the DriveLogix Back-Up Application	6
	Using Indicators to Check Status	13
	Development and Debugging Tips	13
Appendix A	DriveLogix System Specifications	
	Using This Appendix	1
	DriveLogix Controller	1
	1756-BA1 Battery	3
	DriveLogix Controller Serial Cables	4
	DriveLogix Controller LEDs	6
Appendix B	Installing and Maintaining the Battery	
	Using this Appendix	1
	Connecting the Battery	1
	Storing Replacement Batteries	2
	Estimating Battery Life	2
	Replacing a Battery	3

Appendix C Access Procedures

- Using this Appendix 1
- Removing Cover(s) 2
- Removing Cover (For High Power Drives) 3
- Replacing Cover(s) 4
- Replacing Cover(s) Continued. 5
- Replacing Cover (For High Power Drives) 6

Index

Summary of Changes

Introduction

This version of the DriveLogix System User Manual corresponds to version 11 and later of the controller firmware. Changes made to this manual include:

For this updated information:	See:
Using the Event Task	2-6
How the DriveLogix System Uses Connections	2-9
Determining Connections for Produced and Consumed Tags	2-10
Determining Connections for Messages	2-11
Determining Connections for I/O Modules	2-12
Configuring DriveLogix Motion	5-1
Communicating with Devices on a Third-Party Link	11-1
DriveLogix Back-Up on DeviceNet	12-1
Access Procedures	C-1

Notes:

Purpose of this Manual

This manual guides the development of projects for DriveLogix controllers. It provides procedures on how to establish communications:

- over the following networks
 - ControlNet
 - DeviceNet
 - serial
- with the following devices
 - PowerFlex[®] 700S drive
 - controllers
 - I/O
 - workstations
 - PanelView terminals

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

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

This manual works together with the *Logix Controller Motion Instruction Set*, publication 1756-RM007D, which covers the following aspects of Logix Motion commands:

- Instruction Names
- Operands
- Structured Text
- Motion Instruction Structure
- Fault Conditions
- Execution
- Error Codes
- Status Bits

Who Should Use This Manual

This manual is intended for those individuals who program applications that use DriveLogix controllers, such as:

- software engineers
- control engineers
- application engineers
- instrumentation technicians

When to Use This Manual

Use this manual:

- when you are ready to integrate your application with the PowerFlex 700S drive, I/O devices, controllers, and networks in your system.
- *after* you perform these actions:
 - develop the basic code for your application
 - perform isolated tests of your application

How to Use this Manual

This manual is divided into the basic tasks that you perform while programming a DriveLogix controller. Each chapter covers a main task, such as communicating over a specific network. For each main task, the chapter:

- lists what you need
- describes the steps to follow to accomplish that task
- provides details for each step, as necessary
- includes example system configurations

Controller Firmware Revision

This revision of the DriveLogix User Manual corresponds to the following:

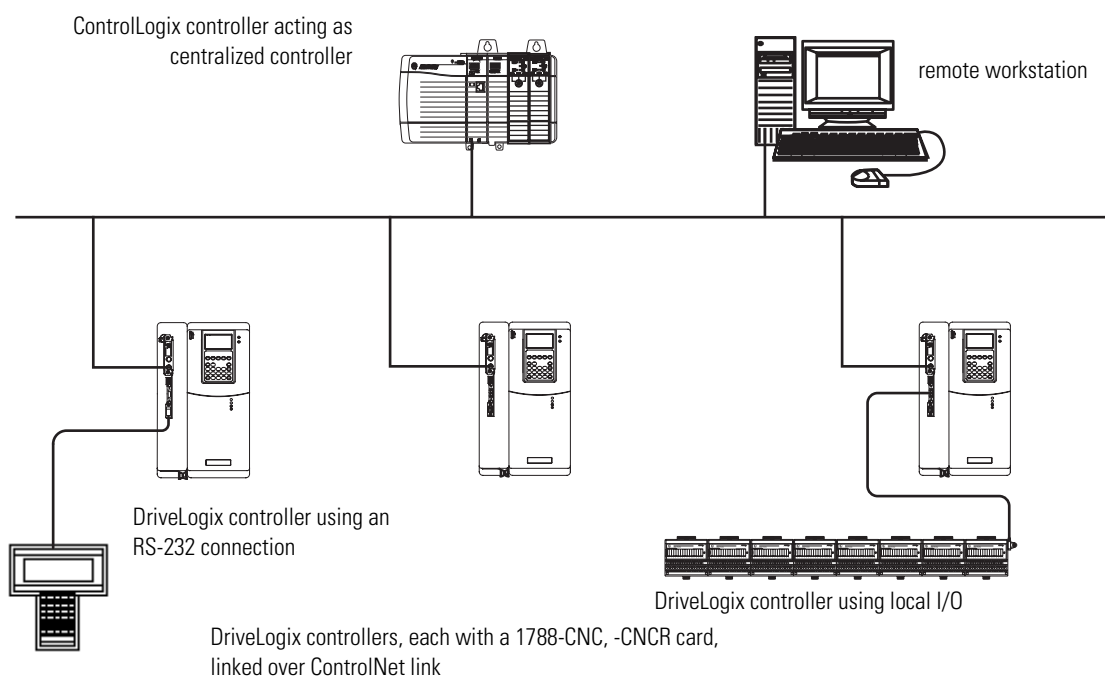
- version 12 and later of the controller firmware
- version 12 of RSLogix 5000 programming software
- version 2.02 of DriveExecutive programming software

Getting Started

Introduction

This chapter introduces the DriveLogix controller and provides a quick overview on creating and downloading a project. The steps in this chapter introduce the basic aspects of the DriveLogix controller.

The DriveLogix controller offers state-of-art control, communications, and I/O elements in an embedded control package.



This example DriveLogix system demonstrates:

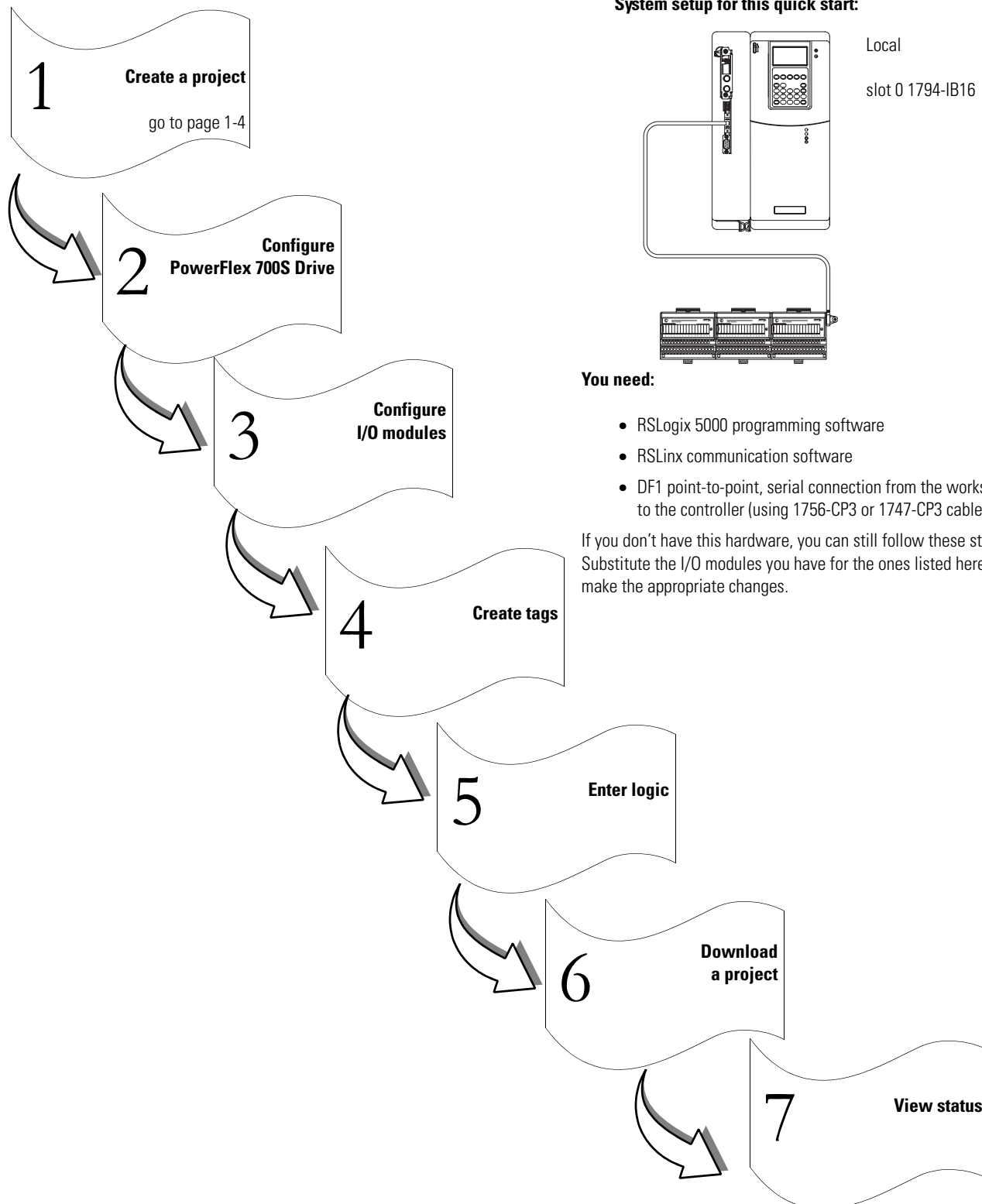
- Centralized control using a ControlLogix™ controller to coordinate several DriveLogix controllers.
- Distributed control using DriveLogix controllers at several locations.
- DriveLogix controller controlling a maximum of 8 local I/O modules.
- Local RS-232 connection for remote upload/download of a controller project, for DF1 master/slave communications, or for ASCII programming.
- Networked DriveLogix controllers using 1788-CNC, -CNCR communication daughtercards to connect ControlNet links.
- Remote programming over ControlNet and EtherNet/IP.

Connecting Battery

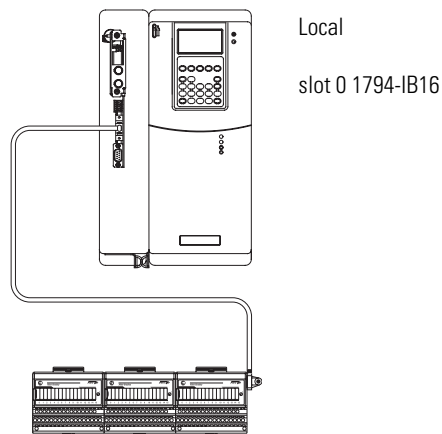
Allen-Bradley ships the DriveLogix controller with the battery installed, but disconnected. You must connect the battery while installing the drive. Refer to *Installing and Maintaining the Battery* on page B-1 and *Access Procedures* on page C-1.

Creating and Downloading a Project

The following diagram illustrates the steps you follow to create and download a project. The remainder of this chapter provides examples of each step.



System setup for this quick start:



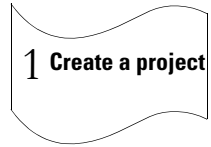
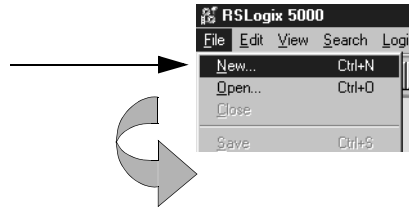
You need:

- RSLogix 5000 programming software
- RSLinx communication software
- DF1 point-to-point, serial connection from the workstation to the controller (using 1756-CP3 or 1747-CP3 cable)

If you don't have this hardware, you can still follow these steps. Substitute the I/O modules you have for the ones listed here and make the appropriate changes.

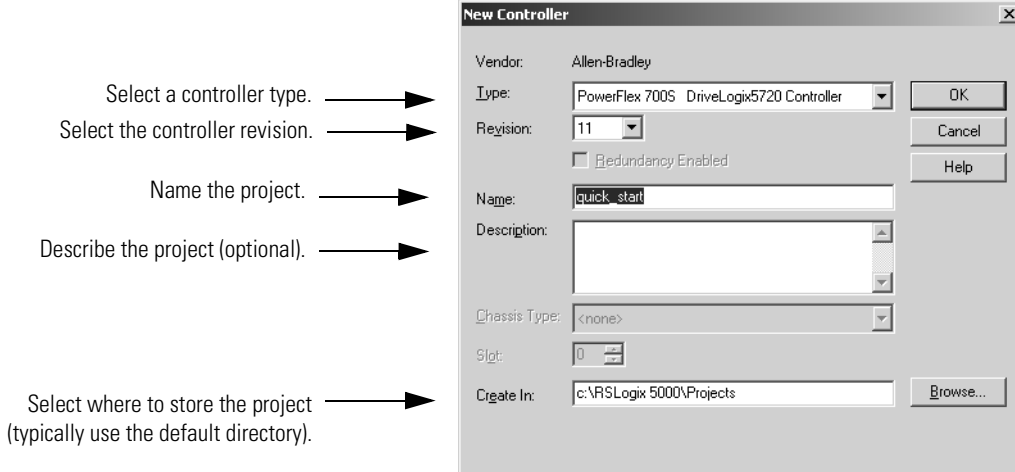
Creating a project

1. Select File →New.



2. Define the project.

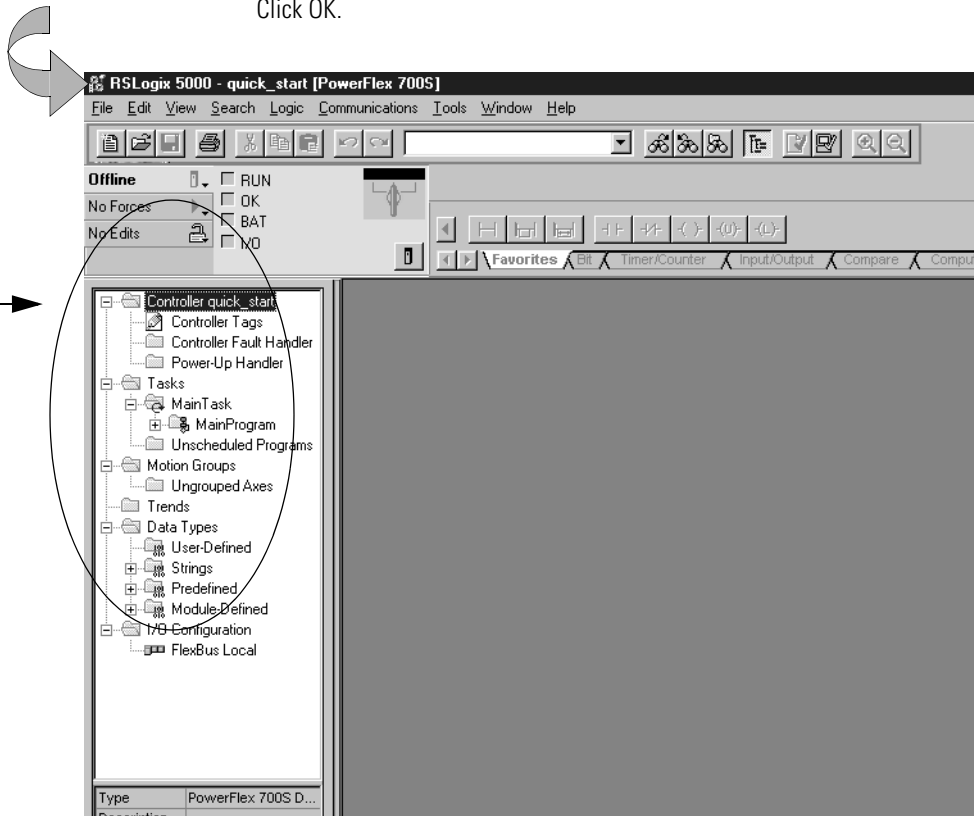
The software uses the project name you enter with an .ACD extension to store your project.



Click OK.

The software creates the new controller and displays:

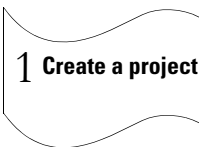
controller organizer



Changing project properties

1. View properties for Controller quick_start.

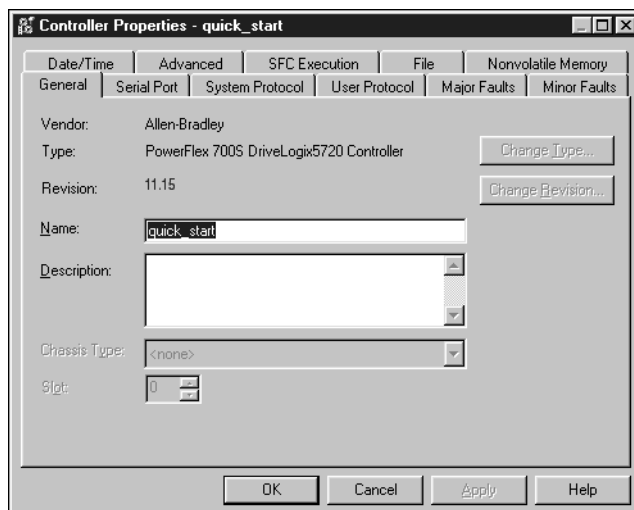
- A. Place the cursor over the Controller quick_start folder. →
- B. Click the right mouse button and select Properties.



2. View the General tab.

The screen defaults to the General tab.

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

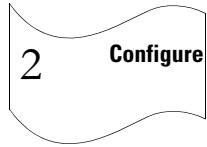


Click OK.

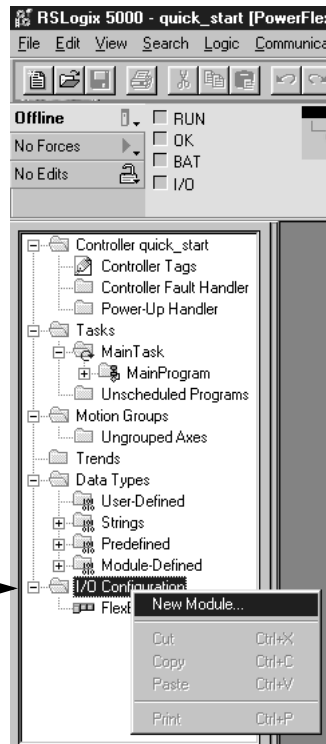
Configuring the host PowerFlex 700S Drive

1. Create the PowerFlex 700S Drive module.

Refer to Chapter 3, "Placing and Configuring the Drive" for more detailed information.

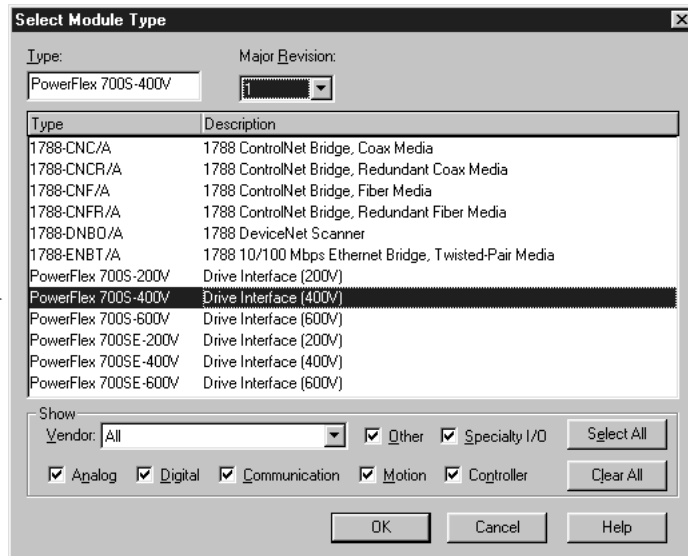


- A. Place the cursor over the I/O Configuration folder.
- B. Click the right mouse button and select New Module.



2. Select the Drive module.

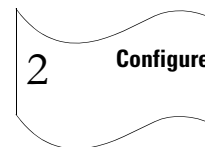
Select the correct drive type. →



Click the OK button

Configuring the host PowerFlex 700S Drive (continued)

3. Identify the drive module.



You should enter a name. →

Describe the module (optional). →

Select the communication format. →

Select the minor revision and →

Specify electronic keying.

Click Next

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

Use default values for this example.

If you do not want to go through each screen in the Create wizard, click Finish

Click Next

Choose the correct drive rating from the pull-down menu

Click Next

Click on Finish to apply and save the changes.

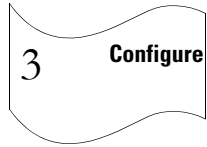
TIP

If no drive ratings appear, download and install DriveExecutive™ Database files from: <http://www.ab.com/drives/data.html>

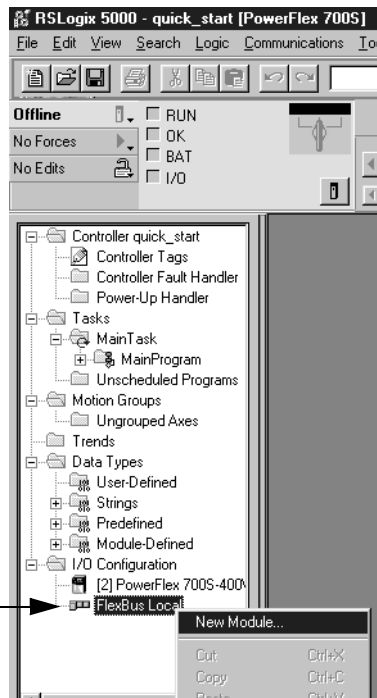
Adding a local input module1

1. Create a new module.

Refer to Chater 4, "Placing and Configuring Local I/O" for more detailed information.

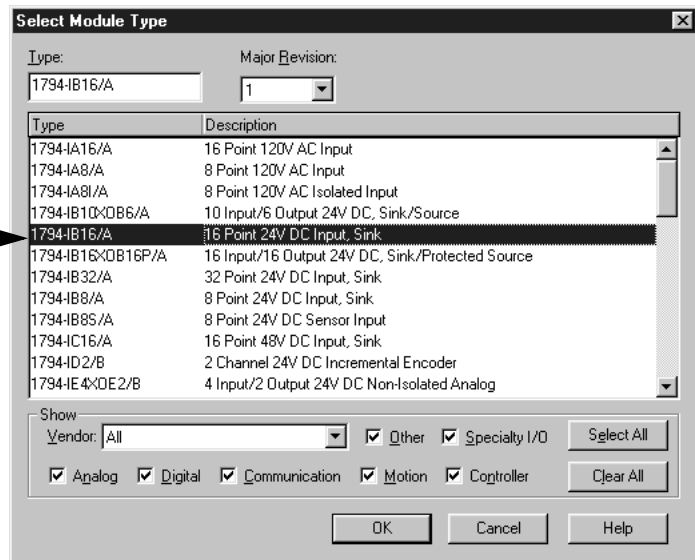


- A. Place the cursor over the local DIN rail (FlexBus Local).
- B. Click the right mouse button and select New Module.



2. Select an input module to add.

Select a catalog number.

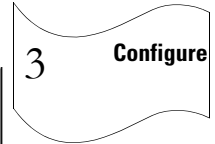


Click OK.

continued

Adding a local input module (*continued*)

3. Identify the input module. These screens are specific to the 1794-IB16 input module.



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



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

Use default values for this example.

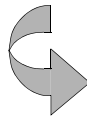
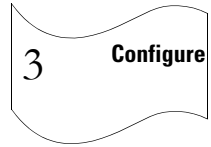
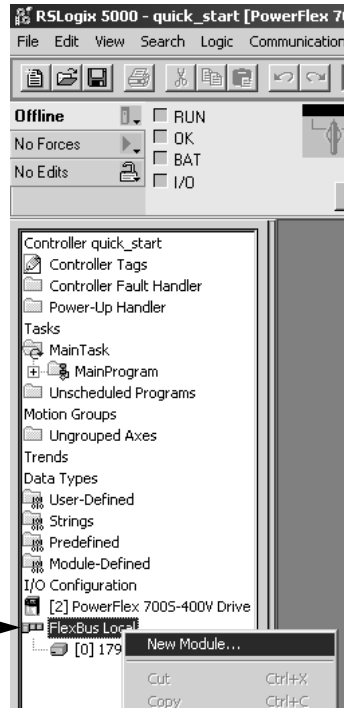
If you do not want to go through each screen in the Create wizard, click Finish

Points	Input Filter Time	
	On->Off	Off->On
0 - 11	0.25 ms	
12 - 15	0.25 ms	

Adding a local output module

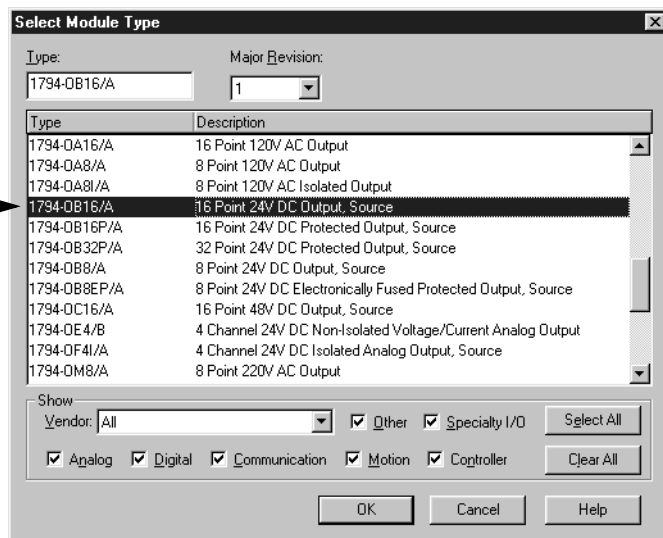
1. Create a new module.

- A. Place the cursor over the local DIN rail (FlexBus Local)
- B. Click the right mouse button and select New Module.



2. Select an output module to add.

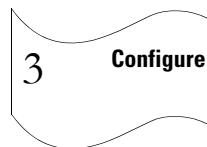
Select a catalog number.



continued

Adding a local output module (continued)

3. **Identify the output module.** These screens are specific to the 1794-OB16 output module.



You should enter a name. →

Describe the module (optional). →

Select the communication format. →

Specify electronic keying. →



4. **Use the Create wizard to configure the output module.** Use default values for this example.

If you do not want to go through each screen in the Create wizard, click

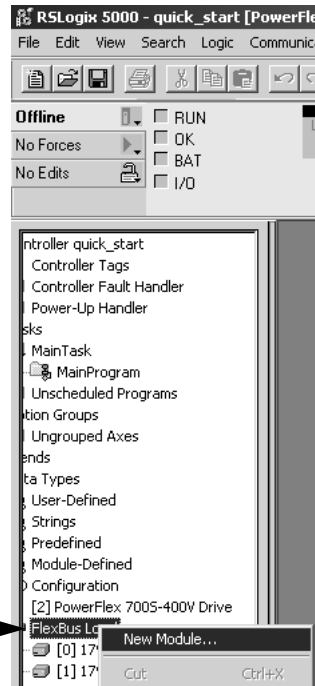
Click Next. →

Point	Safe State Value
0	Off
1	Off
2	Off
3	Off
4	Off
5	Off
6	Off
7	Off

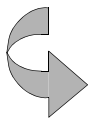
Click Finish. →

Adding a local analog module

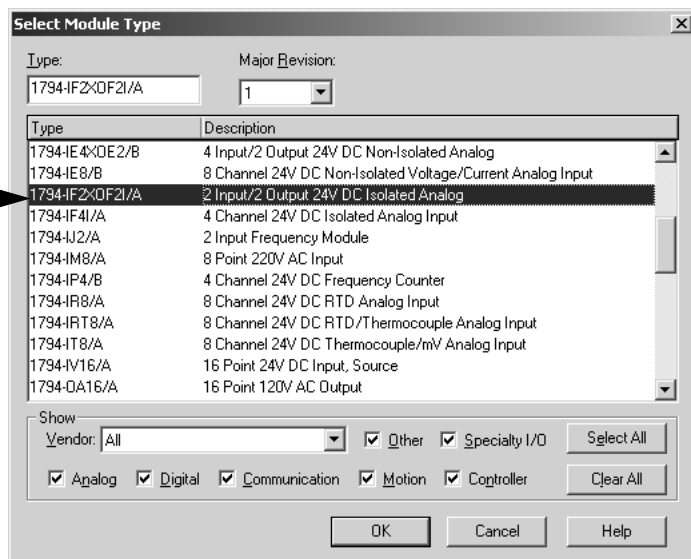
1. Create a new module.



- A. Place the cursor over the local DIN rail (FlexBus Local)
- B. Click the right mouse button and select New Module.



2. Select an output module to add.



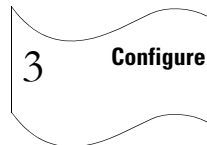
Select a catalog number.
For this quick start example, select



continued

Adding a local analog module (continued)

3. **Identify the output module.** These screens are specific to the 1794-OB16 output module.

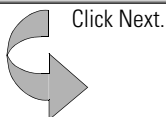


You should enter a name. →

Describe the module (optional). →

Select the communication format. →

Specify electronic keying. →



4. **Use the Create wizard to configure the output module.** Use default values for this example.

If you do not want to go through each screen in the Create wizard, click

Click Next.

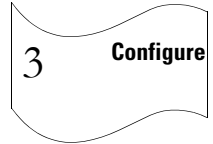
Input Channel	Calibrate this Channel	Calibration Range
0	<input checked="" type="checkbox"/>	0 to 20 mA - Binary
1	<input checked="" type="checkbox"/>	0 to 20 mA - Binary

Output Channel	Calibrate this Channel	Calibration Range
0	<input checked="" type="checkbox"/>	0 to 20 mA - Binary
1	<input checked="" type="checkbox"/>	0 to 20 mA - Binary

Click Finish.

Changing module properties

1. View properties for the module.



- A. Place the cursor over the 1794-IB16 module. →
- B. Click the right mouse button and select Properties.

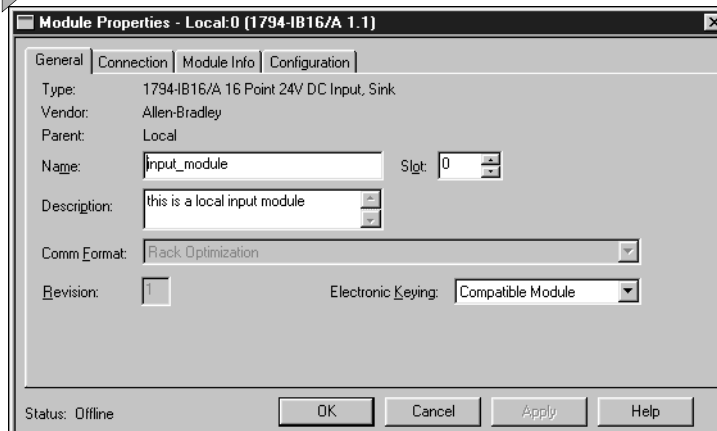


2. View the General tab.

The screen defaults to the General tab.

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

Click OK.

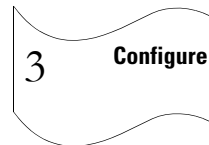


The tabs that appear depend on the type of module.

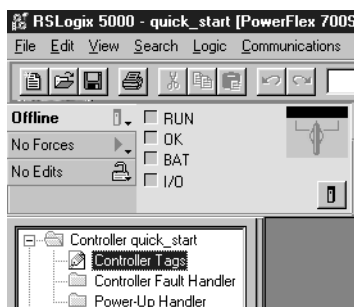
Important: If you want to change the communication format of a module, you must

Viewing I/O tags

1. View the tags for the controller.



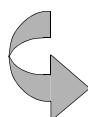
Place the cursor on the Controller Tags folder and double-click.



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

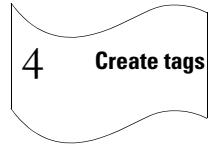
Tag Name	Value	Force Mask	Style	Type	Description
drive_module:I	{...}	{...}		AB:DRIVE_VELO...	
drive_module:O	{...}	{...}		AB:DRIVE_VELO...	
Local:O:C	{...}	{...}		AB:1794_IB16:C:0	
Local:O:I	2#0000_000...		Binary	INT	
Local:O:C	{...}	{...}		AB:1794_DO16:C:0	
Local:O:I	2#0000_000...		Binary	INT	
Local:O:C	{...}	{...}		AB:1794_IF2XOF...	
Local:O:I	{...}	{...}		AB:1794_IF2XOF...	
Local:O:C	{...}	{...}		AB:1794_IF2XOF...	
Local:I	{...}	{...}		AB:1794_AVB_8S...	
Local:O	{...}	{...}		AB:1794_AVB_8S...	

Click the Edit Tags tab.



Creating other tags

1. Create a tag.



Scope: quick_start(controller) Shgw: Show All Sort: Tag Name		P	Tag Name	Alias For	Base Tag	Type	Style	Description
			Drive:I			AB:DRIVE_VELO...		
			Drive:O			AB:DRIVE_VELO...		
			Local:O:C			AB:1794_IB16:C:0		
			Local:O:I	Local:I.Data[0]	Local:I.Data[0]	INT	Binary	
			Local:1:C			AB:1794_DD16:C:0		
			Local:1:O	Local:O.Data[1]	Local:O.Data[1]	INT	Binary	
			Local:2:C			AB:1794_IF2XOF...		
			Local:2:I			AB:1794_IF2XOF...		
			Local:2:O			AB:1794_IF2XOF...		
			Local:I			AB:1794_AVB_8S...		
			Local:O			AB:1794_AVB_8S...		
			timer_1			TIMER	Decimal	

Enter the name of the new tag.

Tab to this column and select the data type.

2. Select the data type.

Select TIMER.

Click OK.

The software displays the tag.

Click + to display the members of the TIMER structure.

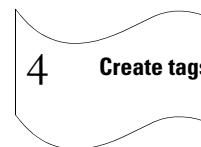
Scope: quick_start(controller) Shgw: Show All Sort: Tag Name		P	Tag Name	Alias For	Base Tag	Type	Style	Description
			Drive:I			AB:DRIVE_VELO...		
			Drive:O			AB:DRIVE_VELO...		
			Local:O:C			AB:1794_IB16:C:0		
			Local:O:I	Local:I.Data[0]	Local:I.Data[0]	INT	Binary	
			Local:1:C			AB:1794_DD16:C:0		
			Local:1:O	Local:O.Data[1]	Local:O.Data[1]	INT	Binary	
			Local:I			AB:1794_AVB_8S...		
			Local:O			AB:1794_AVB_8S...		
			+ timer_1			TIMER		
			+ timer_1.PRE			DINT	Decimal	
			+ timer_1.ACC			DINT	Decimal	
			- timer_1.EN			BOOL	Decimal	
			- timer_1.TT			BOOL	Decimal	
			- timer_1.DN			BOOL	Decimal	
			- timer_1.FS			BOOL	Decimal	
			- timer_1.LS			BOOL	Decimal	
			- timer_1.OV			BOOL	Decimal	
			- timer_1.ER			BOOL	Decimal	

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:0:I.Data.1.

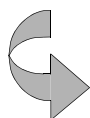


P	Tag Name	Alias For	Base Tag	Type	Style	Description
	Drive:I			AB:DRIVE_VELO...		
	Drive:O			AB:DRIVE_VELO...		
	Local:O:C			AB:1794_IB16:C:0		
	Local:O:I	Local:I.Data[0]	Local:I.Data[0]	INT	Binary	
	Local:1:C			AB:1794_D016:C:0		
	Local:1:O	Local:O.Data[1]	Local:O.Data[1]	INT	Binary	
	Local:2:C			AB:1794_IF2XOF...		
	Local:2:I			AB:1794_IF2XOF...		
	Local:2:O			AB:1794_IF2XOF...		
	Local:I			AB:1794_AVB_8S...		
	Local:O			AB:1794_AVB_8S...		
	timer_1			TIMER		
	input_1	Local:O:I.1		BOOL	Binary	

Enter the name of the tag.

Tab here or click in the box.

2. Select an input data word.



Click here to display a grid of bits and select the input bit.

P	Tag Name	Alias For	Base Tag	Type	Style	Description
	Drive:I			AB:DRIVE_VELO...		
	Drive:O			AB:DRIVE_VELO...		
	Local:O:C			AB:1794_IB16:C:0		
	Local:O:I	Local:I.Data[0]	Local:I.Data[0]	INT	Binary	
	Local:1:C			AB:1794_D016:C:0		
	Local:1:O	Local:O.Data[1]	Local:O.Data[1]	INT	Binary	
	Local:I			AB:1794_AVB_8S...		
	Local:O			AB:1794_AVB_8S...		
	timer_1			TIMER		
	input_1	Local:O:I.1	Local:I.Data[0] 0	BOOL	Decimal	

Tag Name	Data Type	Description
Drive:O	AB:DRIV...	
input_1	BOOL	
Local:O:C	AB:1794...	
Local:O:I	INT	
Local:I	AB:1794...	
Local:O	AB:1794...	
timer_1	TIMER	

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15

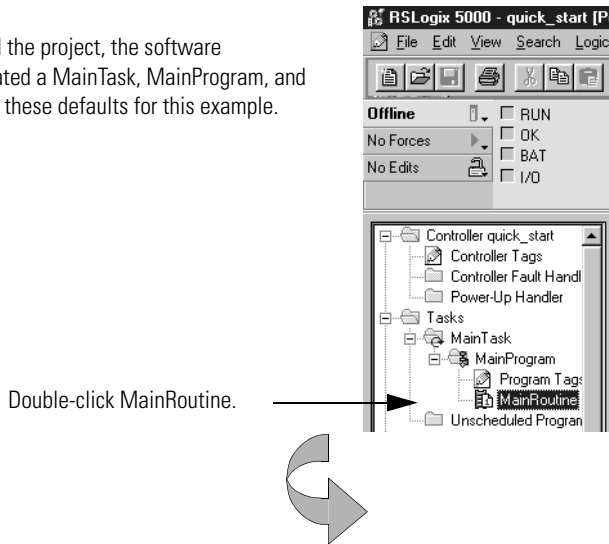
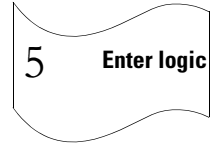
3. Repeat steps 1 and 2 above to create an alias tag *output_1* for Local:1:O.Data.1

P	Tag Name	Alias For	Base Tag	Type	Style	Description
	Drive:I			AB:DRIVE_VELO...		
	Drive:O			AB:DRIVE_VELO...		
	Local:O:C			AB:1794_IB16:C:0		
	Local:O:I	Local:I.Data[0]	Local:I.Data[0]	INT	Binary	
	Local:1:C			AB:1794_D016:C:0		
	Local:1:O	Local:O.Data[1]	Local:O.Data[1]	INT	Binary	
	Local:2:C			AB:1794_IF2XOF...		
	Local:2:I			AB:1794_IF2XOF...		
	Local:2:O			AB:1794_IF2XOF...		
	Local:I			AB:1794_AVB_8S...		
	Local:O			AB:1794_AVB_8S...		
	timer_1			TIMER		
	input_1	Local:O:I.1	Local:I.Data[0] 1	BOOL	Decimal	
	output_1	Local:1:O.1		BOOL	Decimal	

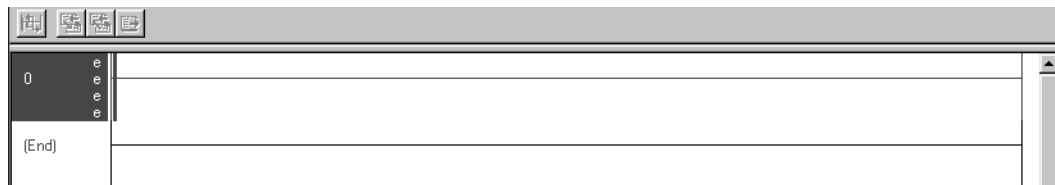
Entering logic

1. Use default task, program, and routine.

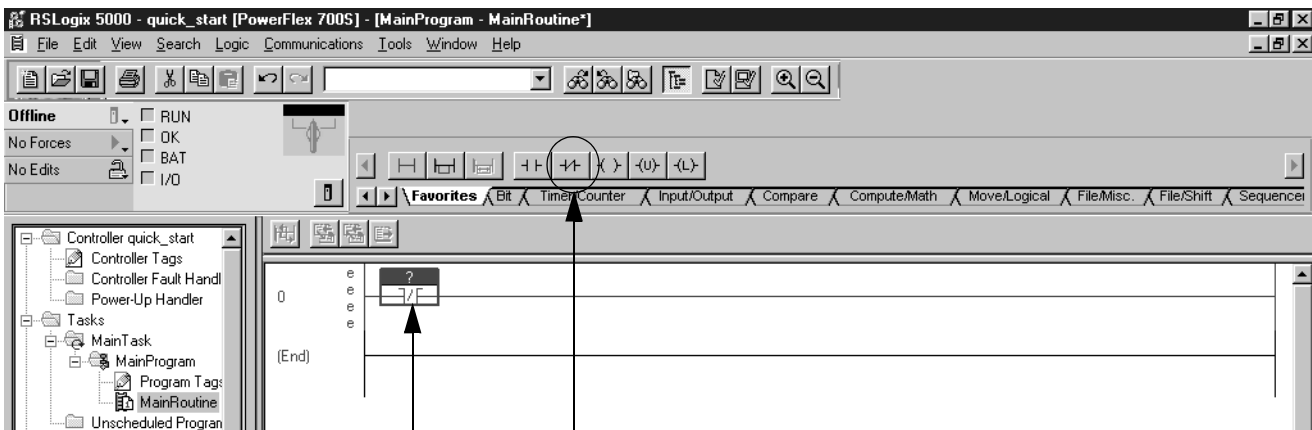
When you created the project, the software automatically created a MainTask, MainProgram, and MainRoutine. Use these defaults for this example.



The software displays an empty routine.



2. Enter an XIO instruction.

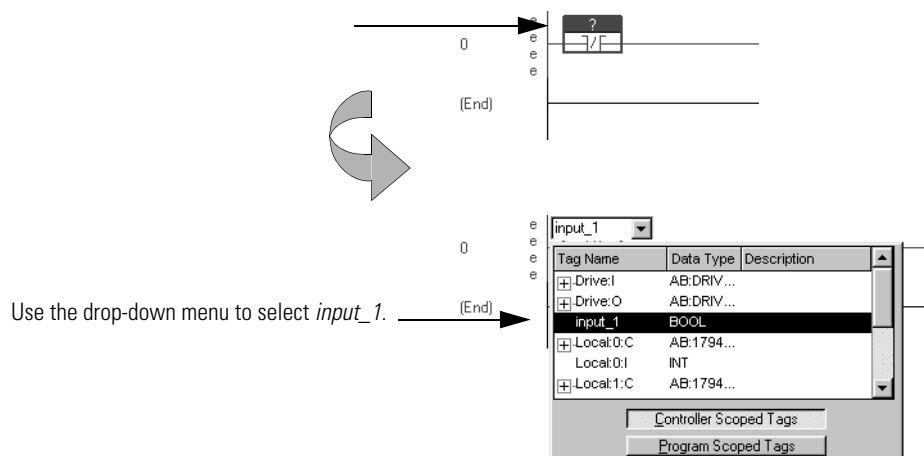


Drag and drop the XIO instruction on an empty rung.

Entering logic (*continued*)

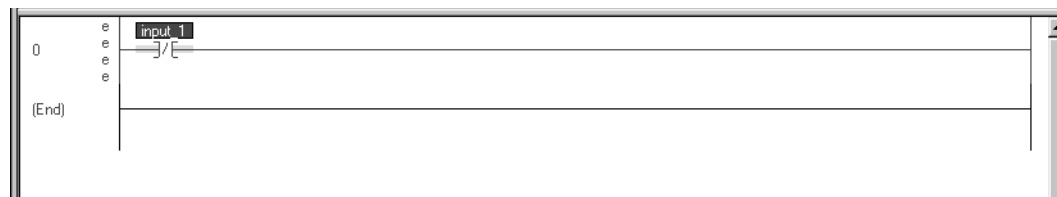
3. Assign a tag to the XIO instruction.

Double-click the tag area of the instruction.

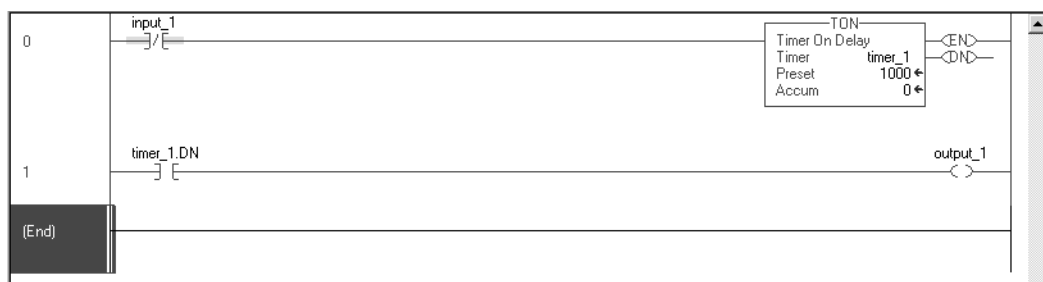


Use the drop-down menu to select *input_1*.

The software displays an incomplete rung.



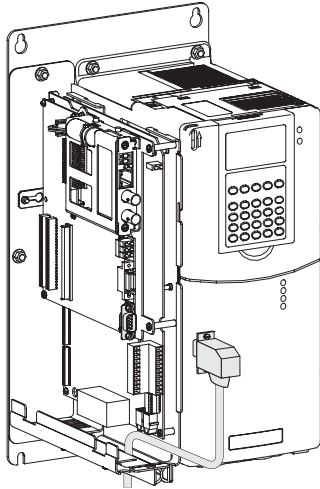
4. Enter this logic.



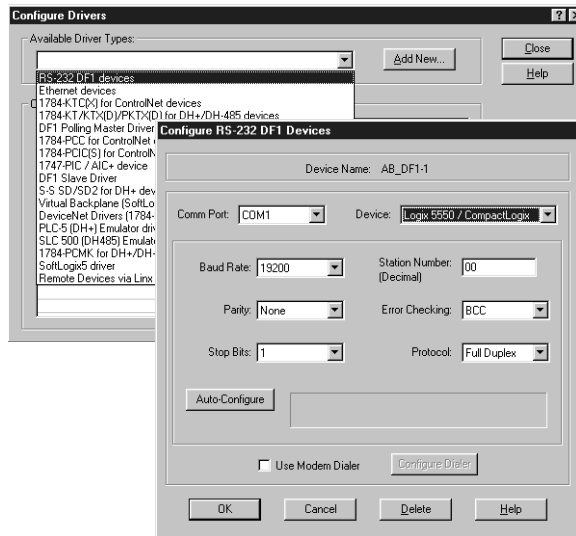
5. To save the project, from the File menu, select Save.

Downloading a project

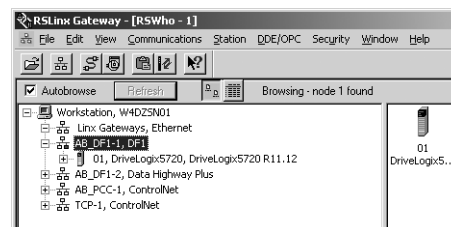
1. Make a serial connection from the workstation to the controller.



2. Configure an RSLinx communication driver:
 - A. In RSLinx software, select Communication →Configure Driver.
 - B. From the Available Driver Types list, select “RS-232 DF1 Devices” and click Add New.
 - C. Select the “Logix5550/CompactLogix serial port” and specify the COM port. Click Autoconfigure to have the software determine the remaining serial settings.



3. Download the project from the Communications menu:
 - A. In RSLogix 5000 software, select Communication →“Who Active”.
 - B. Expand the DF1 network and select your controller.
 - C. Click Download. Confirm the download when prompted.

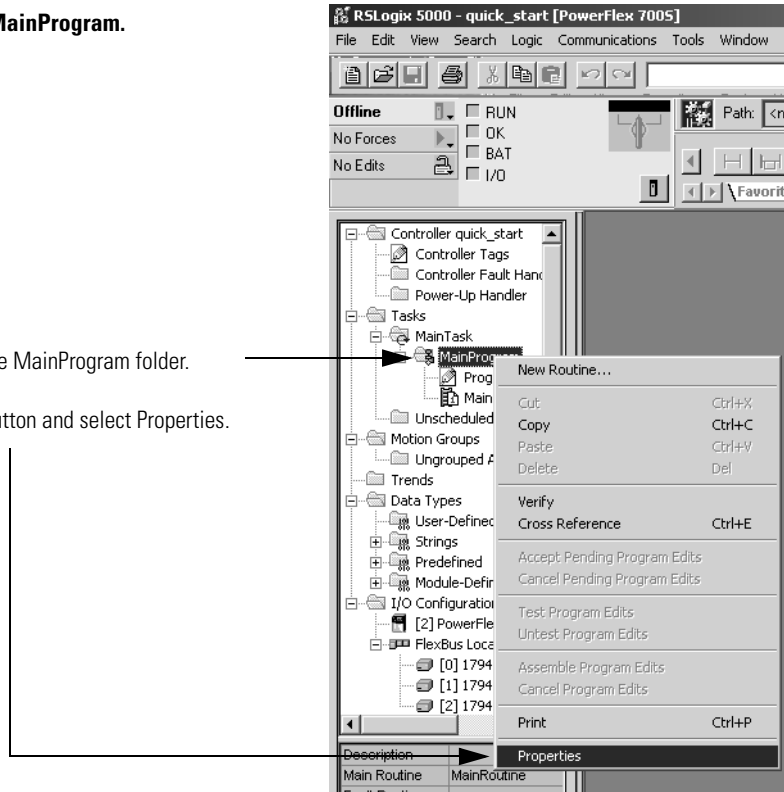


4. Place the controller in Remote Run mode.

Viewing program scan time

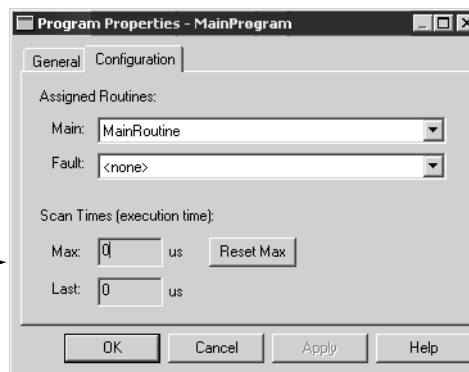
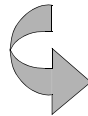
1. View properties for the MainProgram.

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



2. Select the Configuration tab.

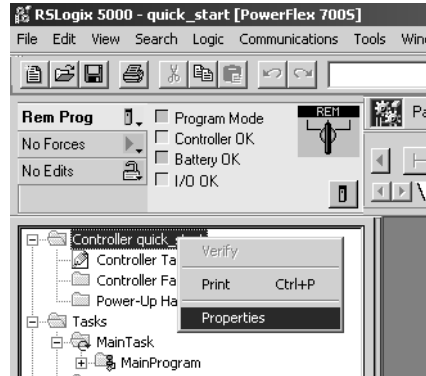
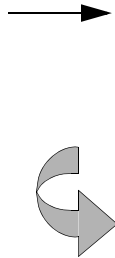
The Configuration tab displays the maximum and last scan times for the program.



Viewing controller memory usage

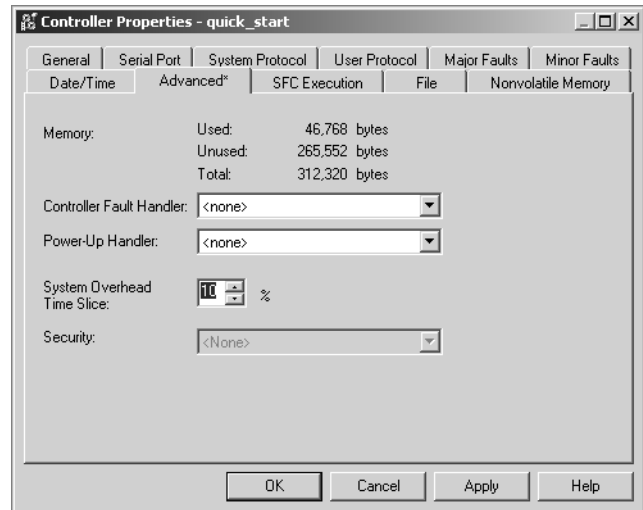
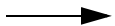
1. View properties for Controller quick_start.

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



2. Select the Advanced tab.

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



Important: The amount of memory that the software displays includes both the user available memory and the memory reserved for overhead. See the specifications for your controller to determine how much memory you have available for programming. This dialog box might display a higher number, but the additional memory is required by system overhead and may not be available for programming.

What To Do Next

Once your controller is installed and operating, you can use RSLogix™ 5000 programming software to develop and test your control application.

Use the remaining chapters in this manual as reference material for how the DriveLogix controller operates in the Logix environment.

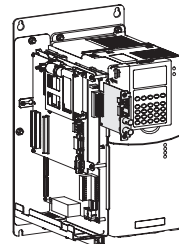
What Is DriveLogix?

Using This Chapter

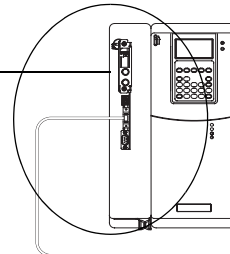
The DriveLogix controller is part of the Logix environment. The DriveLogix controller provides a distributed control system built on these components:

- DriveLogix controller that supports the Logix instructions.
- RSLogix 5000 programming software that supports every Logix controller.
- Direct connection to host PowerFlex 700S drive.
- FLEX™ I/O modules that provide a compact, DIN-rail mounted I/O system.
- 1788 communication daughtercard that provides communication over a standards-based ControlNet, EtherNet/IP or DeviceNet network.

1788 communication daughtercard installs directly in the DriveLogix controller.



The same RSLogix 5000 programming software supports program development for all Logix



The DriveLogix controller supports FLEX I/O modules.



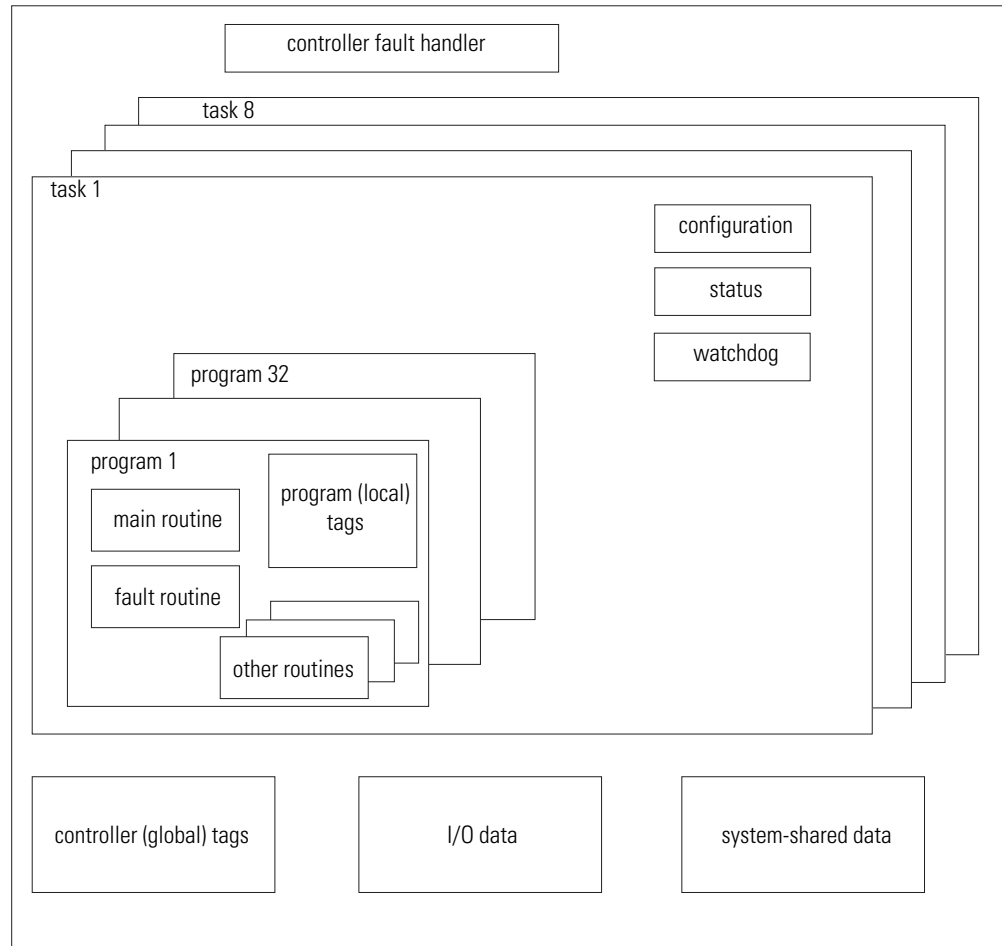
For information about:	See page
developing programs	2-2
how the DriveLogix system uses connections	2-9
selecting a system overhead percentage	2-22

Developing programs

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

control application



Defining tasks

A task provides scheduling and priority information for a set of one or more programs. You can configure tasks as either continuous or periodic. The DriveLogix controller supports as many as 8 tasks, only one of which can be continuous.

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

Specifying task priorities

Each task in the controller has a priority level. The 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.

The DriveLogix controller uses a dedicated periodic task at priority 7 to process I/O data. This periodic task executes at the fastest RPI (Requested Packet Interval) you have scheduled for the DriveLogix system. Its total execution time is as long as it takes to scan the configured I/O modules.

How you configure your tasks affects how the controller receives I/O data. Tasks at priorities 1-6 can starve the dedicated I/O task; tasks at priority 8-15 can be starved by the dedicated I/O task.

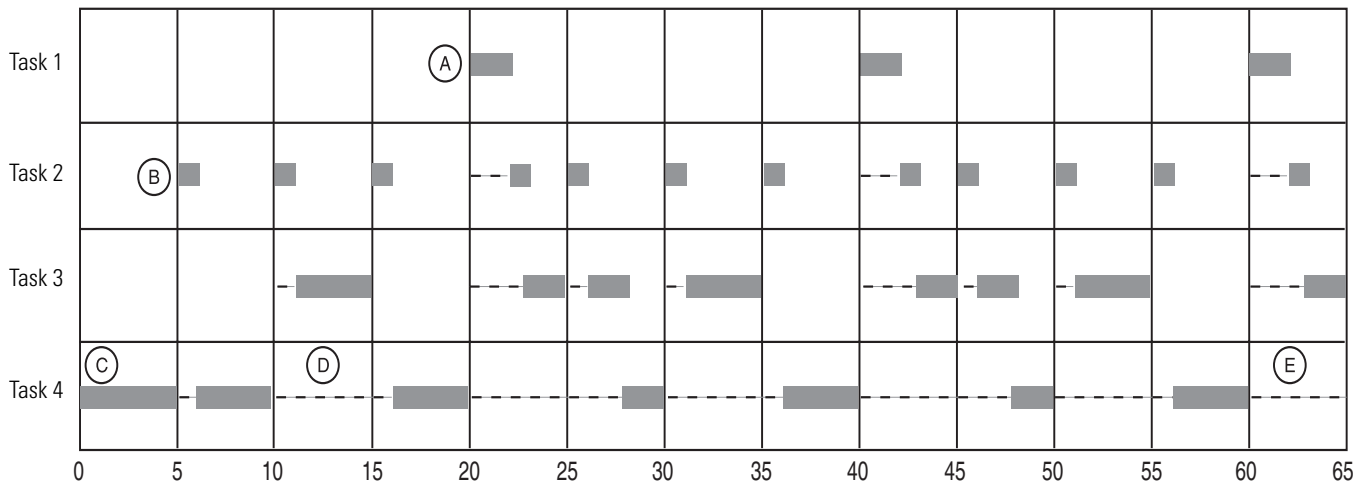
TIP

For typical applications, the periodic task priority should be set at a priority level lower than 7. (Remember, priority levels are ranked in ascending order—a priority lower than 7 means a priority level of 8-15.) The default priority of 10 should be sufficient for most applications.

If a periodic task must execute without interruption from the I/O update task, set the priority level higher than 7 (i.e. 1-6).

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

Task:	Priority Level:	Task Type:	Actual Execution Time:	Worst Case Execution Time:
1	5	20ms periodic task	2ms	2ms
2	7	dedicated I/O task	1ms	3ms
5ms fastest RPI				
3	10	10ms periodic task	4ms	8ms
4	none (lowest)	continuous task	25ms	60ms



Notes:

- A.** The highest priority task interrupts all lower priority tasks.
- B.** The dedicated I/O task can be interrupted by tasks with priority levels 1-6. The dedicated I/O task interrupts tasks with priority levels 8-15. This task runs at the fastest RPI rate scheduled for the DriveLogix system (5ms in this example).
- C.** The continuous task runs at the lowest priority and is interrupted by all other tasks.
- D.** A lower priority task can be interrupted multiple times by a higher priority task.
- E.** When the continuous task completes a full scan it restarts immediately, unless a higher priority task is running.

Defining programs

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

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

Defining routines

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

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

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

Using the Event Task

The event task is available with DriveLogix controllers using firmware version 12.x or greater. Previously, the only tasks available were the continuous task and periodic task. However, the event task offers DriveLogix controller users a task that executes a section of logic immediately when an event occurs.

An event task performs a function only when a specific event (trigger) occurs. Whenever the trigger for the event task occurs, the event task:

- interrupts any lower priority tasks
- executes one time
- returns control to where the previous task left off

For DriveLogix controller, the event task trigger can only be the EVENT instruction.

Prioritizing Periodic and Event Tasks

Although a DriveLogix project can contain up to 8 tasks, the controller executes only one task at a time. If a periodic or event task is triggered while another task is currently executing, the priority of each task tells the controller what to do.

The DriveLogix controller has 15 priority levels for its tasks. To assign a priority to a task, use the guidelines described in Table 2.1.

Table 2.1

If you want:	Then	Notes:
this task to interrupt another task	Assign a priority number that is less than (higher priority) the priority number of the other task.	<ul style="list-style-type: none"> • A higher priority task interrupts all lower priority tasks. • A higher priority task can interrupt a lower priority task multiple times.
another task to interrupt this task	Assign a priority number that is greater than (lower priority) the priority number of the other task.	
this task to share controller time with another task	Assign the same priority number to both tasks.	The controller switches back and forth between each task and executes each one for 1ms.

Triggering the Event Task

To trigger an event task based on conditions in your logic, use the EVENT Instruction trigger.

Let an event trigger this task. →

Let an EVENT instruction trigger the task. →

No tag is required. →

Task Properties - Task_1

General Configuration Program Schedule Monitor

Type: Event

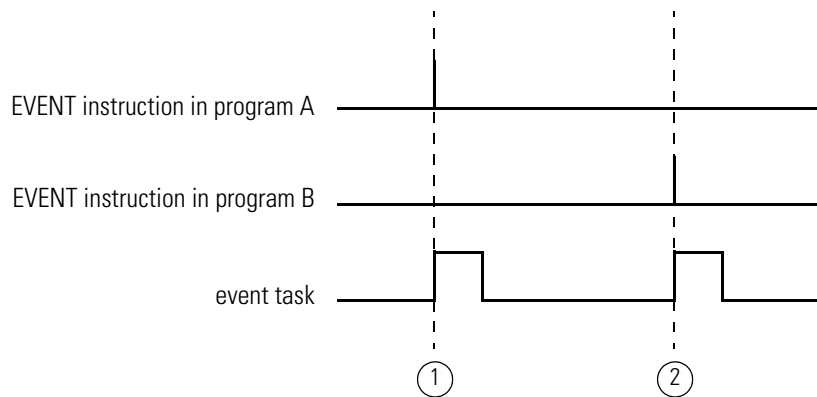
Trigger: EVENT Instruction Only

Tag: <none>

Execute Task If No Event Occurs Within 1000.000 ms

Priority: 1 (Lower Number Yields Higher Priority)

The *EVENT Instruction Only* trigger requires that you use a Trigger Event Task (EVENT) instruction to trigger the task. You can use an EVENT instruction from multiple points in your project. Each time the instruction executes, it triggers the specified event task.



Description:

- ① Program A executes an EVENT instruction.
The event task that is specified by the EVENT instruction executes one time.

- ② Program B executes an EVENT instruction.
The event task that is specified by the EVENT instruction executes one time.

Programmatically Determine if an EVENT Instruction Triggered a Task

To determine if an EVENT instruction triggered an event task, use a Get System Value (GSV) instruction to monitor the Status attribute of the task.

Table 2.2 Status Attribute of the TASK Object

Attribute:	Data Type:	Instruction:	Description:						
Status	DINT	GSV	Provides status information about the task. Once the controller sets a bit, you must manually clear the bit to determine if another fault of that type occurred.						
		SSV	<table border="1"> <thead> <tr> <th>To determine if:</th> <th>Examine this bit:</th> </tr> </thead> <tbody> <tr> <td>An EVENT instruction triggered the task (event task only).</td> <td>0</td> </tr> <tr> <td>A timeout triggered the task (event task only).</td> <td>1</td> </tr> <tr> <td>An overlap occurred for this task.</td> <td>2</td> </tr> </tbody> </table>	To determine if:	Examine this bit:	An EVENT instruction triggered the task (event task only).	0	A timeout triggered the task (event task only).	1
To determine if:	Examine this bit:								
An EVENT instruction triggered the task (event task only).	0								
A timeout triggered the task (event task only).	1								
An overlap occurred for this task.	2								

The controller does not clear the bits of the Status attribute once they are set.

- To use a bit for new status information, you must manually clear the bit.
- Use a Set System Value (SSV) instruction to set the attribute to a different value.

Checklist for an EVENT Instruction Task

For this:	Make sure you:
<input type="checkbox"/> 1. EVENT instruction	Use a Trigger Event Task (EVNT) instruction at each point in your logic that you want to trigger the event task.
<input type="checkbox"/> 2. Task priority	Configure the event task as the highest priority task. If a periodic task has a higher priority, the event task may have to wait until the periodic task is done.
<input type="checkbox"/> 3. Number of event tasks	Limit the number of event tasks. Each additional task reduces the processing time that is available for other tasks. This could cause an overlap.
<input type="checkbox"/> 4. Automatic Output Processing	For an event task, you can typically disable automatic output processing (default). This reduces the elapsed time of the task.

For more information on using the event task, see Logix5000 Controllers Common Procedures programming manual, publication 1756-PM001.

How the DriveLogix System Uses Connections

The DriveLogix system uses a connection to establish a communication link between two devices. The DriveLogix system has enough internal resources to support a connection to every local I/O module and 32 connections through the daughtercard (e.g. the 1788-ENBT card). **However, the daughtercard's connection limit is the limiting factor when sizing a system.**

Connections can be:

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

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

Method:	Description:
scheduled connection <ul style="list-style-type: none"> • most deterministic • unique to ControlNet 	A scheduled connection is unique to ControlNet communications. A scheduled connection lets you send and receive data repeatedly at a predetermined rate, which is the requested packet interval (RPI). For example, a connection to an I/O module is a scheduled connection because you repeatedly receive data from the module at a specified rate. Other scheduled connections include connections to: <ul style="list-style-type: none"> • communication devices • produced/consumed tags On a ControlNet network, you must use RSNetWorx for ControlNet to enable all scheduled connections and establish a network update time (NUT).
unscheduled connection <ul style="list-style-type: none"> • deterministic • used by both ControlNet and EtherNet/IP 	An unscheduled connection is a message transfer between controllers that is triggered by the requested packet interval (RPI) or the program (such as a MSG instruction). Unscheduled messaging lets you send and receive data when needed. <p>All EtherNet/IP connections are unscheduled.</p>
unconnected message <ul style="list-style-type: none"> • least deterministic 	An unconnected message is a message that does not require connection resources. An unconnected message is sent as a single request/response.

The communication module you select determines the number of connections you have available for I/O and messages:

This communication card:	Supports this number of connections:										
1788-CNx	32 connections—the maximum number of scheduled connections is dependent on the RPI: <table border="1" data-bbox="964 470 1284 688"> <thead> <tr> <th>RPI (with 5 ms NUT)</th> <th>Max Scheduled Connections</th> </tr> </thead> <tbody> <tr> <td>5 ms</td> <td>3</td> </tr> <tr> <td>10 ms</td> <td>6</td> </tr> <tr> <td>20 ms</td> <td>13</td> </tr> <tr> <td>40 ms</td> <td>20</td> </tr> </tbody> </table> <p>The remaining connections (or all 32, if you have no scheduled connections) can be used for unscheduled connections</p>	RPI (with 5 ms NUT)	Max Scheduled Connections	5 ms	3	10 ms	6	20 ms	13	40 ms	20
RPI (with 5 ms NUT)	Max Scheduled Connections										
5 ms	3										
10 ms	6										
20 ms	13										
40 ms	20										
1788-ENBT	32 connections - can be used for explicit and implicit connections <p>(all 32 connections are any combination of remote I/O, produce/consume, and messaging connections)</p>										

How you configure connections determines how many remote devices a communication card can support. If you have two communication cards, use one for messaging (e.g. HMI) and the other for control of I/O. While one card can support both functions, performance can improve by separating these functions onto separate cards.

Determining Connections for Produced and Consumed Tags

The DriveLogix controller supports the ability to produce (broadcast) and consume (receive) system-shared tags. Produced and consumed tags each require connections. Over ControlNet, produced and consumed tags are scheduled connections.

This type of tag:	Requires these connections:
produced	By default, a produced tag allows two other controllers to consume the tag, which means that as many as two controllers can simultaneously receive the tag data. The local controller (producing) must have one connection for the produced tag and the first consumer and one more connection for each additional consumer (heartbeat). The default produced tag requires as many connections as there are consumers for the produced tag. For example, if the 3 consumers will consume the produced tag, it requires 3 connections. <p>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.</p>
consumed	Each consumed tag requires one connection for the controller that is consuming the tag.

DriveLogix controllers can produce and consume tags over:

- a ControlNet network
- an EtherNet/IP network.

IMPORTANT

For two controllers to share produced or consumed tags, both controllers must be attached to the same control network (such as a ControlNet or Ethernet/IP network). You cannot bridge produced and consumed tags over two networks.

The total number of tags that can be produced or consumed is limited by the number of available connections and memory. If the controller uses all of its connections for I/O and communication devices, no connections are left for produced and consumed tags.

Determining Connections for Messages

Messages transfer data to other devices, such as other controllers or operator interfaces. Connected messages can leave the connection open (cache) or close the connection when the message is done transmitting. The following table shows which messages use a connection:

This type of message:	And this communication method:	Uses a connection:
CIP data table read or write		X
PLC2, PLC3, PLC5, or SLC (all types)	CIP	
	CIP with Source ID	
	DH+	X
CIP generic	CIP	Optional ⁽¹⁾
block-transfer read or write		X

⁽¹⁾ You can connect CIP generic messages, but for most applications, we recommend you leave CIP generic messages unconnected.

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

If a MSG instruction uses a connection, you have the option to leave the connection open (cache) or close the connection when the message is done transmitting.

If you:	Then:
Cache the connection	The connection stays open after the MSG instruction is done. This optimizes execution time. Opening a connection each time the message executes increases execution time.
Do not cache the connection	The connection closes after the MSG instruction is done. This frees up that connection for other uses.

The controller has the following limits on the number of connections that you can cache:

If you have this software and firmware revision:	Then you can cache:
11.x or earlier	<ul style="list-style-type: none"> • block transfer messages for up to 16 connections • other types of messages up to 16 connections
12.x or later	up to 32 connections

Determining Connections for I/O Modules

The DriveLogix system uses connections to transmit I/O data. These connections can either be direct connections or rack-optimized connection. Over ControlNet, I/O connections are scheduled connections:

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

Connections for local I/O modules

The DriveLogix controller automatically assigns one rack-optimized connection for the local DIN rail. You then configure each I/O module on the DIN rail to either use that rack-optimized connection or to use a direct connection. The rack-optimized connection for the DIN rail exists whether or not you configure the I/O modules to use that rack-optimized connection.

The rack-optimized connection lets you organize all the digital I/O modules on the DIN rail into one connection to the controller. Or you can choose to configure each I/O module to have a direct connection to the controller. Analog I/O modules must have a direct connection to the controller.

It is not as critical to manage the number of connections for local I/O modules as it is for remote devices because the controller supports a direct connection for each possible local I/O device.

Connections for remote devices

To optimize the number of available connections, place remote, digital I/O in the same location and use a rack-optimized connection to the remote adapter that connects the remote I/O to the DriveLogix system.

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

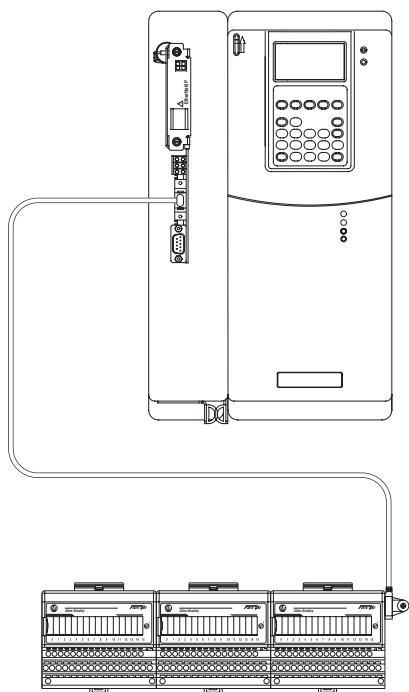
IMPORTANT

It is vital that you manage your connections to remote devices because, while the DriveLogix controller allows up to 250 total connections, the communications cards that connect to remote devices are limited to far fewer connections (i.e. 32 connections for ControlNet or EtherNet/IP).

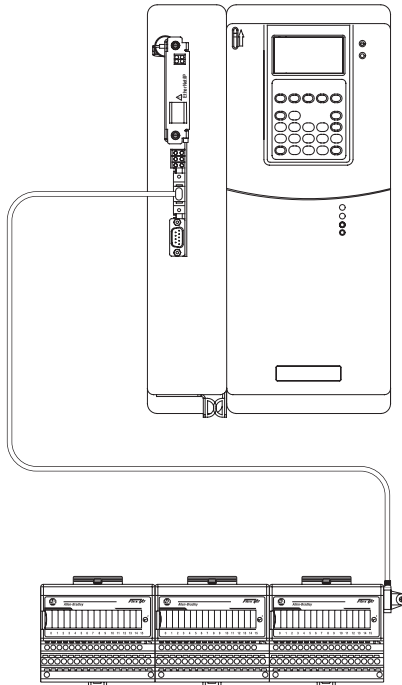
Direct connections for I/O modules

In this example, assume that each I/O module is configured for a direct connection to the controller.

The following table calculates the connections in this example.



Connection:	Amount:
DriveLogix controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for each I/O module	3
DriveLogix controller to host PowerFlex 700S drive	1
total connections used:	4



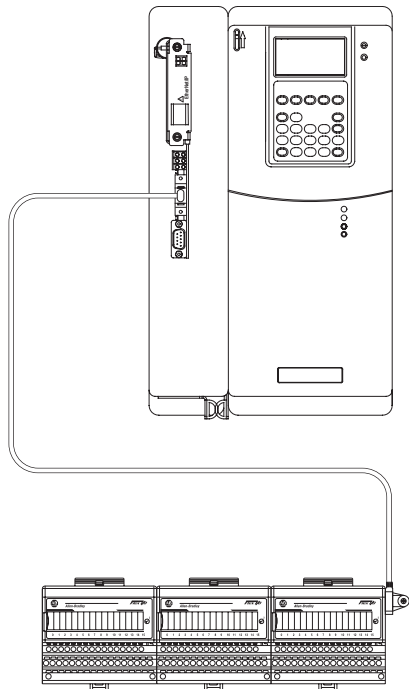
Rack-optimized connections for I/O modules

In this example, assume that each I/O module is configured for a rack-optimized connection to the controller.

The following table calculates the connections in this example.

Connection:	Amount:
DriveLogix controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
DriveLogix controller to host PowerFlex 700S drive	1
total connections used:	2

TIP The rack-optimized connection conserves connections and lowers controller overhead in the I/O update task. However, the rack-optimized connection also limits the status and diagnostic information that is available from the I/O modules and is limited to a single RPI.

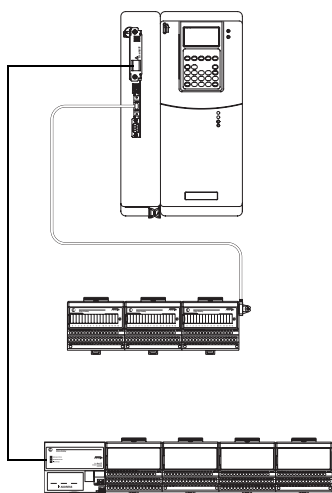


Combining direct and rack-optimized connections

A DIN rail can have both a rack-optimized connection and direct connections. Assume that the I/O modules in slot 0 and slot 1 on the local rail are configured for a rack-optimized connection and that the I/O module in slot 2 is configured for a direct connection.

The following table calculates the connections in this example.

Connection:	Amount:
DriveLogix controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for one I/O module (slot 2)	1
DriveLogix controller to host PowerFlex 700S drive	1
total connections used:	3



Connections to remote ControlNet or EtherNet/IP devices

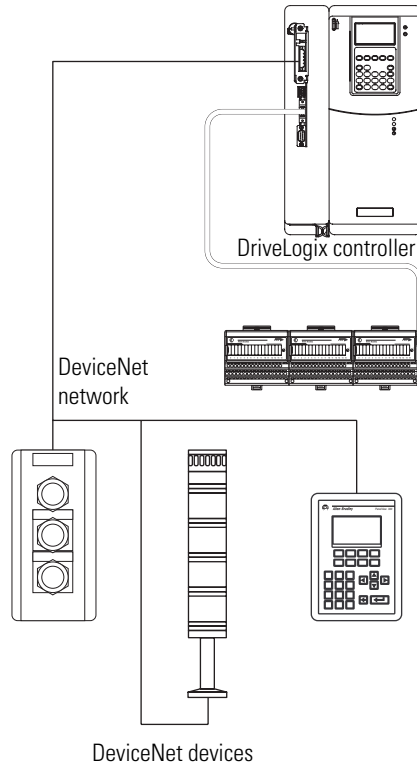
A remote device over ControlNet and EtherNet/IP can be configured as either a rack-optimized connection and direct connection. In this example, the DriveLogix controller uses one rack-optimized connection to communicate with the communication adapter to receive data from the digital I/O modules (two in this example) and uses one direct connection to communicate with the analog module.

The following table calculates the connections in this example.

Connection:	Amount:
DriveLogix controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for one I/O module (slot 2)	1
DriveLogix controller to communication card (1788-CN _x or 1788-ENBT)	0
DriveLogix controller to communication adapter (rack-optimized connection for digital I/O modules)	1
DriveLogix controller to remote analog I/O module	1
DriveLogix controller to host PowerFlex 700S drive	1
total connections used:	5
total connections through the communications card:	2 – This number is within the connection limits of either the 1788-CN _x card (maximum connections = 9) or the 1788-ENBT card (maximum connections = 32).

Connections to DeviceNet devices

In this example the controller uses two connections (one for status and one for I/O) to communicate with the DeviceNet devices through the 1788-DNBO module. The 1788-DNBO module uses a rack-optimized connection to the DeviceNet devices.



The following table calculates the connections in this example.

Connection:	Amount:
DriveLogix controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for one I/O module (slot 2)	1
DriveLogix controller to the 1788-DNBO DeviceNet communication card ⁽¹⁾	2
DriveLogix controller to host PowerFlex 700S drive	1
total connections used:	5

⁽¹⁾ DriveLogix controller connection to remote DeviceNet devices are accounted for in the 2 connections to the 1788-DNBO card.

The 1788-DNBO card does not establish connections to its devices; and therefore, the controller does not establish connections with DeviceNet devices. The 1788-DNBO module acts as a scanner that gathers all the data from its devices and packs that data together into one image that is passed to

the controller. However, the controller can use a MSG instruction to get information directly to or from a DeviceNet device.

Determining Total Connection Requirements

To calculate the total connection requirements for a DriveLogix controller, consider the connections to local I/O modules, the host PowerFlex 700S drive and the connections to remote modules.

Use the following table to tally **local** connections:

Connection Type:	Device Quantity:	Connections per Device:	Total Connections:
connection to host PowerFlex 700S drive	1	1	1
rack-optimized connection for the local DIN rail	2	1	2
I/O module (rack-optimized connection) on local rail		0	
I/O module (direct connection) on local rail		1	
1788-CNx ControlNet communication card		0	0
1788-DNBO communication card (rack-optimized connection)		2	
1788-ENBT Ethernet/IP communication card		0	0
total			

Regardless of how you configure the I/O modules (rack-optimized or direct connect) on the local rail, the controller establishes a rack-optimized connection for the rail. The data for any I/O module configured for a rack-optimized connection is stored in the rack-optimized connection for the rail. You can have 8 I/O modules, for a maximum of 8 direct connections.

Remote connections depend on the communication card. Use the following table to tally remote connections:

Connection Type:	Device Quantity:	Connections per Device:	Total Connections:
remote ControlNet communication device (such as a 1794-ACN15, -ACNR15 or 1756-CNB, -CNBR module) configured as: direct (none) connection or rack-optimized connection listen-only rack-optimization (1756-CNB, -CNBR only)		0 or 1 1	
remote I/O device over ControlNet (direct connection)		1	
remote EtherNet/IP communication device (such as a 1794-AEN adapter or 1756-ENBT module) configured as: direct (none) connection or rack-optimized connection listen-only rack-optimization (1756-ENBT only)		0 or 1 1	
remote I/O device over EtherNet/IP (direct connection)		1	
produced and consumed tag			
produced tag and one consumer		1	
each additional consumer		1	
consumed tag		1	
maximum active message		1	
		total	

After calculating the number of remote connections, make sure they do not exceed the limitations of the communication card:

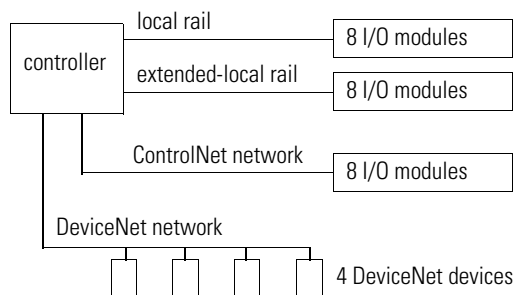
- each ControlNet communication card supports 32 total connections, 9 of which can be scheduled (such as direct I/O connections and produced and consumed tags)
- the Ethernet/IP communication card supports 32 total connections of any type

Even if the total number of connections is within the card limitations, the total number of messages per second must also be within the card limitations. You can estimate the number of messages per second for a connection as $(2 * 1000 \text{ ms}) / \text{RPI}$.

The communication cards support:

- each ControlNet communication card supports 1490 messages/second
- the EtherNet/IP communication card supports 4000 messages/second

This example system has these details:



- I/O modules on the local rail are digital, so configure each module for a rack-optimized connection
- I/O modules on the extended-local rail are analog, so configure each module for a direct connection
- I/O modules on the ControlNet network are 4 digital and 4 analog, so configure each digital module for a rack-optimized connection and each analog module for a direct connection
- there are no produced or consumed tags
- the controller sends 2 messages to other devices on the ControlNet network
- the controller uses 2 connections to the 1788-DNBO module to collect data from the DeviceNet devices

Local connections

Connection Type:	Device Quantity:	Connections per Device:	Total Connections:
rack-optimized connection to DIN rail	2	1	2
connection to host PowerFlex 700S drive	1	1	1
1788-DNBO communication card (rack-optimized connection)	1	2	2
total			5

Remote connections

Connection Type:	Device Quantity:	Connections per Device:	Total Connections:
remote ControlNet communication device configured as a rack-optimized connection	1	1	1
I/O module over ControlNet (direct connection)	4	1	4
cached message	2	1	2
total			7

Downloading Projects

In general, you use the programming software to download a project from your programming computer to the controller. The DriveLogix controller, with expanded memory, supports nonvolatile memory for project storage.

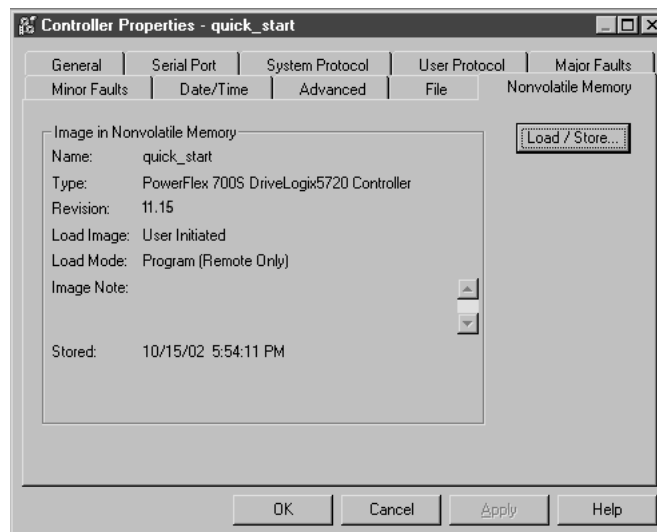
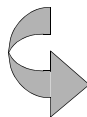
IMPORTANT

Nonvolatile memory stores the contents of user memory at the time that you store the project.

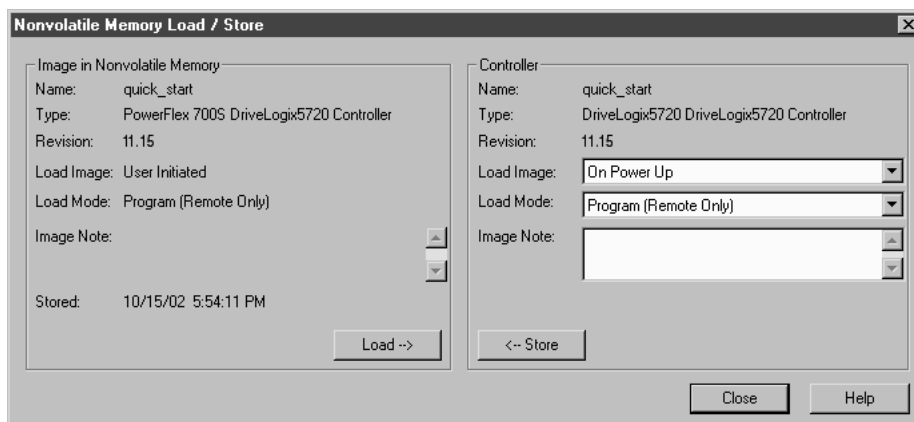
- Changes that you make after you store the project are not reflected in nonvolatile memory.
- If you want to store changes such as online edits, RPI changes, tag values, or this particular DriveLogix controller's portion of the ControlNet network schedule (i.e. the portion of the ControlNet schedule that affects the ControlNet nodes this controller makes connections to), store the project again after you make changes.

To store a project in nonvolatile memory:

1. Go online with the controller.
2. View properties for the controller and select the Nonvolatile Memory tab.



3. Click the Load/Store button and specify when you want the controller to load the project from nonvolatile memory.



4. Click the Load button to load the project from nonvolatile memory into the controller.

You can select:

In this field:	Select this option:	If you want:
Load Image	On Power Up	to load memory when you turn on or cycle the chassis power
	On Corrupt Memory	to load memory whenever there is no project in the controller and you turn on or cycle the chassis power
	User Initiated	only use RSLogix 5000 software to load a project
Load Mode	Remote Program	the controller to go to Remote Program mode after loading from nonvolatile memory
	Remote Run	the controller to go to Remote Run mode after loading from nonvolatile memory

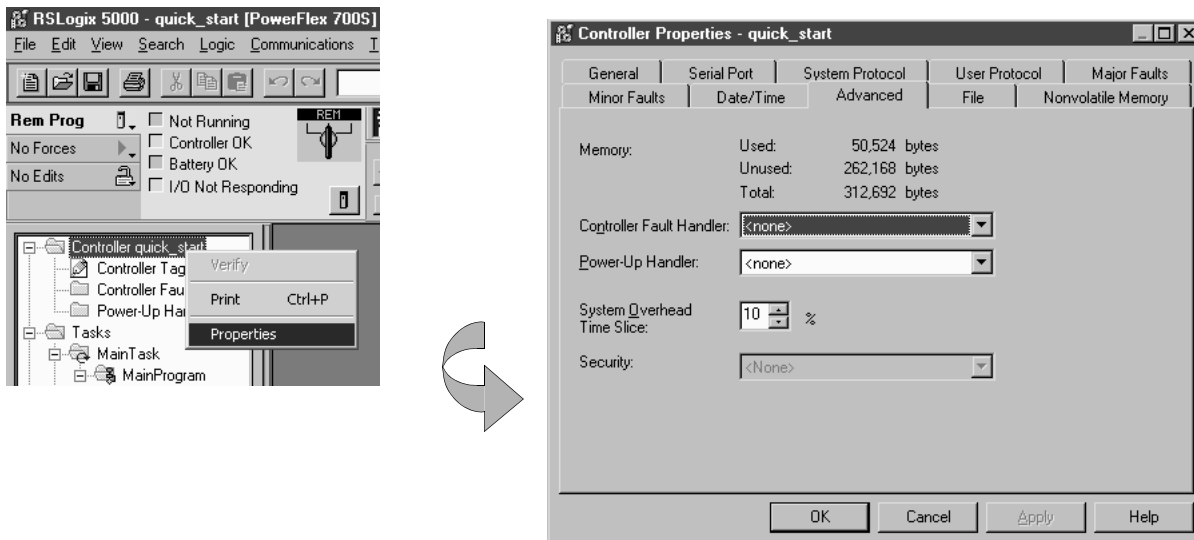
After you load or store to or from nonvolatile memory, RSLogix 5000 software goes offline from the controller.

For details on storing to nonvolatile memory or restoring from nonvolatile memory, see the *Logix5000 Controllers Common Procedures Programming Manual*, publication 1756-PM001.

Selecting a System Overhead Percentage

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

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



System overhead functions include

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

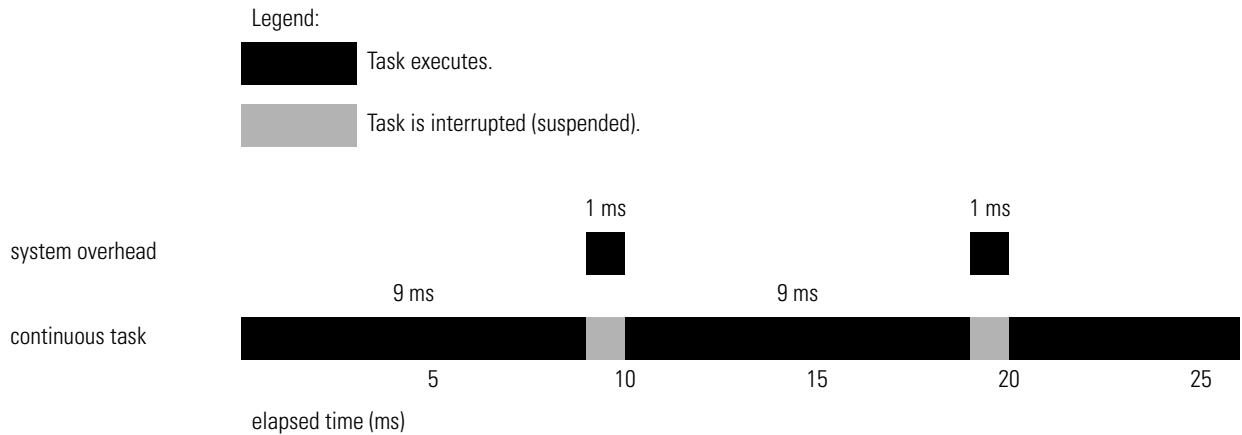
The controller performs system overhead functions for up to 1 ms at a time. If the controller completes the overhead functions in less than 1 ms, it resumes the continuous task.

If communications are not completing fast enough, increase the system overhead percentage. As you increase the system overhead percentage, the overall program scan also increases.

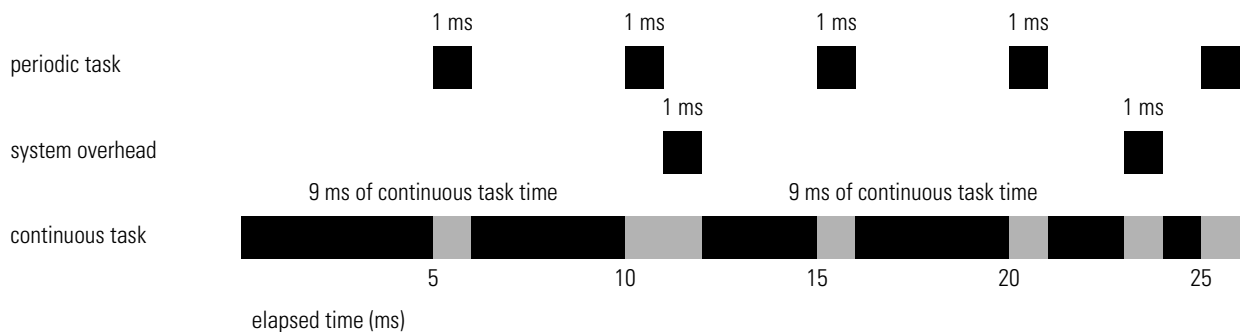
The following table shows the ratio between the continuous task and the system overhead functions:

At this time slice:	The continuous tasks runs for:	And then overhead occurs for up to:
10%	9 ms	1 ms
20%	4 ms	1 ms
33%	2 ms	1 ms
50%	1 ms	1 ms

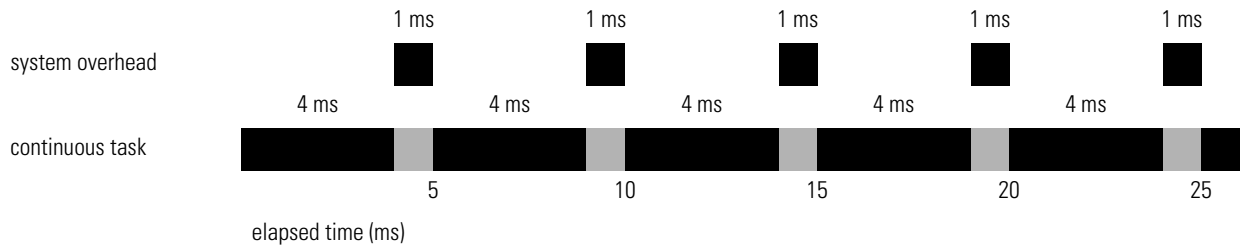
At the default time slice of 10%, system overhead interrupts the continuous task every 9ms (of continuous task time).



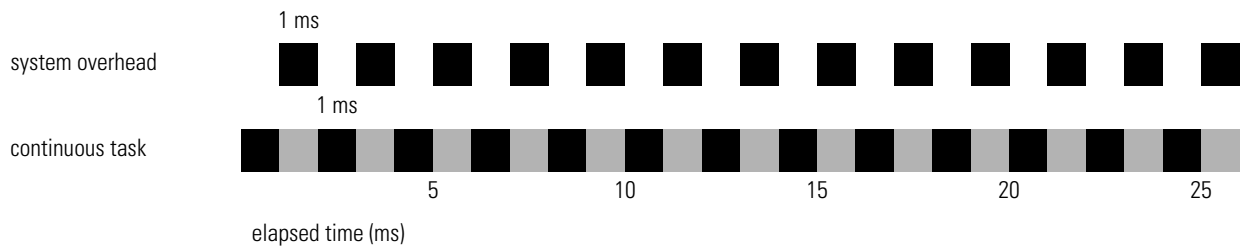
The interruption of a periodic task increases the elapsed time (clock time) between the execution of system overhead.



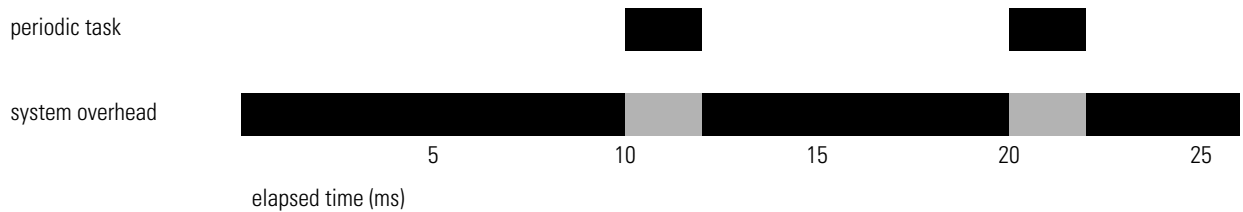
If you increase the time slice to 20%, the system overhead interrupts the continuous task every 4ms (of continuous task time).



If you increase the time slice to 50%, the system overhead interrupts the continuous task every 1ms (of continuous task time).



If the controller only contains a periodic task (s), the system overhead timeslice value has no effect. System overhead runs whenever a periodic task is not running.



Placing and Configuring the Drive

Using This Chapter

For information about:	See page
Understanding the Interface to the Drive	3-1
Determining When the Controller Updates the Drive	3-3
Placing and Configuring the Drive	3-4
Inhibiting the Drive Connection	3-13
Using DriveExecutive Lite	3-15
Accessing Drive Data	3-23
Monitoring Drive Data	3-23

Understanding the Interface to the Drive

The DriveLogix controller supports a direct connection to the drive consisting of 16 inputs and 16 outputs. The tag names and data types associated with the inputs and outputs are determined by the communication format selection. Currently, the following three communications formats are available:

- Velocity Control – for typical speed regulated applications
- Position Control – for typical positioning applications
- User-Defined Control – for general use as required.
- Motion Control - for use with Logix motion commands
- Custom User-Defined Control - for general use as required.

Each communication format contains a number of pre-defined tags and user-defined tags.

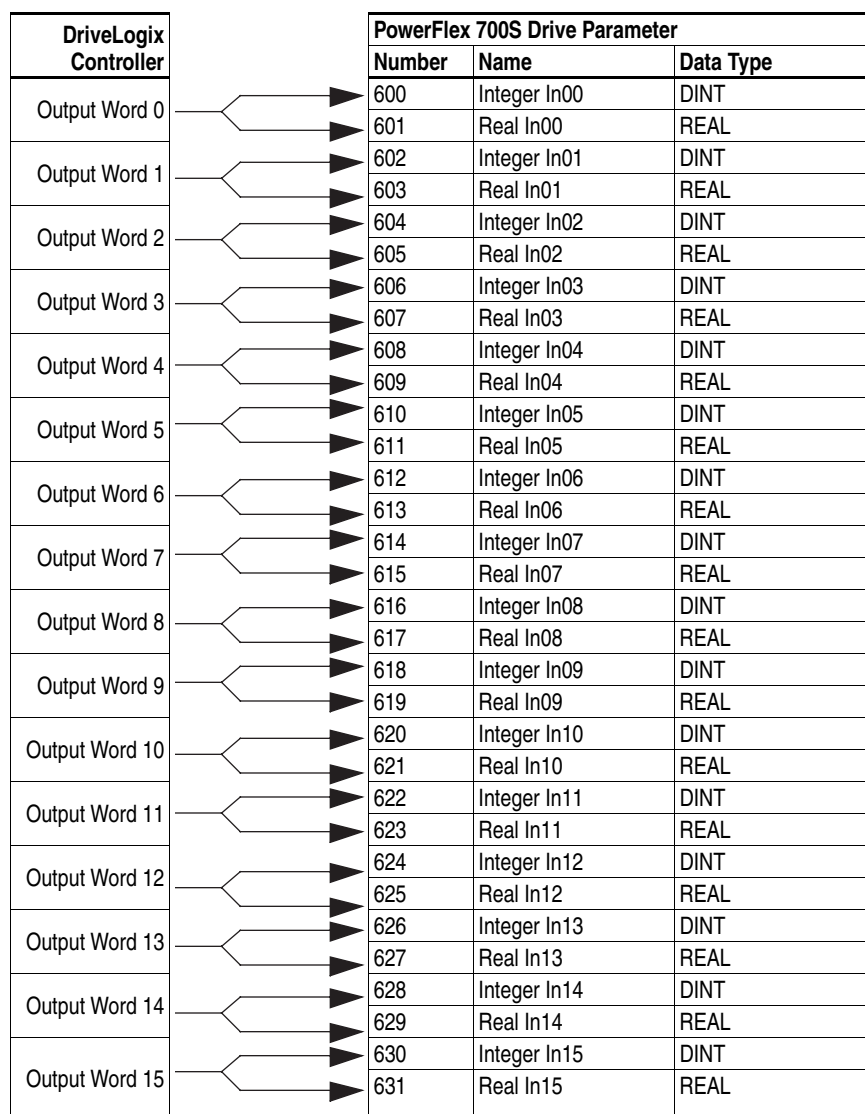
The pre-defined tag names and data types correspond to the associated parameters within the drive necessary to support the selected communications format. Links must be established in the drive to support the pre-defined tags and are configured using DriveExecutive™ software. Linking is a mechanism within the drive that configures data flow within the drive. The links within the drive to support the pre-defined tags are protected and must be present. If the associated links are not present, or are deleted, the communication connection between the controller and drive will be broken.

The user-defined tags are made up of a fixed number of REAL (floating point) and DINT (double integer) data types. No links are required within the drive to support these tags. Therefore, links may be created and deleted as required with no affect on the communication connection between the controller and the drive. The user-defined tags may be used to address application specific data needs not covered by the pre-defined tags.

Mapping for Inputs and Outputs

For each of the 16 inputs or 16 outputs, there are two dedicated parameters within the drive for a total of 64 parameters. One parameter is a DINT type and the other is a REAL type. Selecting a communication format defines the data types for each input and selects the correct parameter for each input and output in the communication link. The remaining parameter is not utilized.

DriveLogix Controller	PowerFlex 700S Drive Parameter		
	Number	Name	Data Type
Input Word 0	632	Integer Out00	DINT
	633	Real Out00	REAL
Input Word 1	634	Integer Out01	DINT
	635	Real Out01	REAL
Input Word 2	636	Integer Out02	DINT
	637	Real Out02	REAL
Input Word 3	638	Integer Out03	DINT
	639	Real Out03	REAL
Input Word 4	640	Integer Out04	DINT
	641	Real Out04	REAL
Input Word 5	642	Integer Out05	DINT
	643	Real Out05	REAL
Input Word 6	644	Integer Out06	DINT
	645	Real Out06	REAL
Input Word 7	646	Integer Out07	DINT
	647	Real Out07	REAL
Input Word 8	648	Integer Out08	DINT
	649	Real Out08	REAL
Input Word 9	650	Integer Out09	DINT
	651	Real Out09	REAL
Input Word 10	652	Integer Out10	DINT
	653	Real Out10	REAL
Input Word 11	654	Integer Out11	DINT
	655	Real Out11	REAL
Input Word 12	656	Integer Out12	DINT
	657	Real Out12	REAL
Input Word 13	658	Integer Out13	DINT
	659	Real Out13	REAL
Input Word 14	660	Integer Out14	DINT
	661	Real Out14	REAL
Input Word 15	662	Integer Out15	DINT
	663	Real Out15	REAL



Determining When the Controller Updates the Drive

The DriveLogix controller follows a producer/consumer model for the drive connection, similar to the interface to an I/O module. The drive acts as both an input module, producing data for the controller; and an output module, consuming data from the controller. Although the producer/consumer model multi casts data, all data in the drive connection is exclusive to the DriveLogix controller.

The controller updates the input and output data in the drive connection asynchronously to the logic scan, consistent with the way it handles other I/O data. All input data from the drive is read in a single block and all output data is written to the drive in a single block.

You must configure the Requested Packet Interval (RPI) rate for the drive. This setting affects how fast the controller reads and writes the data in the drive interface.

TIP

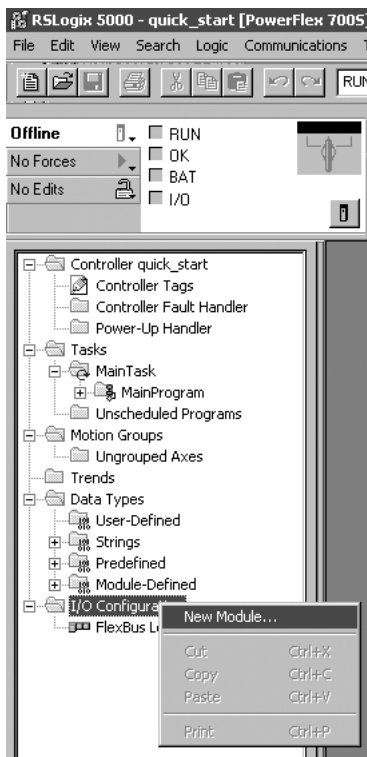
If you want data to remain constant throughout one scan, make a copy of the data at the beginning of the scan and use the copy throughout the scan.

The Drive consumes data from the DriveLogix controller every 2 milliseconds, and produces data to the controller every 2 milliseconds. The drive updates the inputs and outputs to the controller asynchronous to both the program scan and I/O scan of the controller.

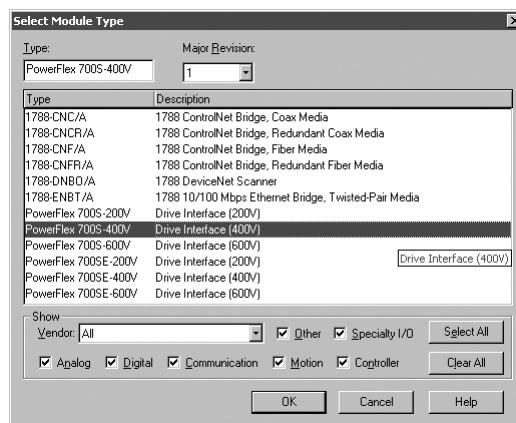
Placing and Configuring the Drive

When you create a project for the DriveLogix controller in RSLogix 5000, the Controller Organizer automatically displays the local DIN rail for Flex I/O. You must add the PowerFlex 700S drive to the configuration, in a manner similar to adding an I/O module. The Controller Organizer automatically places the drive in slot two.

1. In the Controller Organizer, select the I/O Configuration folder. Right-click the selected folder and select New Module..



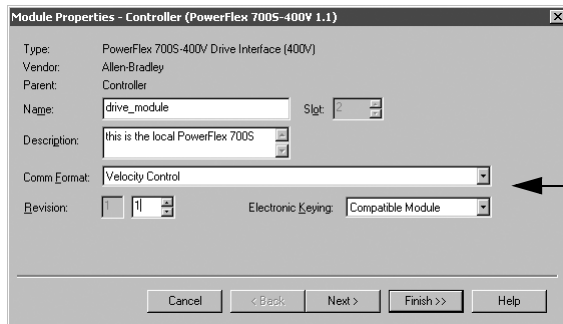
2. Select the drive (PowerFlex 700S-400V in this example).



IMPORTANT

You must select the correct voltage rating for the drive, when adding the drive. You can find this on the drive data nameplate.

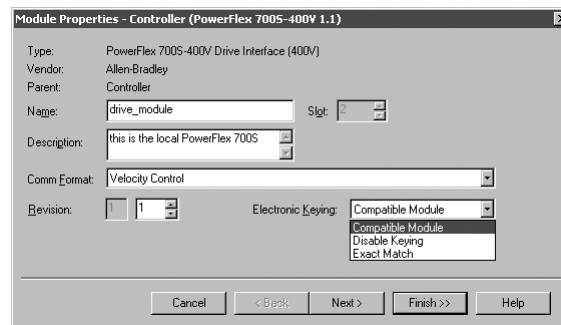
3. Configure the drive. Use the module properties wizard to specify characteristics for the module. Click Next
4. Click finish when you are done. The completed module appears in the Controller Organizer.



The selection you make for the Comm Format determines the communication format for the connection to the drive. This determines the tag names and data types. See page 3-6. Once you complete adding a module, you cannot change this selection.

Electronic Keying

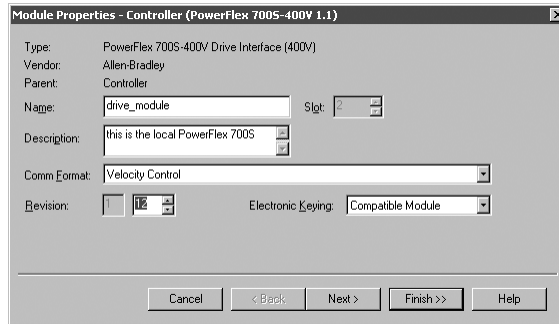
Electronic keying has no effect on drive module. However, the default setting (Compatible Module) is recommended.



Selecting “Compatible Module” allows you to enter the drive firmware minor revision.

Revision

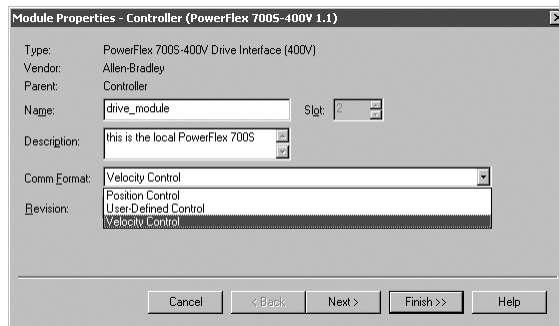
You must enter the correct drive VPL firmware revision, in order to launch DriveExecutive Lite and create the appropriate links for the selected communication format. Determine the firmware revision by viewing parameter 314 [VPL Firmware Rev] in the drive.



Communication Formats

The communication format determines the data structure, tag names, and required links for communication to the drive. Each communication format has been structured to meet the requirements of a specific type of application (Speed Control, Position Control, or general purpose), and supports a different data structure. The links within the PowerFlex 700S required to support the selected format are also different. Any of the available communication formats create one direct connection to the drive.

You select the communication format when you configure the drive module.



The default communication format for the drive is Velocity Control. The tags are created as controller-scoped tags. The following tag structure shows the Velocity Control format. The tag structure for this example’s drive connection has the tag name of “drive_module”.

Controller Tags - quick_start(controller)							
Scope	Tag Name	Alias For	Base Tag	Type	Style	Description	
+	drive_module:I			AB:DRIVE_VELOCITYCONTROL:I:0			
+	drive_module:O			AB:DRIVE_VELOCITYCONTROL:O:0			
+	Local:I			AB:1794_AVB_8SLOT:I:0			
+	Local:O			AB:1794_AVB_8SLOT:O:0			
*							

The following tables show the tag names and their relationship to parameters in the drive. These examples use a module name of “drive_module”.

Table 3.1 Mapping for the Velocity Control Communication Format

DriveLogix Controller Outputs		PowerFlex 700S Inputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:O.LogicCommand	DINT	600	Integer In00	Logic Command	151
drive_module:O.SpeedRef1	REAL	603	Real In01	Speed Ref 1	10
drive_module:O.TorqueRef1	DINT	605	Real In02	Torque Ref 1	111
drive_module:O.SpdTorqModeSel	DINT	606	Integer In03	Spd/Trq ModeSel	110
drive_module:O.TorqueStep	REAL	609	Real In04	Torque Step	116
drive_module:O.SpdRegDroop	REAL	611	Real In05	Spd Reg Droop	86
drive_module:O.UserDefinedRealData[0]	REAL	613	Real In06		
drive_module:O.UserDefinedRealData[1]	REAL	615	Real In07		
drive_module:O.UserDefinedRealData[2]	REAL	617	Real In08		
drive_module:O.UserDefinedRealData[3]	REAL	619	Real In09		
drive_module:O.UserDefinedRealData[4]	REAL	621	Real In10		
drive_module:O.UserDefinedRealData[5]	REAL	623	Real In11		
drive_module:O.UserDefinedRealData[6]	REAL	625	Real In12		
drive_module:O.UserDefinedIntegerData[0]	DINT	626	Integer In13		
drive_module:O.UserDefinedIntegerData[1]	DINT	628	Integer In14		
drive_module:O.UserDefinedIntegerData[2]	DINT	630	Integer In15		

DriveLogix Controller Inputs		PowerFlex 700S Outputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:I.LogicStatus	DINT	632	Integer Out00	Logic Status	155
drive_module:I.FilteredSpdFdbk	REAL	635	Real Out01	Filtered SpdFdbk	71
drive_module:I.MotorTorqueRef	REAL	637	Real Out02	Motor Torque Ref	303
drive_module:I.OutputCurrent	REAL	639	Real Out03	Output Current	308
drive_module:I.MCStatus	DINT	640	Integer Out04	MC Status	555
drive_module:I.LocalIOStatus	DINT	642	Integer Out05	Local I/O Status	824
drive_module:I.UserDefinedRealData[0]	REAL	645	Real Out06		
drive_module:I.UserDefinedRealData[1]	REAL	647	Real Out07		
drive_module:I.UserDefinedRealData[2]	REAL	649	Real Out08		
drive_module:I.UserDefinedRealData[3]	REAL	651	Real Out09		
drive_module:I.UserDefinedRealData[4]	REAL	653	Real Out10		
drive_module:I.UserDefinedRealData[5]	REAL	655	Real Out11		
drive_module:I.UserDefinedRealData[6]	REAL	657	Real Out12		
drive_module:I.UserDefinedIntegerData[0]	DINT	658	Integer Out13		
drive_module:I.UserDefinedIntegerData[1]	DINT	660	Integer Out14		
drive_module:I.UserDefinedIntegerData[2]	DINT	662	Integer Out15		

Table 3.2 Mapping for the Position Control Communication Format

DriveLogix Controller Outputs		PowerFlex 700S Inputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:O.LogicCommand	DINT	600	Integer In00	Logic Command	151
drive_module:O.SpeedRef1	REAL	603	Real In01	Speed Ref 1	10
drive_module:O.TorqueRef1	REAL	605	Real In02	Torque Ref 1	111
drive_module:O.SpdTrqModeSel	DINT	606	Integer In03	Spd/Trq ModeSel	110
drive_module:O.TorqueStep	REAL	609	Real In04	Torque Step	116
drive_module:O.SpdRegDroop	REAL	611	Real In05	Spd Reg Droop	86
drive_module:O.PositionControl	DINT	612	Integer In06	Position Control	740
drive_module:O.CoarsePositTrgt	DINT	614	Integer In07	CoarsePosit Trgt	748
drive_module:O.PtPtPositRef	DINT	616	Integer In08	Pt-Pt Posit Ref	758
drive_module:O.PositRefSel	DINT	618	Integer In09	Posit Ref Sel	742
drive_module:O.PositOffset1	DINT	620	Integer In10	Posit Offset 1	753
drive_module:O.UserDefinedRealData[0]	REAL	623	Real In11		
drive_module:O.UserDefinedRealData[1]	REAL	625	Real In12		
drive_module:O.UserDefinedRealData[2]	REAL	627	Real In13		
drive_module:O.UserDefinedIntegerData[0]	DINT	628	Integer In14		
drive_module:O.UserDefinedIntegerData[1]	DINT	630	Integer In15		

DriveLogix Controller Inputs		PowerFlex 700S Outputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:I.LogicStatus	DINT	632	Integer Out00	Logic Status	155
drive_module:I.FilteredSpdFdbk	REAL	635	Real Out01	Filtered SpdFdbk	71
drive_module:I.MotorTorqueRef	REAL	637	Real Out02	Motor Torque Ref	303
drive_module:I.OutputCurrent	REAL	639	Real Out03	Output Current	308
drive_module:I.MCStatus	DINT	640	Integer Out04	MC Status	555
drive_module:I.LocalIOStatus	DINT	642	Integer Out05	Local I/O Status	824
drive_module:I.MtrPositFdbk	DINT	644	Integer Out06	Mtr Posit Fdbk	762
drive_module:I.ActMotorPosit	DINT	646	Integer Out07	Act Motor Posit	763
drive_module:I.PositionStatus	DINT	648	Integer Out08	Position Status	741
drive_module:I.PositionError	DINT	650	Integer Out09	Position Error	769
drive_module:I.UserDefinedRealData[0]	REAL	653	Real Out10		
drive_module:I.UserDefinedRealData[1]	REAL	655	Real Out11		
drive_module:I.UserDefinedRealData[2]	REAL	657	Real Out12		
drive_module:I.UserDefinedRealData[3]	REAL	659	Real Out13		
drive_module:I.UserDefinedIntegerData[0]	DINT	660	Integer Out14		
drive_module:I.UserDefinedIntegerData[1]	DINT	662	Integer Out15		

Table 3.3 Mapping for the User-Defined Control Communication Format

DriveLogix Controller Outputs		PowerFlex 700S Inputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:O.LogicCommand	DINT	600	Integer In00	Logic Command	151
drive_module:O.UserDefinedRealData[0]	REAL	603	Real In01		
drive_module:O.UserDefinedRealData[1]	REAL	605	Real In02		
drive_module:O.UserDefinedRealData[2]	REAL	607	Real In03		
drive_module:O.UserDefinedRealData[3]	REAL	609	Real In04		
drive_module:O.UserDefinedRealData[4]	REAL	611	Real In05		
drive_module:O.UserDefinedRealData[5]	REAL	613	Real In06		
drive_module:O.UserDefinedRealData[6]	REAL	615	Real In07		
drive_module:O.UserDefinedRealData[7]	REAL	617	Real In08		
drive_module:O.UserDefinedRealData[8]	REAL	619	Real In09		
drive_module:O.UserDefinedRealData[9]	REAL	621	Real In10		
drive_module:O.UserDefinedRealData[10]	REAL	623	Real In11		
drive_module:O.UserDefinedRealData[11]	REAL	625	Real In12		
drive_module:O.UserDefinedIntegerData[0]	DINT	626	Integer In13		
drive_module:O.UserDefinedIntegerData[1]	DINT	628	Integer In14		
drive_module:O.UserDefinedIntegerData[2]	DINT	630	Integer In15		

DriveLogix Controller Inputs		PowerFlex 700S Outputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:I.LogicStatus	DINT	632	Integer Out00	Logic Status	155
drive_module:I.UserDefinedRealData[0]	REAL	635	Real Out01		
drive_module:I.UserDefinedRealData[1]	REAL	637	Real Out02		
drive_module:I.UserDefinedRealData[2]	REAL	639	Real Out03		
drive_module:I.UserDefinedRealData[3]	REAL	641	Real Out04		
drive_module:I.UserDefinedRealData[4]	REAL	643	Real Out05		
drive_module:I.UserDefinedRealData[5]	REAL	645	Real Out06		
drive_module:I.UserDefinedRealData[6]	REAL	647	Real Out07		
drive_module:I.UserDefinedRealData[7]	REAL	649	Real Out08		
drive_module:I.UserDefinedRealData[8]	REAL	651	Real Out09		
drive_module:I.UserDefinedRealData[9]	REAL	653	Real Out10		
drive_module:I.UserDefinedRealData[10]	REAL	655	Real Out11		
drive_module:I.UserDefinedRealData[11]	REAL	657	Real Out12		
drive_module:I.UserDefinedIntegerData[0]	DINT	658	Integer Out13		
drive_module:I.UserDefinedIntegerData[1]	DINT	660	Integer Out14		
drive_module:I.UserDefinedIntegerData[2]	DINT	662	Integer Out15		

Table 3.4 Mapping for the Motion Control Communication Format

DriveLogix Controller Outputs		PowerFlex 700S Inputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:O.UserDefinedRealData[0]	REAL	601	Real In00		
drive_module:O.UserDefinedRealData[1]	REAL	603	Real In01		
drive_module:O.UserDefinedRealData[2]	REAL	605	Real In02		
drive_module:O.UserDefinedRealData[3]	REAL	606	Real In03		
drive_module:O.UserDefinedRealData[4]	REAL	609	Real In04		
drive_module:O.UserDefinedRealData[5]	REAL	611	Real In05		
drive_module:O.UserDefinedRealData[6]	REAL	613	Real In06		
drive_module:O.UserDefinedRealData[7]	REAL	615	Real In07		
drive_module:O.UserDefinedRealData[8]	REAL	617	Real In08		
drive_module:O.UserDefinedRealData[9]	REAL	619	Real In09		
drive_module:O.UserDefinedRealData[10]	REAL	621	Real In10		
drive_module:O.UserDefinedRealData[11]	REAL	623	Real In11		
drive_module:O.UserDefinedIntegerData[0]	DINT	624	Integer In12		
drive_module:O.UserDefinedIntegerData[1]	DINT	626	Integer In13		
drive_module:O.UserDefinedIntegerData[2]	DINT	628	Integer In14		
drive_module:O.UserDefinedIntegerData[2]	DINT	630	Integer In15		

DriveLogix Controller Inputs		PowerFlex 700S Outputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:I.LogicStatus	DINT	632	Integer Out00	Logic Status	155
drive_module:I.UserDefinedRealData[0]	REAL	635	Real Out01		
drive_module:I.UserDefinedRealData[1]	REAL	637	Real Out02		
drive_module:I.UserDefinedRealData[2]	REAL	639	Real Out03		
drive_module:I.UserDefinedRealData[3]	REAL	641	Real Out04		
drive_module:I.UserDefinedRealData[4]	REAL	643	Real Out05		
drive_module:I.UserDefinedRealData[5]	REAL	645	Real Out06		
drive_module:I.UserDefinedRealData[6]	REAL	647	Real Out07		
drive_module:I.UserDefinedRealData[7]	REAL	649	Real Out08		
drive_module:I.UserDefinedRealData[8]	REAL	651	Real Out09		
drive_module:I.UserDefinedRealData[9]	REAL	653	Real Out10		
drive_module:I.UserDefinedRealData[10]	REAL	655	Real Out11		
drive_module:I.UserDefinedRealData[11]	REAL	657	Real Out12		
drive_module:I.UserDefinedIntegerData[0]	DINT	658	Integer Out13		
drive_module:I.UserDefinedIntegerData[1]	DINT	660	Integer Out14		
drive_module:I.UserDefinedIntegerData[2]	DINT	662	Integer Out15		

Table 3.5 Mapping for the Custom User-Defined Control Communication Format

DriveLogix Controller Outputs		PowerFlex 700S Inputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:O.LogicCommand	DINT	600	Integer In00	Logic Command	151
drive_module:O.UserDefinedRealData[0]	REAL	603	Real In01		
drive_module:O.UserDefinedRealData[1]	REAL	605	Real In02		
drive_module:O.UserDefinedRealData[2]	REAL	607	Real In03		
drive_module:O.UserDefinedRealData[3]	REAL	609	Real In04		
drive_module:O.UserDefinedRealData[4]	REAL	611	Real In05		
drive_module:O.UserDefinedRealData[5]	REAL	613	Real In06		
drive_module:O.UserDefinedRealData[6]	REAL	615	Real In07		
drive_module:O.UserDefinedRealData[7]	REAL	617	Real In08		
drive_module:O.UserDefinedIntegerData[0]	DINT	618	Integer In09		
drive_module:O.UserDefinedIntegerData[1]	DINT	620	Integer In10		
drive_module:O.UserDefinedIntegerData[2]	DINT	622	Integer In11		
drive_module:O.UserDefinedIntegerData[3]	DINT	624	Integer In12		
drive_module:O.UserDefinedIntegerData[4]	DINT	626	Integer In13		
drive_module:O.UserDefinedIntegerData[5]	DINT	628	Integer In14		
drive_module:O.UserDefinedIntegerData[6]	DINT	630	Integer In15		

DriveLogix Controller Inputs		PowerFlex 700S Outputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:I.LogicStatus	DINT	632	Integer Out00	Logic Status	155
drive_module:I.UserDefinedRealData[0]	REAL	635	Real Out01		
drive_module:I.UserDefinedRealData[1]	REAL	637	Real Out02		
drive_module:I.UserDefinedRealData[2]	REAL	639	Real Out03		
drive_module:I.UserDefinedRealData[3]	REAL	641	Real Out04		
drive_module:I.UserDefinedRealData[4]	REAL	643	Real Out05		
drive_module:I.UserDefinedRealData[5]	REAL	645	Real Out06		
drive_module:I.UserDefinedRealData[6]	REAL	647	Real Out07		
drive_module:I.UserDefinedRealData[7]	REAL	649	Real Out08		
drive_module:I.UserDefinedIntegerData[0]	DINT	650	Integer Out09		
drive_module:I.UserDefinedIntegerData[1]	DINT	652	Integer Out10		
drive_module:I.UserDefinedIntegerData[2]	DINT	654	Integer Out11		
drive_module:I.UserDefinedIntegerData[3]	DINT	656	Integer Out12		
drive_module:I.UserDefinedIntegerData[4]	DINT	658	Integer Out13		
drive_module:I.UserDefinedIntegerData[5]	DINT	660	Integer Out14		
drive_module:I.UserDefinedIntegerData[6]	DINT	662	Integer Out15		

For each of the communication formats, drive_module:O.LogicCommand and drive_module:I.LogicStatus are provided as DINT data types. In addition to these tags, the control bits for each are also available as Boolean values with tag names that correspond to the control bits in the drive. This gives you the option of programming the Logic Command and Status words at the Boolean level or as an integer value.

Not all 32-bits within parameter 151 [Logic Command], are directly visible in the PowerFlex 700S. To view all 32-bits, refer to parameter 152 [Applied LogicCmd].

DriveLogix Controller Outputs		PowerFlex 700S Inputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:O.LogicCommand	DINT	600	Integer In00	Applied LogicCmd	152
drive_module:O.SpdRampDsbl	BOOL	bit 0	Integer In00	SpdRamp Dsbl	bit 0
drive_module:O.SpdScrvEn	BOOL	bit 1	Integer In00	Spd S Crv En	bit 1
drive_module:O.TachLossRst	BOOL	bit 2	Integer In00	TachLoss Rst	bit 2
drive_module:O.TimeAxisEn	BOOL	bit 3	Integer In00	Time Axis En	bit 3
drive_module:O.MCAtuneEn	BOOL	bit 4	Integer In00	MC Atune En	bit 4
drive_module:O.DirCtrlEn	BOOL	bit 6	Integer In00	Dir Sel En	bit 6
drive_module:O.PMOffsetEn	BOOL	bit 7	Integer In00	PM Offset En	bit 7
drive_module:O.MtrInertEn	BOOL	bit 8	Integer In00	Mtr Inert En	bit 8
drive_module:O.SysInertEn	BOOL	bit 9	Integer In00	Sys Inert En	bit 9
drive_module:O.InertiaComp	BOOL	bit 10	Integer In00	Inertia Comp	bit 10
drive_module:O.FrictComp	BOOL	bit 11	Integer In00	Frict Comp	bit 11
drive_module:O.ProcsTrimEn	BOOL	bit 12	Integer In00	ProcsTrim En	bit 12
drive_module:O.PositionEnbl	BOOL	bit 13	Integer In00	PositionEnbl	bit 13
drive_module:O.NormalStop	BOOL	bit 16	Integer In00	Normal Stop	bit 16
drive_module:O.Start	BOOL	bit 17	Integer In00	Start	bit 17
drive_module:O.Jog1	BOOL	bit 18	Integer In00	Jog 1	bit 18
drive_module:O.ClearFault	BOOL	bit 19	Integer In00	Clear Fault	bit 19
drive_module:O.UnipolFwd	BOOL	bit 20	Integer In00	Unipol Fwd	bit 20
drive_module:O.UnipolRev	BOOL	bit 21	Integer In00	Unipol Rev	bit 21
drive_module:O.Jog2	BOOL	bit 23	Integer In00	Jog 2	bit 23
drive_module:O.CurrLimStop	BOOL	bit 24	Integer In00	CurrLim Stop	bit 24
drive_module:O.CoastStop	BOOL	bit 25	Integer In00	Coast Stop	bit 25

DriveLogix Controller Outputs		PowerFlex 700S Inputs			
Tag Name	Data Type	Parameter		Linked Parameter	
		Number	Name	Name	Number
drive_module:I.LogicStatus	DINT	600	Integer In00	Logic Status	155
drive_module:I.Enabled	BOOL	bit 0	Integer In00	Enabled	bit 0
drive_module:I.Running	BOOL	bit 1	Integer In00	Running	bit 1
drive_module:I.CommandDir	BOOL	bit 2	Integer In00	Command Dir	bit 2
drive_module:I.ActualDir	BOOL	bit 3	Integer In00	Actual Dir	bit 3
drive_module:I.Accelerating	BOOL	bit 4	Integer In00	Accelerating	bit 4
drive_module:I.Decelerating	BOOL	bit 5	Integer In00	Decelerating	bit 5
drive_module:I.Jogging	BOOL	bit 6	Integer In00	Jogging	bit 6
drive_module:I.Faulted	BOOL	bit 7	Integer In00	Faulted	bit 7
drive_module:I.Alarm	BOOL	bit 8	Integer In00	Alarm	bit 8
drive_module:I.FlashMode	BOOL	bit 9	Integer In00	Flash Mode	bit 9
drive_module:I.RunReady	BOOL	bit 10	Integer In00	Run Ready	bit 10
drive_module:I.AtLimit	BOOL	bit 11	Integer In00	At Limit	bit 11
drive_module:I.TachLossSw	BOOL	bit 12	Integer In00	Tach Loss Sw	bit 12
drive_module:I.AtZeroSpd	BOOL	bit 13	Integer In00	At Zero Spd	bit 13
drive_module:I.AtSetptSpd	BOOL	bit 14	Integer In00	At Setpt Spd	bit 14
drive_module:I.AtSetpt1	BOOL	bit 16	Integer In00	At Setpt 1	bit 16
drive_module:I.AboveSetpt2	BOOL	bit 17	Integer In00	Above Setpt2	bit 17
drive_module:I.MCEnAck	BOOL	bit 28	Integer In00	MC En Ack	bit 28
drive_module:I.MCCommis	BOOL	bit 19	Integer In00	MC Commis	bit 19
drive_module:I.SpdCommis	BOOL	bit 20	Integer In00	Spd Commis	bit 20
drive_module:I.TorqueMode	BOOL	bit 22	Integer In00	Torque Mode	bit 22
drive_module:I.SpeedMode	BOOL	bit 23	Integer In00	Speed Mode	bit 23
drive_module:I.PositionMode	BOOL	bit 24	Integer In00	PositionMode	bit 24
drive_module:I.StartActive	BOOL	bit 25	Integer In00	Start Active	bit 25
drive_module:I.CommandRun	BOOL	bit 26	Integer In00	Command Run	bit 26

Inhibiting the Drive Connection

RSLogix 5000 programming software allows you to inhibit the controller's connection to the drive, in the same way you inhibit its connection to an I/O module. Inhibiting the drive module shuts down the connection from the controller to the drive. When you create the module you can choose to inhibit it. After you have created the module you can inhibit or un-inhibit it by manipulating its properties window.

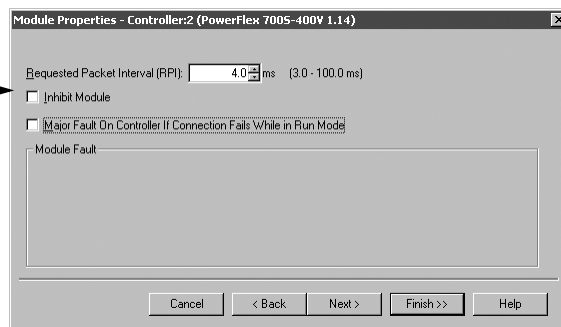
ATTENTION




Inhibiting a drive module breaks the controller's connection to the drive. In this situation, the controller can neither start/stop the drive nor read the status of the drive. The drive can continue to operate based on its parameter settings and inputs. To avoid potential personal injury and damage to machinery, make sure this does not create unsafe operation.

On the Connection tab during creation or in the Properties window

Check the inhibit box to
inhibit the connection to
the drive



When you inhibit the drive module, the Controller Organizer displays a yellow attention symbol  over the module.

If you are: **Inhibit the drive module to:**

offline put a place holder for the drive module to indicate that configuration is not yet complete.

The inhibit status is stored in the project. When you download the project, the module is still inhibited.

online stop communication to the drive.

If you inhibit the drive while you are connected to the module, the connection to the module is closed. By default, the PowerFlex 700S drive will fault. The data inputs to the drive will either hold last state, or reset to zero data based on the setting of parameter 385 [Lgx Comm Loss Data].

If you inhibit the drive 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.

If you uninhibit the drive (clear the check box), and no fault condition occurs, a connection is made to the drive.

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:

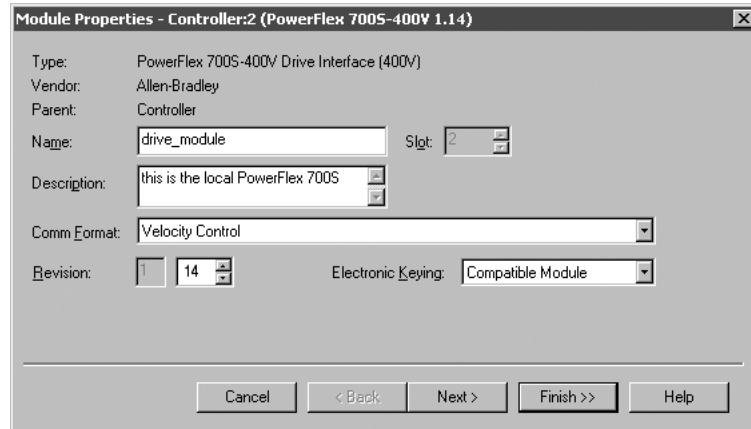
The GSV instruction gets the current status of the drive named "drive_module." The SSV instruction sets the state of "drive_module" as either inhibited or uninhibited.



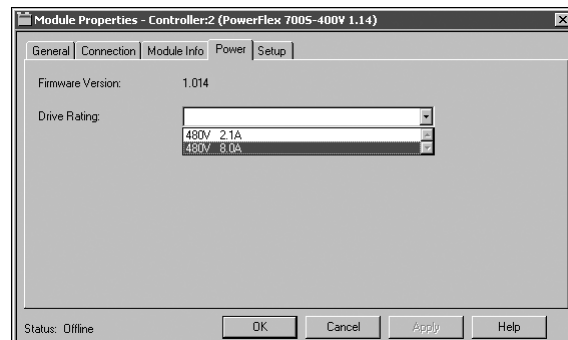
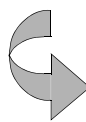
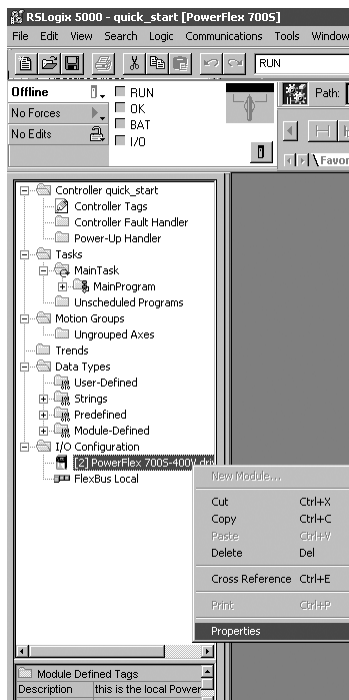
Using DriveExecutive Lite

In order to launch DriveExecutive Lite from within RSLogix 5000, the drives power rating must be selected. The drive firmware revision must be applied prior to selecting the power rating.

1. If not already done, enter the drive firmware revision. Click the Finish button to apply the revision data



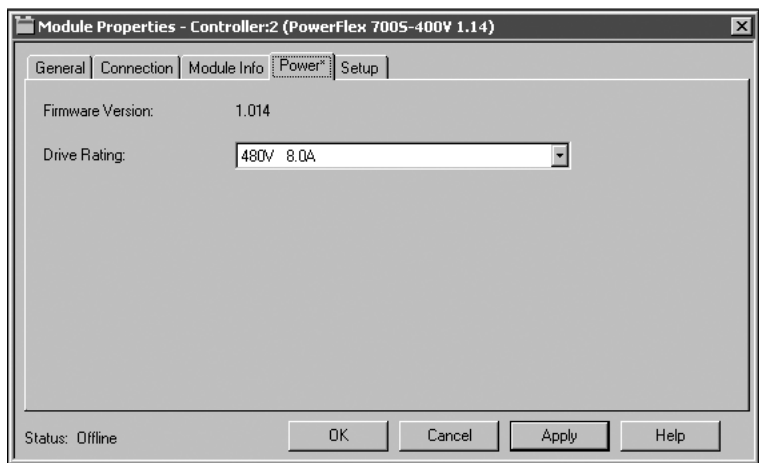
2. In the Controller Organizer, select the PowerFlex 700S drive. Right-click the drive module and select Properties
3. Select the Power tab.
4. Select the correct Drive Rating. This data can be found on the PowerFlex 700S data nameplate.



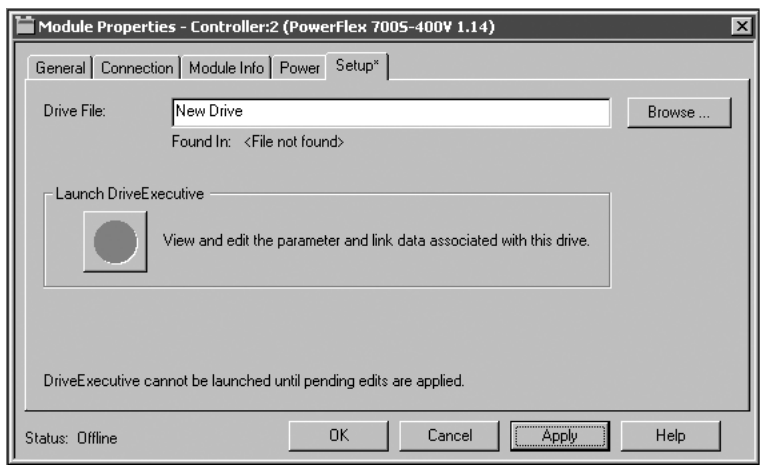
TIP

If your drive's power rating does not appear as a selection, you do not have the DriveExecutive Lite database file for your drive. To create a database file, connect to the drive with DriveExecutive Lite. This will automatically create the database. You can also download the database file from <http://www.ab.com/drives/data.html>

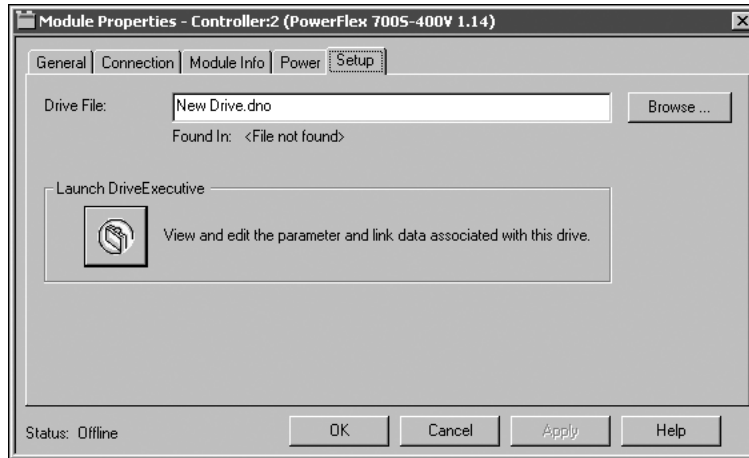
- Once the power rating is selected, apply your changes by selecting the Apply button.



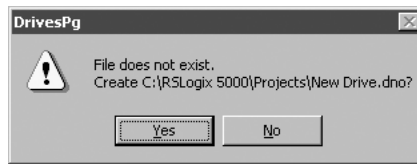
- Select the Setup tab.
- Enter the file name for your DriveExecutive Lite parameter file, then click the Apply button.



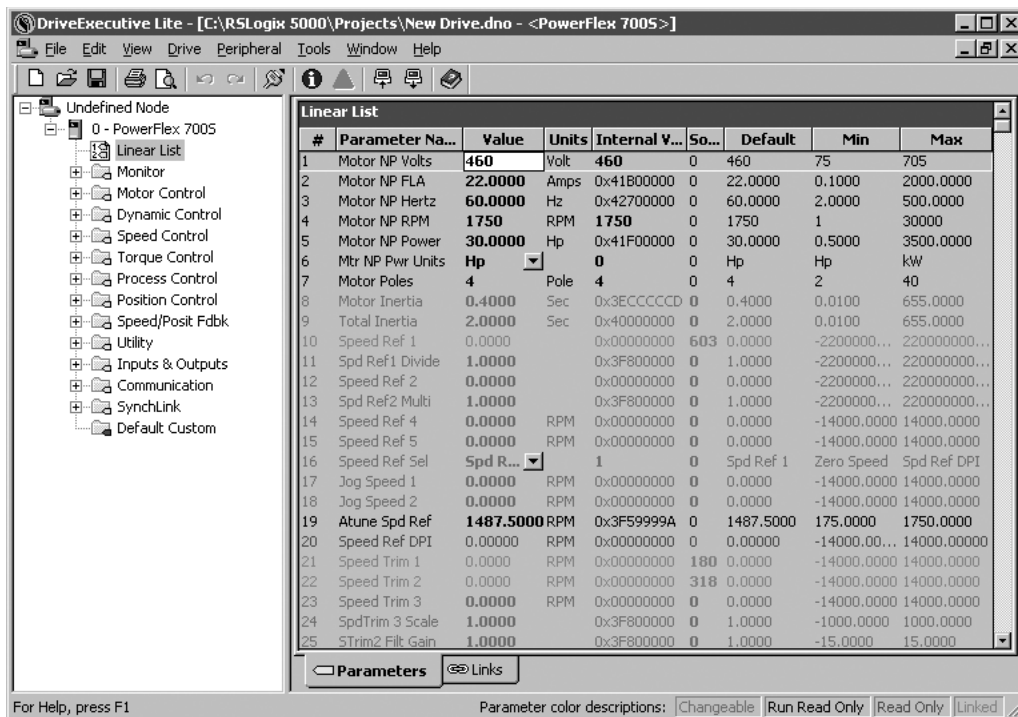
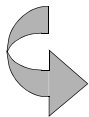
- Click the DriveExecutive button to launch DriveExecutive Lite.



- When asked to create a new DriveExecutive Lite file, select yes.



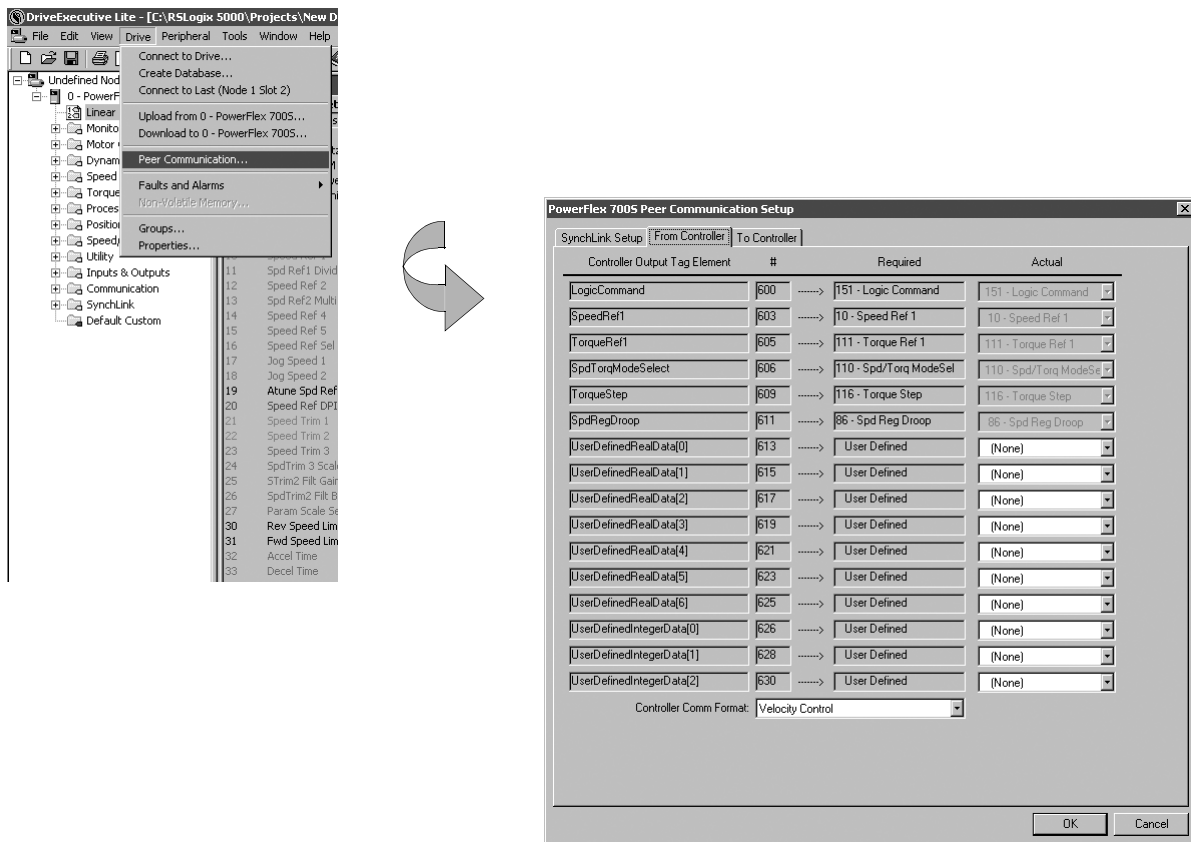
DriveExecutive will then launch and open the newly created file



Viewing the Communication Interface to the Controller

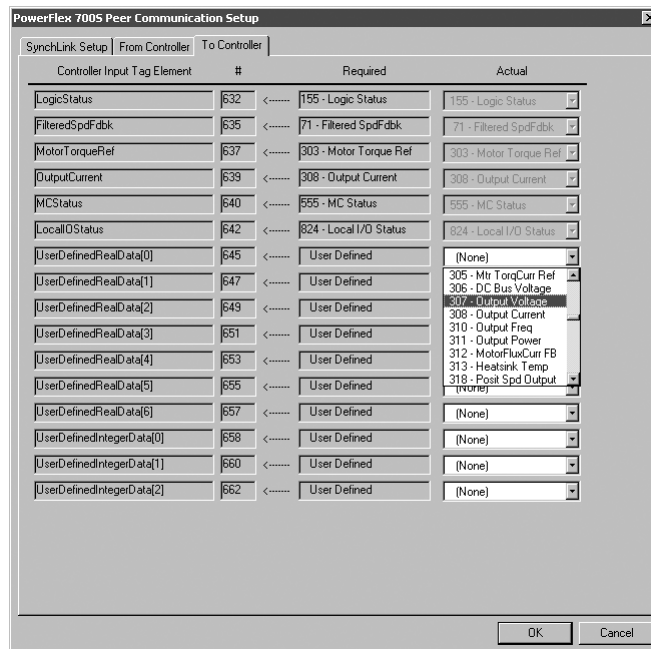
DriveExecutive Lite has a setup screen that details the communication interface between the controller and drive. From this screen, the relationship between drive parameters and controller tags is presented for the selected communication format. You can create additional links within the drive for use with the user-defined tags in the controller.

1. To view the setup screen select Peer Communication from the Drive drop-down menu. Then select the From Controller



Important: In some revisions of DriveExecutive programming software, the “From Controller” and “To Controller” tabs may be labeled “From DriveLogix” and “To DriveLogix”.

- To send additional data from the drive to the controller go to the To Controller tab. Click the To Controller tab. Select the desired source for the user-defined tag (Output Voltage in this example).

**TIP**

Use a UserDefinedRealData tag for parameters that contain floating point data, and use a UserDefinedIntegerData tag for parameters that contain integer data.

Configuring the Drive's Response to a Connection Failure or Controller Mode Change

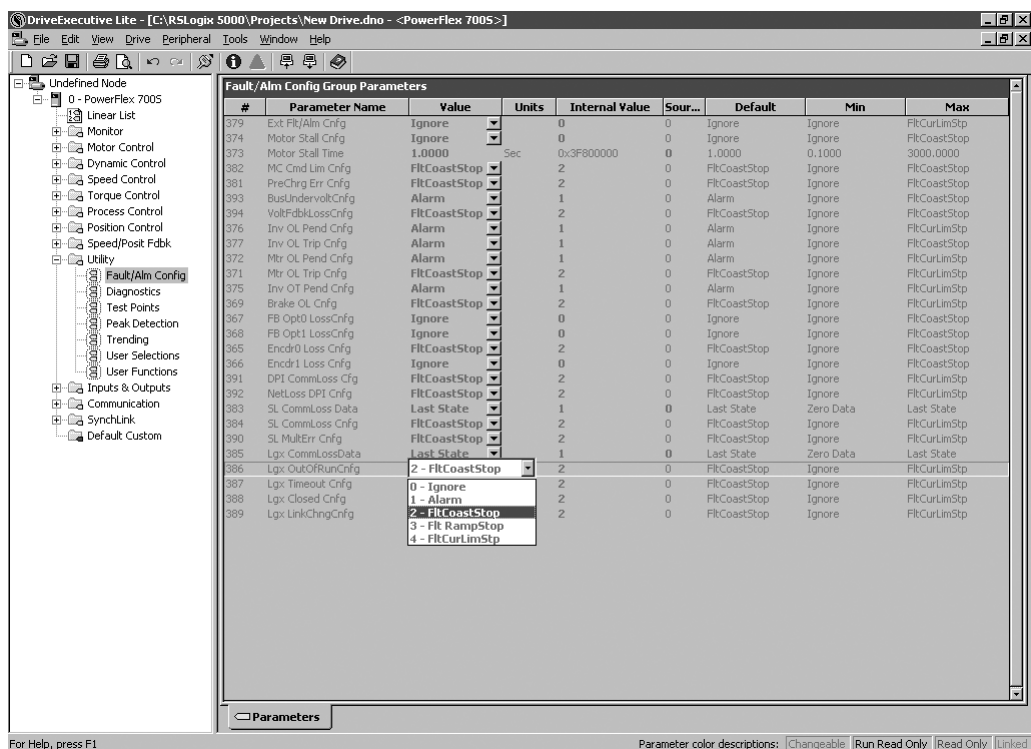
The drive contains several parameters that allow you to configure the drive's response to communication loss to the controller. From the drive's perspective, a communication loss can come in the following two forms.

- The controller closes the connection (for example, the connection is inhibited).
- A general failure occurs causing the connection to time out.

Parameter 386 [Lgx OutofRunCnfg] configures the drive's response to the controller is removed from the Run Mode. Parameter 387 [Lgx Timeout Cnfg] configures the drive's response to a general connection failure as detected by the drive. Parameter 388 [Lgx Closed Cnfg] configures the drive's response to the controller closing the connection. All of these parameters configure the drive's response to these exception events in the following ways: ignore, alarm, fault and coast to stop, fault and ramp to stop, fault and stop in current limit.

Parameter 385 [Lgx CommLossData] determines what the drive does with data from the controller when communication is lost. It determines if the drive resets the data to zero or holds the data in its last state.

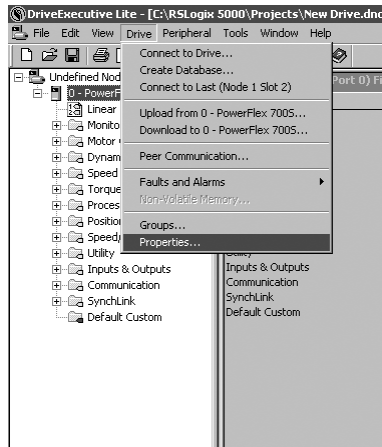
Configure these parameters, using DriveExecutive Lite. Locate them in the Fault/Alm Config group of the Utility file.



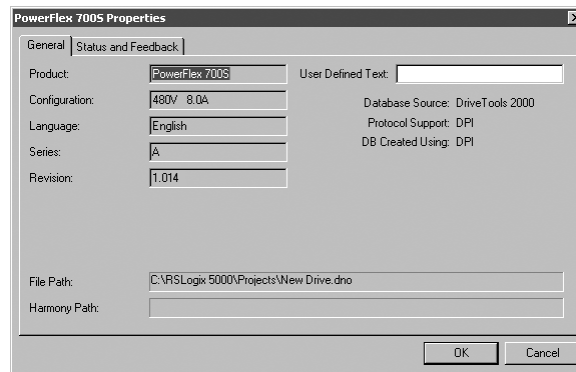
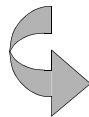
Using Existing DriveExecutive Lite Files

Before using an existing DriveExecutive Lite file, verify the firmware revision, communication format, and power rating in the drive file match the data entered in drive module properties in your DriveLogix application.

1. Select Properties from the Drive menu.

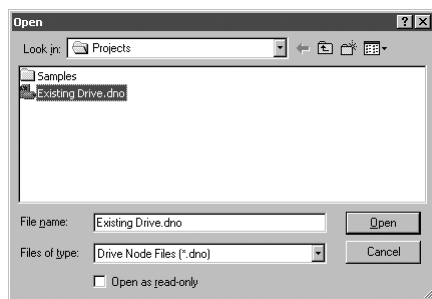
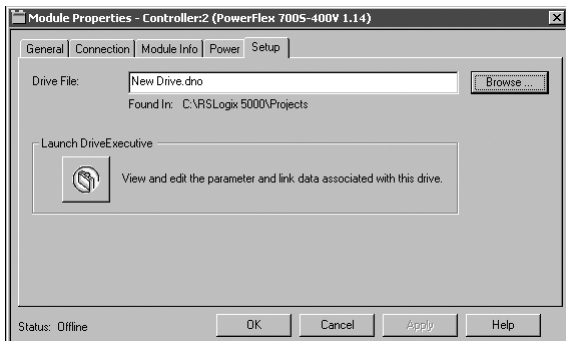


2. View revision and ratings on the General tab of the Properties window.

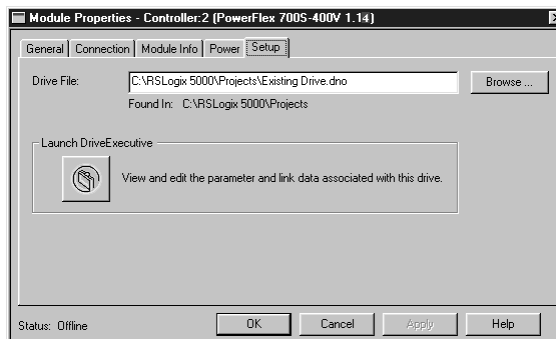
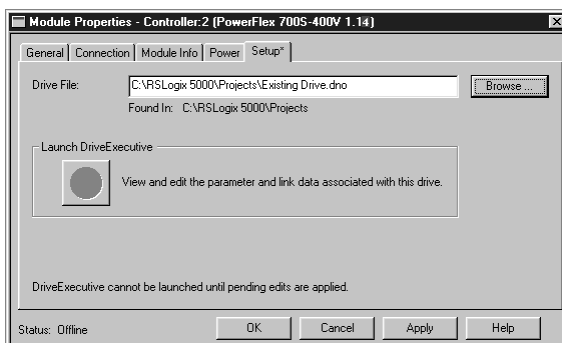
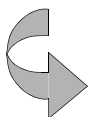


3. Refer to Viewing the Communication Interface to the Controller on page 3-18, to view the communication format.

- 4. In RSLogix 5000, go to the Setup tab of the Properties window. Click the Browse button. Select the existing DriveExecutive file (Existing Drive.dno in this example). Click the Open button.



- 5. Click the Apply button and launch DriveExecutive Lite.



Accessing Drive Data

Drive data is displayed as structures of multiple tags. The names and data structures are based on the selected communication format. The programming software automatically creates the necessary structures and tags when you configure the drive module. Each tag name follows this format:

ModuleName.Type.MemberName.SubMemberName.Bit

where:

This address variable:	Is:
ModuleName	Identifies the module name entered during the drive module configuration
Type	Type of data I = input O = output
MemberName	Specific data from the drive; depends on the selected communication format. For tags associated with pre-defined data links, this name will be the same as the corresponding parameter name in the drive.
SubMemberName	Specific data related to a MemberName
Bit (optional)	Specific bit of a DINT data value

Refer to Communication Formats on page 3-6 for sample tag names.

Monitoring Drive Data

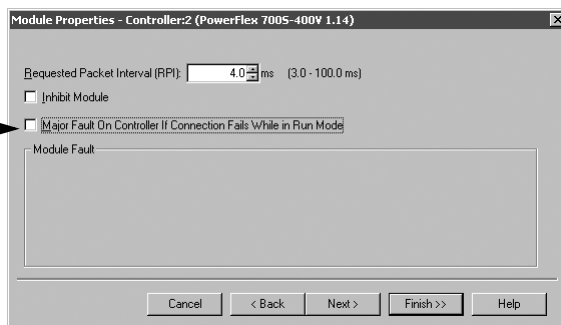
The DriveLogix controller offers different levels at which you can monitor the drive module. You can:

- configure the drive module so that the controller faults if the drive loses its connection to the controller.
- use the programming software to display fault data
- program logic to monitor fault data so you can take appropriate action

Configuring the Controller's Response to a Connection Failure

You can configure the drive module to generate a major fault in the controller if the drive loses its connection to the controller.

Check this box to configure the drive module to generate a major fault if it loses its connection to the controller



If you do not configure the major fault to occur, you should monitor the drive module status. If the drive loses its connection to the controller:

- Outputs remain in their last
- Inputs remain in their last state
- By default the drive will fault

ATTENTION



If a drive loses its connection to the controller, the controller and other I/O modules continue to operate based on old data from the drive, and the drive can continue to operate based on old data from the controller. To avoid potential personal injury and damage to machinery, make sure this does not create unsafe operation.

Configure the drive to generate a controller major fault when the drive loses its connection to the controller. Or, monitor the status of the drive module.

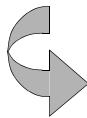
Monitoring the drive module

Each communication format provides a drive status word that will indicate when a drive fault or alarm occurs. To view this data through the programming software:

1. In the Controller Organizer, select Controller Tags. Right-click the selected icon and select Monitor Tags.



2. Expand the data as necessary.



Tag Name	Value
drive_module:1	{...}
drive_module:1.LogicStatus	2#0000_0000_0000_0000_0000_0000_0000_0000
-drive_module:1.Enabled	0
-drive_module:1.Running	0
-drive_module:1.CommandDir	0
-drive_module:1.ActualDir	0
-drive_module:1.Accelerating	0
-drive_module:1.Decelerating	0
-drive_module:1.Jogging	0
-drive_module:1.Faulted	0
-drive_module:1.Alarm	0

You can write logic to monitor these bits and take appropriate action if a fault or alarm occurs. For example, you may want a drive alarm to turn on a warning lamp and a drive fault to sound an alarm and set the motor brake.

EXAMPLE

Given this configuration, the following logic checks the fault and alarm drive status bits.



Local Drive Alarm 1 = Alarm drive_module:1.Alarm] [Unwind Drive Alarm Indicating Lamp Local:0:0.0 <Local:0.Data[0].0> ()
Local Drive Fault 1 = Fault drive_module:1.Faulted] [Unwind Drive Fault Siren Local:0:0.1 <Local:0.Data[0].1> ()
Local Drive Fault 1 = Fault drive_module:1.Faulted] / [Unwind Drive Brake 0 = Brake Set Local:0:0.2 <Local:0.Data[0].2> ()

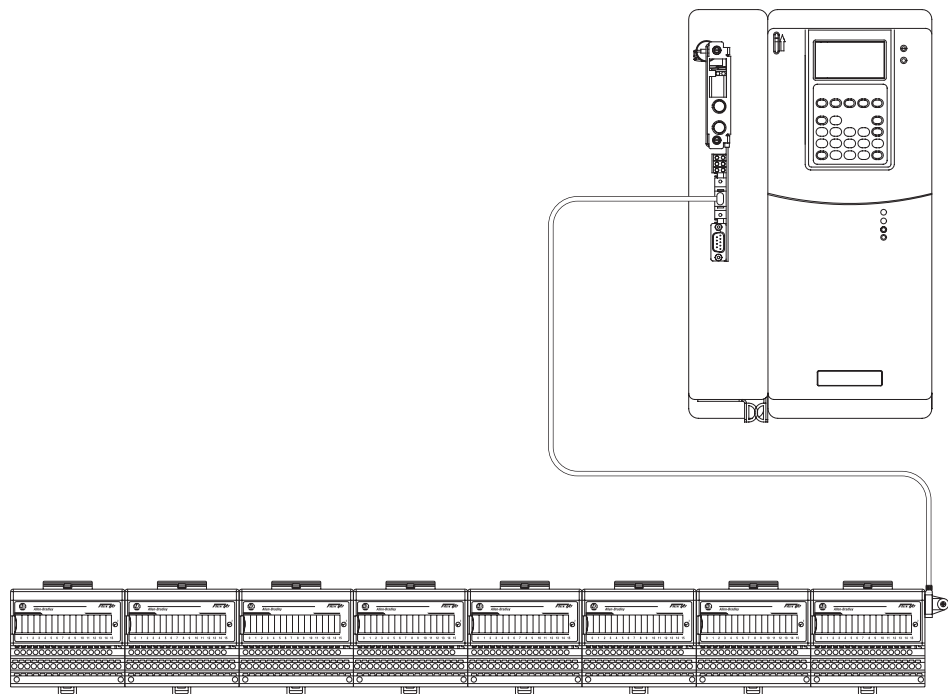
Notes:

Placing and Configuring Local I/O

Using This Chapter

For information about:	See page
Placing local I/O modules	4-2
Determining when the controller updates local I/O	4-3
Configuring a DIN rail	4-5
Configuring local I/O modules	4-6
Inhibiting I/O module operation	4-10
Accessing I/O data	4-13
Monitoring I/O modules	4-16

The DriveLogix controller supports a local DIN rail of as many as 8 I/O modules.



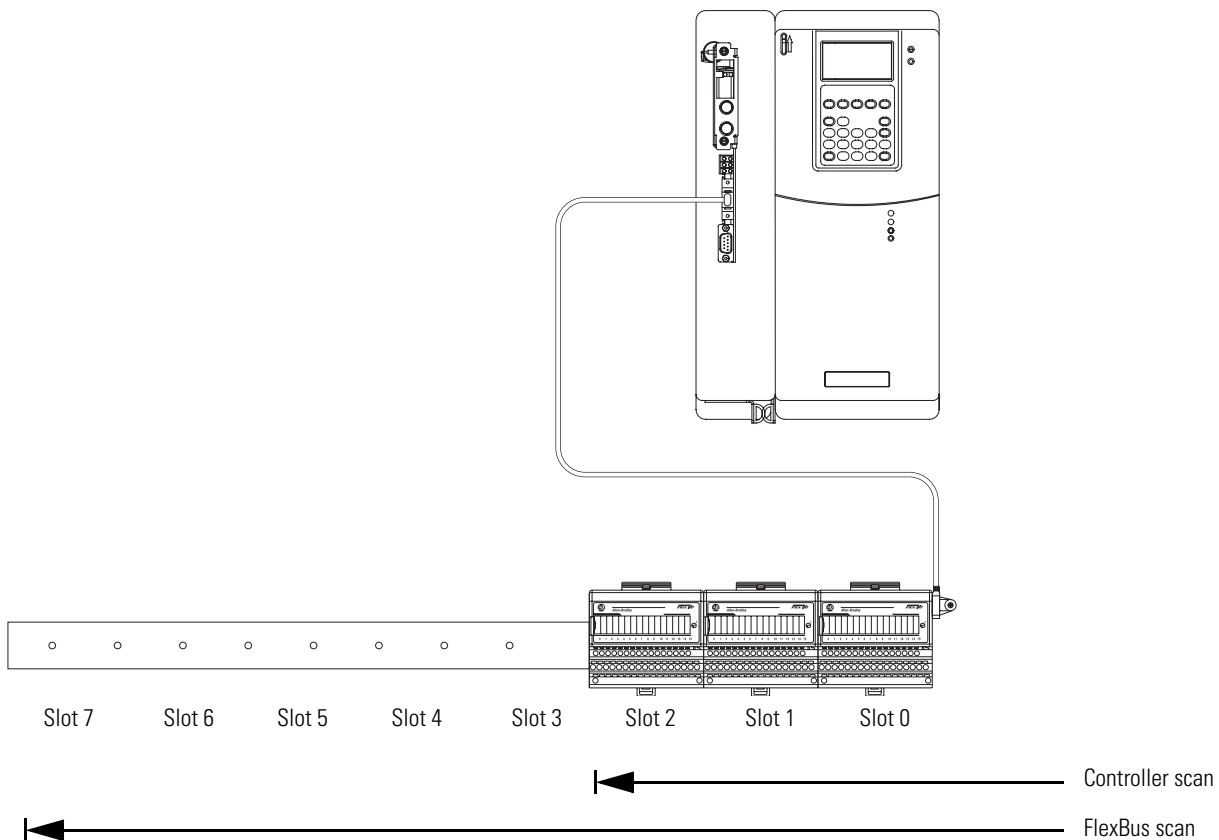
Placing Local I/O Modules

When you create a project for a DriveLogix controller, the Controller Organizer for that project automatically displays the local DIN rail.

You must configure an RPI rate for the DIN rail. This rate applies to all the I/O modules you install on the DIN rail.

If you have:	The fastest possible RPI is:
one rail of digital I/O modules	30 ms
one rail of analog I/O modules	30 ms
one rail of digital and analog I/O modules mixed	30 ms

The DriveLogix controller uses two services to scan I/O: the FlexBus and the controller itself.



IMPORTANT

When installing local I/O with a DriveLogix controller in a high power drive, the length of the FLEXBUS cable must not exceed 3 ft.

The FlexBus continually scans all the slots (0-7) on the DIN rail. The FlexBus scans the DIN rail, starting with slot 0, then scanning slot 1, and continuing with all the slots, and then repeating the cycle. Even if a module is inhibited or a slot is empty, the FlexBus scans that slot. The FlexBus scan identifies where modules reside and collects module data for the controller scan.

The controller scans only those modules that are configured in the Control Organizer. This scan updates the module tags with current data. The RPI for the DIN rail affects how fast the controller gets data from the FlexBus.

Determining When the Controller Updates I/O

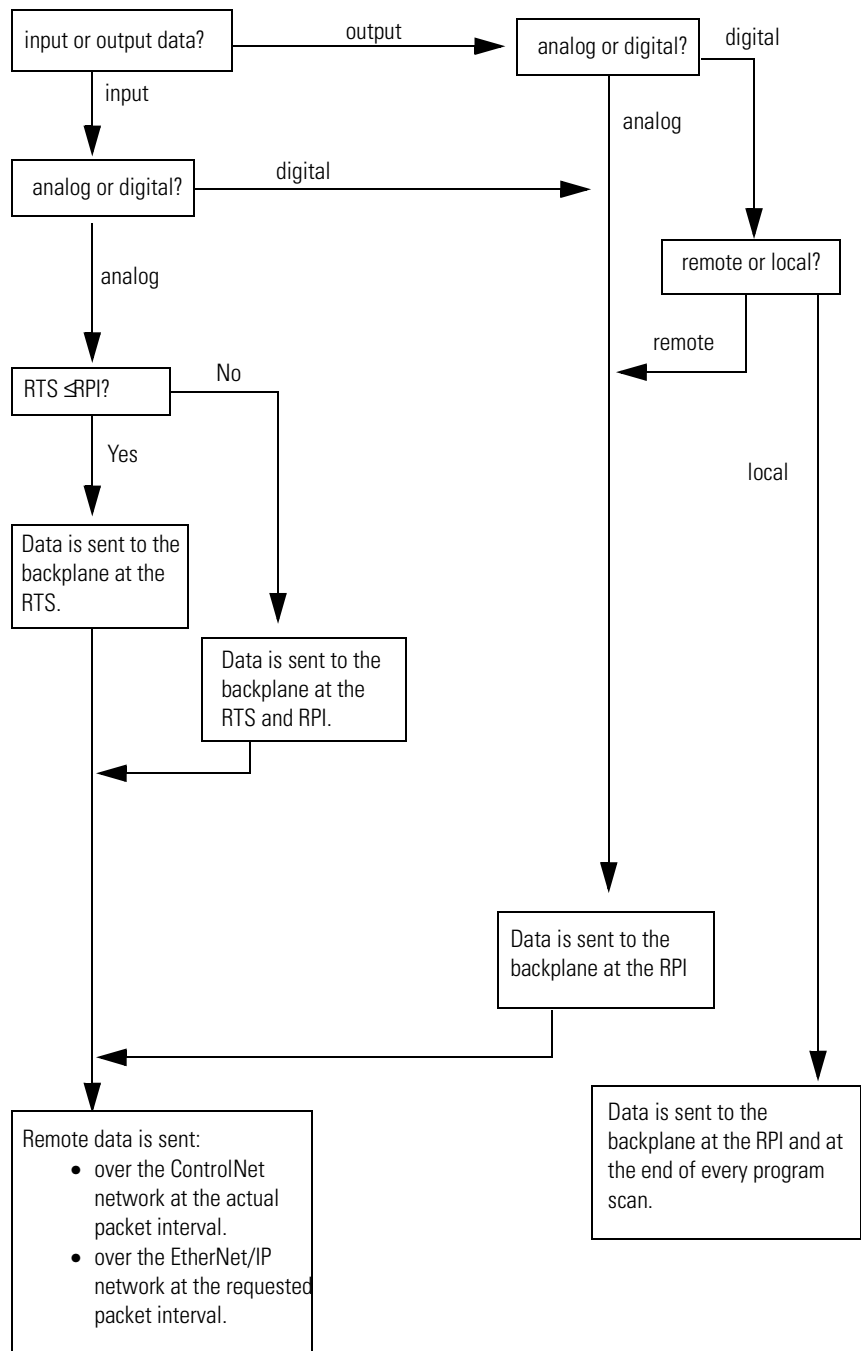
The DriveLogix system follows a producer/consumer model. Input modules produce data for the system. Controllers, output modules, and intelligent modules produce and consume data. The producer/consumer model multicasts data. This means that multiple nodes can consume the same data at the same time from a single device.

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 and output data transfers to output modules asynchronous to the logic scan.

TIP

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

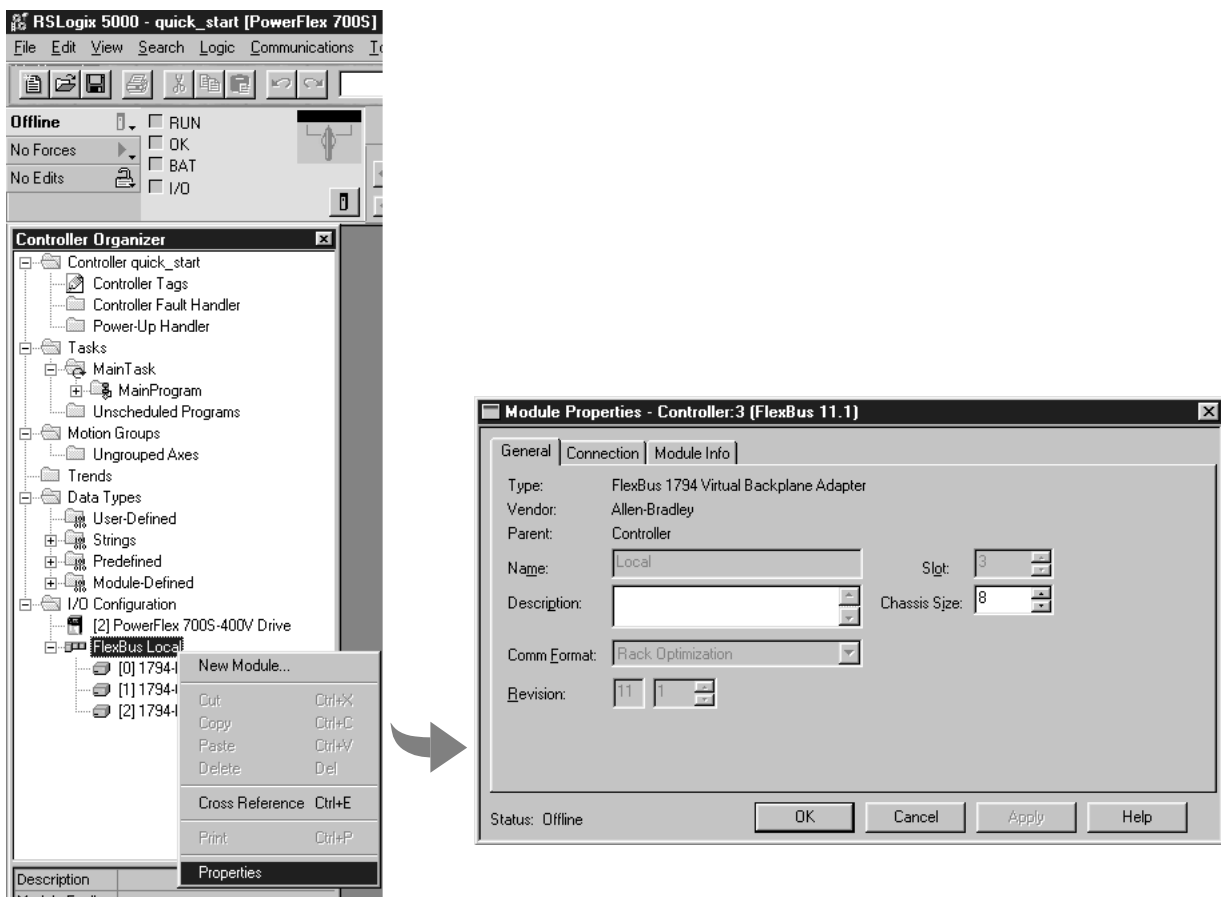
Use the following flowchart to determine when a producer (controller, input module, or ControlNet bridge module) will send data.



Configuring a DIN Rail

When you create a DriveLogix project, the programming software automatically creates the DIN rail for the project. You must configure the DIN rail.

1. In the Controller Organizer, select the local (Local) rail of the controller. Right-click and select Properties.
2. Specify the configuration options for the rail.



You must specify these characteristics:

- On the General tab, specify the size of the chassis. Enter the number of modules (1-8) that you plan to install on the rail. The controller uses this chassis size to determine the size of tag for the rail's rack-optimized data.
- On the Connection tab, specify the RPI. The RPI of the DIN rail applies to all the I/O modules you install on that DIN rail.

IMPORTANT

Set the RPI of the DIN rail to a minimum of 30 ms. I/O function will be intermittent if the value is set lower,

IMPORTANT

If there are no modules installed on a rail, make sure to inhibit that rail.

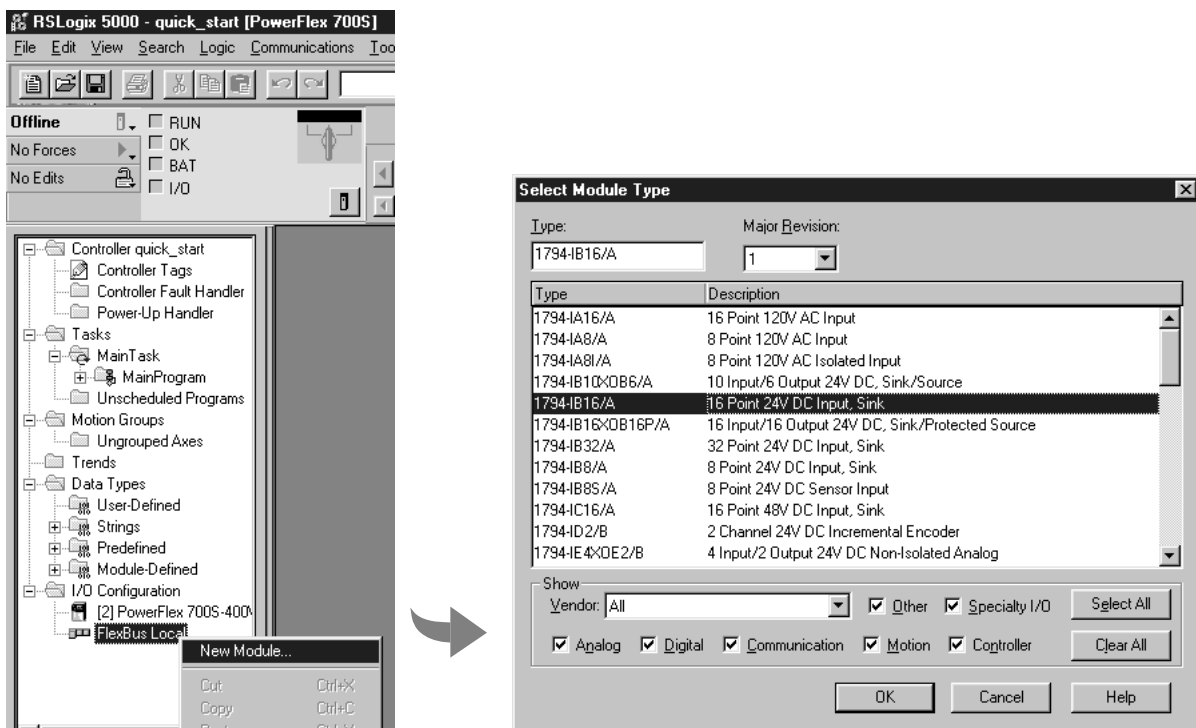
The communication format for the DIN rail is automatically set for rack-optimized. You cannot change this setting because the controller uses one rack-optimized connection for each DIN rail, whether you configure any I/O modules for rack-optimized or not.

Configuring Local I/O Modules

Use the programming software to configure the I/O modules for the controller. You can configure I/O modules for the local rail. Before you configure I/O modules, specify the RPI rate for the DIN rail. All the I/O modules on the DIN rail operate at this RPI. The DIN rail always operates as rack optimized.

To configure an I/O module:

1. In the Controller Organizer, select either the local or the extended-local rail of the controller. Right-click the selected rail and select New Module.



3. Configure the module. Use the module wizard to specify characteristics for the module. Click Next to continue through the wizard.

The selection you make for the Comm Format determines the connections required for the I/O module. Once you complete adding a module, you cannot change this selection. See page 4-8.

IMPORTANT

The DriveLogix controller supports FLEX and FLEX Ex I/O modules, but these I/O modules do not behave the same. If you have a communication or program fault with a FLEX I/O module that is configured for “Reset Outputs,” the outputs of the module go to zero (as expected). If the same fault occurs with a FLEX Ex module that is configured for “Reset Outputs,” the adapter goes to its safe state. If the module itself is defined as “ON,” the outputs actually turn on (don’t reset as expected).

Electronic keying

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

Keying:	Description:
compatible module	The module must be compatible with the software configuration. These characteristics must match: <ul style="list-style-type: none"> • module type • catalog number
disable keying	No attributes of the software or hardware are required to match.



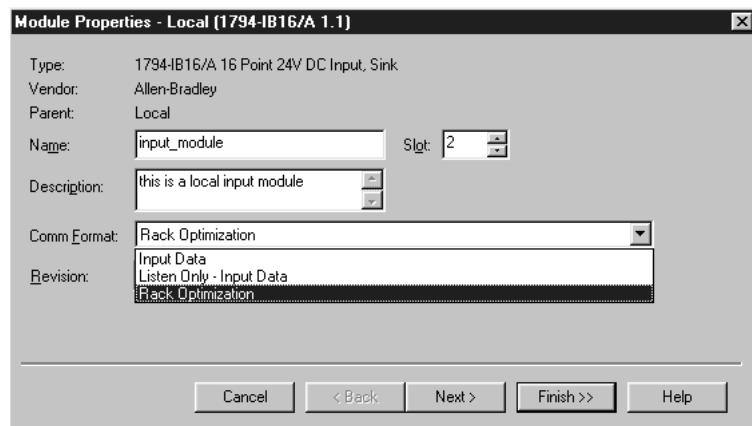
ATTENTION If a module is configured for a direct connection, changing the RPI and electronic keying selections can cause the connection to the module to be broken and may result in loss of data.

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

Communication formats

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.

You select the communications format when you configure the I/O module.



The default communication format for an I/O module is for a direct connection. Each rail for the DriveLogix controller is automatically configured

for a rack-optimized connection, so consider configuring all local I/O modules for rack-optimized connections.

Use the documentation for the I/O module to determine what data format to use.

The listen-only communication format works for remote I/O only. Because of the distributed nature of a DriveLogix system, the DriveLogix controller must own its local I/O modules. No other Logix-based controller can listen to or own the local DriveLogix I/O. The DriveLogix controller must produce its local I/O data for other controller to consume. If you select listen-only for a local I/O module, the connection to that module will fault.

The following tag structures are possible for a 1794-IA16 module. The communication format determines the structure that is created for the module. Assume that the module is in slot 0. The software creates the appropriate tags using the slot number to differentiate the tags for this example module from any other module.

communication format: input data (which corresponds to a direct connection for the I/O module)

Tag Name	Value	Force Mask	Style	Type	Description
Example_Drive:I	{...}	{...}		AB:DRIVE_VELOCITYC...	
Example_Drive:O	{...}	{...}		AB:DRIVE_VELOCITYC...	
Local:O:C	{...}	{...}		AB:1794_DI_Delay16:C:0	
Local:O:C.Config	2#0000_0000_0000_0000			Binary	INT
Local:O:C.DelayTime_0	0			Decimal	BOOL
Local:O:C.DelayTime_1	0			Decimal	BOOL
Local:O:C.DelayTime_2	0			Decimal	BOOL
Local:O:C.DelayTime_3	0			Decimal	BOOL
Local:O:C.DelayTime_4	0			Decimal	BOOL
Local:O:C.DelayTime_5	0			Decimal	BOOL
Local:O:I	{...}	{...}		AB:1794_DI_16:I:0	
Local:O:I.Fault	2#0000_0000_0000_000...			Binary	DINT
Local:O:I.Data	2#0000_0000_0000_0000			Binary	INT
Local:I	{...}	{...}		AB:1794_AVB_8SLOT:I:0	
Local:O	{...}	{...}		AB:1794_AVB_8SLOT:O:0	

communication format: rack optimization (which corresponds to a rack-optimized connection for the I/O module)

Tag Name	Value	Force Mask	Style	Type	Description
[-] Local:O:C	{...}	{...}		AB:1794_DI_Delay16:C:0	
[-] Local:O:C.Config	2#0000_0000_0000_0000		Binary	INT	
[-] Local:O:C.DelayTime_0	0		Decimal	BOOL	
[-] Local:O:C.DelayTime_1	0		Decimal	BOOL	
[-] Local:O:C.DelayTime_2	0		Decimal	BOOL	
[-] Local:O:C.DelayTime_3	0		Decimal	BOOL	
[-] Local:O:C.DelayTime_4	0		Decimal	BOOL	
[-] Local:O:C.DelayTime_5	0		Decimal	BOOL	
[+] Local:I	2#0000_0000_0000_0000		Binary	INT	
[-] Local:I.0	0		Decimal	BOOL	
[-] Local:I.1	0		Decimal	BOOL	
[-] Local:I.2	0		Decimal	BOOL	
[-] Local:I.3	0		Decimal	BOOL	
[-] Local:I.4	0		Decimal	BOOL	
[-] Local:I.5	0		Decimal	BOOL	
[-] Local:I.6	0		Decimal	BOOL	
[-] Local:I.7	0		Decimal	BOOL	
[-] Local:I.8	0		Decimal	BOOL	
[-] Local:I.9	0		Decimal	BOOL	
[-] Local:I.10	0		Decimal	BOOL	
[-] Local:I.11	0		Decimal	BOOL	
[-] Local:I.12	0		Decimal	BOOL	
[-] Local:I.13	0		Decimal	BOOL	
[-] Local:I.14	0		Decimal	BOOL	
[-] Local:I.15	0		Decimal	BOOL	
[+] Local:I	{...}	{...}		AB:1794_AVB_BSLOT:I:0	

The rack-optimized tags are created as aliases into the array tag Local:I, which is the array for input modules on the local rail. This array contains one element for each slot on the rail (based on the chassis size you specify when you configure the rail). You can either address the rack-optimized module by the alias tag (which uses the slot number) or the array element in the rail tag. If you enter the alias tag in your logic, the programming software displays the base tag.

Local:I contains an element for each possible slot on the rail, whether you actually install an input module there or not. Local:O also contains an element for each possible slot. If you configure a module on the local rail as a direct connection, do not use the associated array element in Local:I or Local:O. Use the tag the software creates for the module (which uses the slot number).

Inhibiting I/O 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. Inhibiting a module shuts down the connection from the controller to that module.

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.

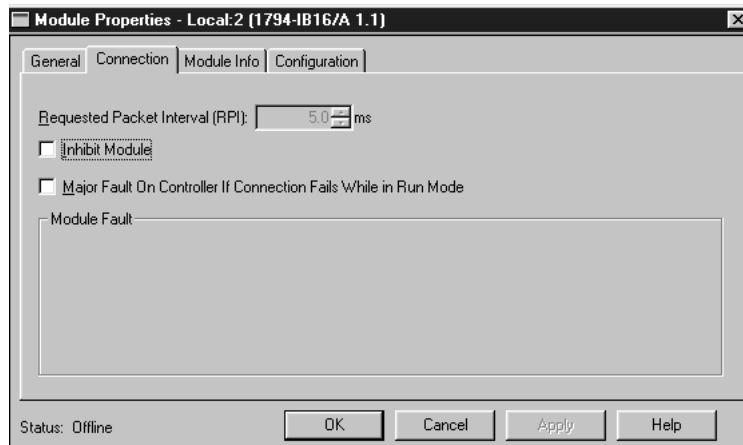
ATTENTION



Inhibiting a module causes the connection to the module to be broken and prevents communication of I/O data. The controller and other I/O modules continue to operate based on old data from that module. To avoid potential personal injury and damage to machinery, make sure this does not create unsafe operation.


Even if you inhibit an I/O module, the FlexBus still scans the module each scan sequence.

You can only inhibit an I/O module if you configured the module to operate with a direct connection. On the Connection tab of the module properties in the programming software, you can select to inhibit that specific module.



To inhibit a rack-optimized connection, you must inhibit the DIN rail, which in turns inhibits all the modules on that rail, whether configured for rack-optimized or direct connections.

When you inhibit a communication module, such as a 1788-CNC communication card, the controller shuts down the connections to the communication card and to all the modules that depend on that card. Inhibiting a communication module lets you disable an entire branch of the I/O network.

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

If you are:	Inhibit a module to:
offline	put a place holder for a module you are configuring

The inhibit status is stored in the project. When you download the project, the module is still inhibited.

online	stop communication to a module
--------	--------------------------------

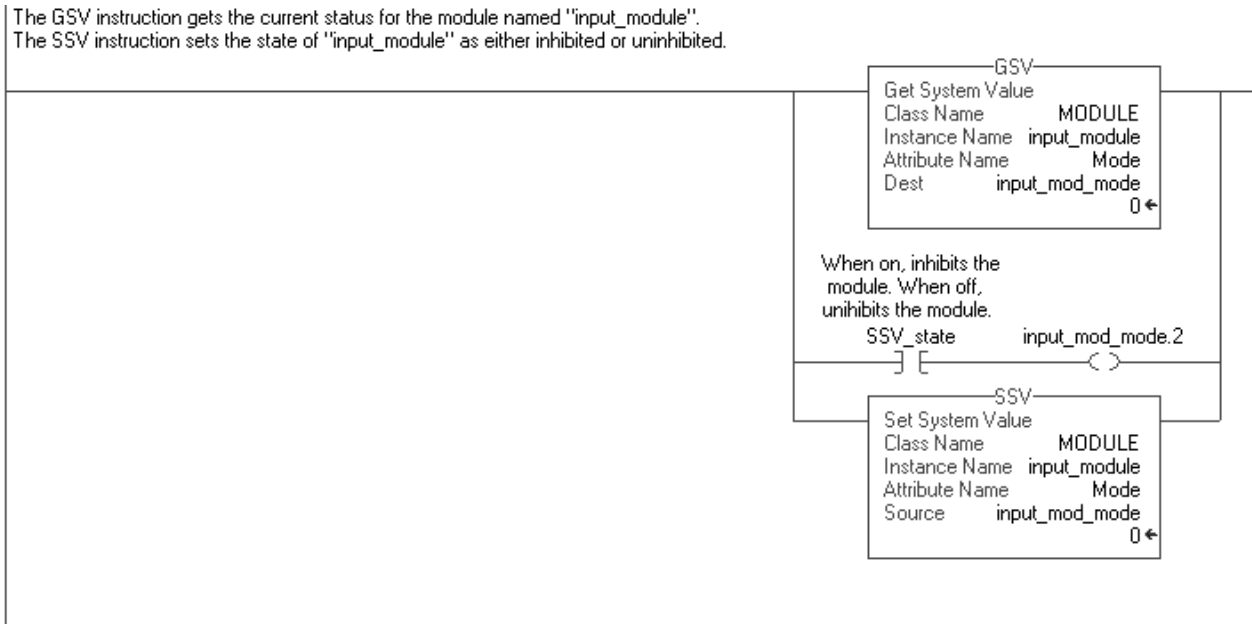
If you inhibit a module while you are connected to the module, the connection to the module is closed. The modules' outputs go to the last configured program mode.

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.

If you uninhibit a module (clear the check box), 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 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.

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:



Accessing I/O Data

The programming software displays I/O data as structures of multiple tags that depend on the specific features of the I/O module. The names of the data structures are based on the location of the I/O module. The programming software automatically creates the necessary structures and tags when you configure the module. Each tag name follows this format:

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

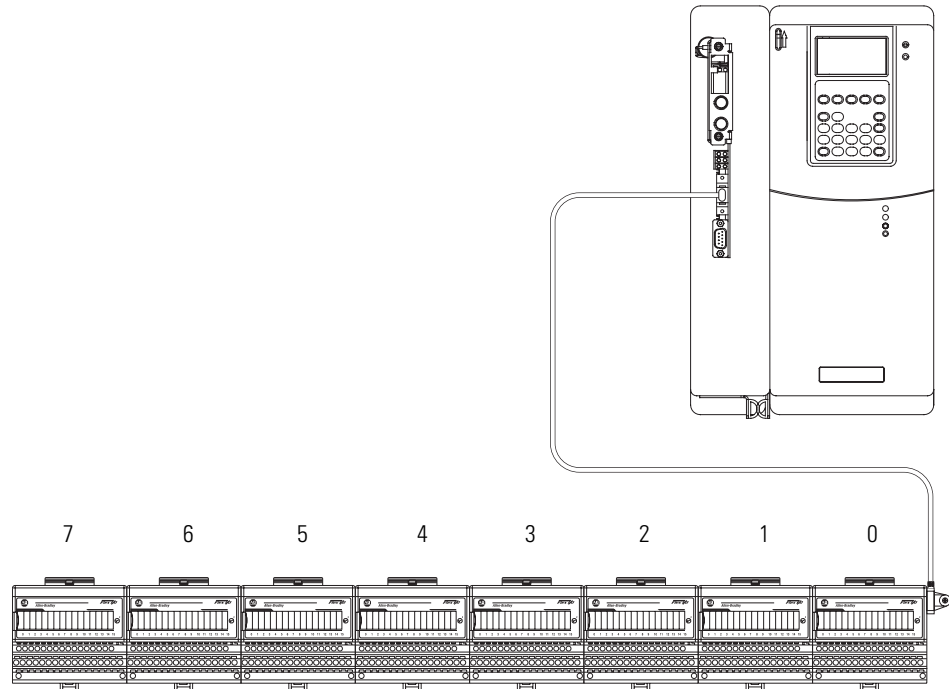
where:

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

The following examples show addresses for data in a DriveLogix system.

EXAMPLE

I/O module on the local DIN rail



Sample tag names for this example:

Location:	Example Tag Name:
input module in slot 0 of LOCAL	Local:0:I.Data
	Local:0:I.Fault
output module in slot 1 of LOCAL	Local:1:C.SSData
	Local:1:I.Fault
	Local:1:O.Data
data for the LOCAL DIN rail	Local:I.Data
	Local:I.Fault
	Local:O.Data
	Local:O.Fault

Using aliases to simplify tag names

An alias lets you create a tag that represents another tag. This is useful for defining descriptive tag names for I/O values. For example:

Example:	Description:
I/O structure	Local:0:0.Data.0
	Local:0:1.Fault.0
alias	light_on = Local:0:0.Data.0
	light_off = Local:0:1.Fault.0

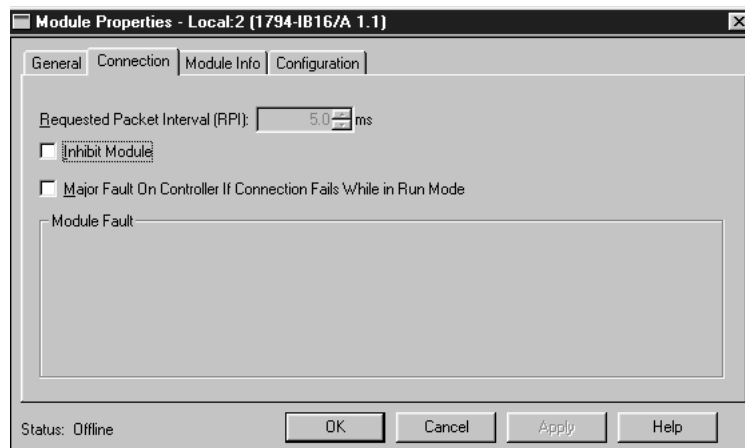
Monitoring I/O Modules

The DriveLogix controller offers different levels at which you can monitor I/O modules. You can:

- configure an I/O module so that the controller faults if that I/O module loses its connection with the controller
- use the programming software to display fault data
- program logic to monitor fault data so you can take appropriate action

Configuring the module's response to a connection failure

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 loses its connection to the controller:

- outputs go to their configured faulted state

- inputs remain in their last, non-faulted state

ATTENTION

If a module loses its connection to the controller, the controller and other I/O modules continue to operate based on old data from that module. To avoid potential personal 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.

Monitoring an I/O module

Most I/O modules have fault bits that indicate when a fault occurs at a specific point of a module. To view this data through the programming software:

1. In the Controller Organizer, select Controller Tags. Right-click to select Monitor Tags.

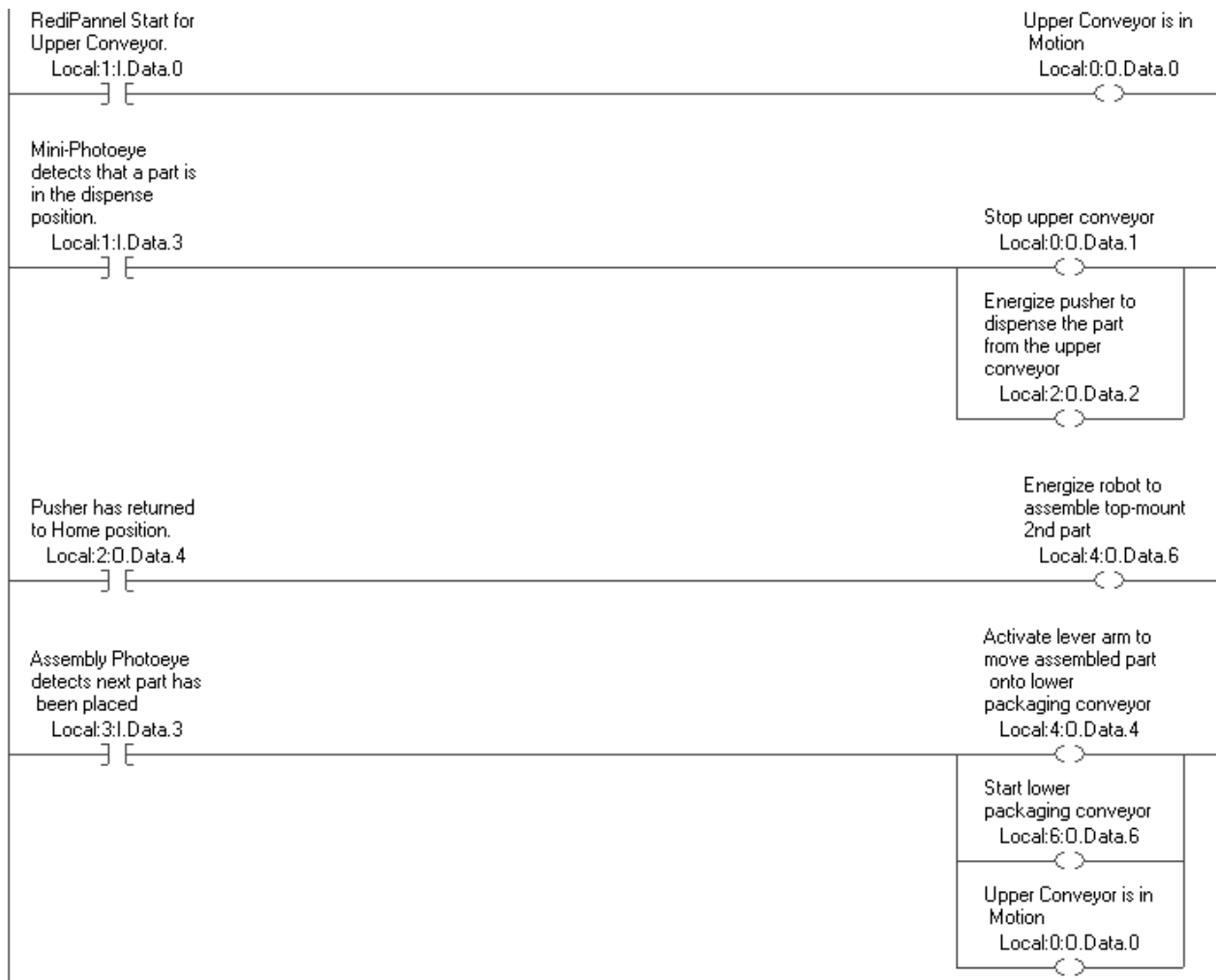
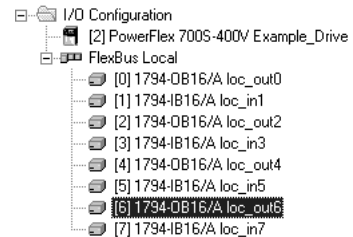
The screenshot shows the RSLogix 5000 software interface. On the left, the Controller Organizer tree is visible, with 'Controller Tags' selected. A right-click context menu is open over 'Controller Tags', and 'Monitor Tags' is highlighted. On the right, the 'Controller Tags - example(controller)' window is displayed, showing a table of tags and their values.

Tag Name	Value
Example_Drive:I	{...}
Example_Drive:O	{...}
Local:I	{...}
Local:O	{...}
Local:OI	{...}
Local:OO	{...}
Local:O:O:Data	2#0000_0000_0000_0000
Local:O:O:Data 0	0
Local:O:O:Data 1	0
Local:O:O:Data 2	0
Local:O:O:Data 3	0
Local:O:O:Data 4	0
Local:O:O:Data 5	n

You can write logic to monitor these bits and take appropriate action if a fault occurs. For example, you may want to shut down the system if a specific point

experiences a fault. This example assumes a direct connection for the I/O module.

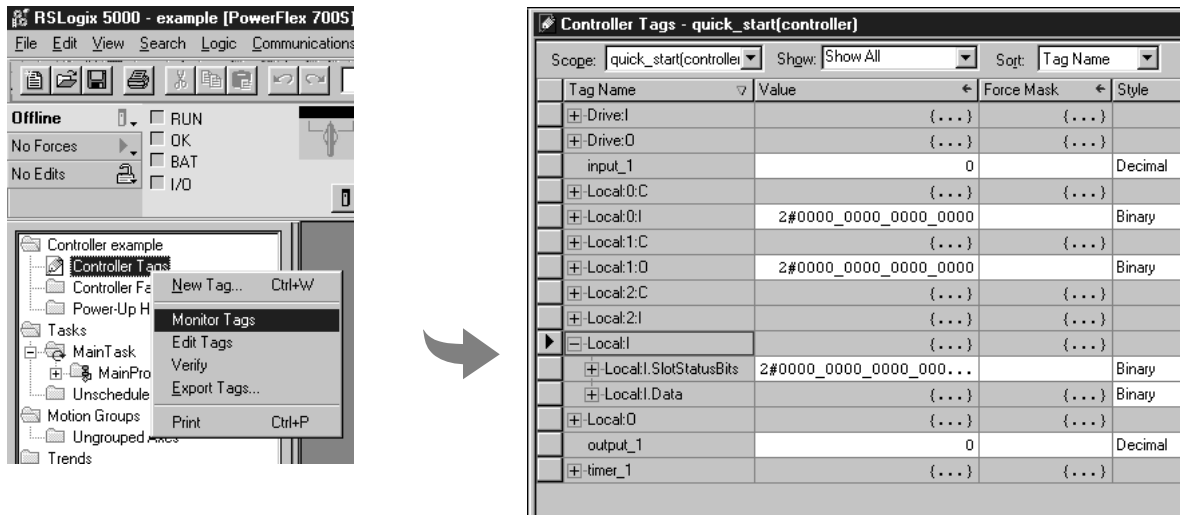
EXAMPLE Given this I/O configuration, the following logic tests bits of I/O modules to determine status.



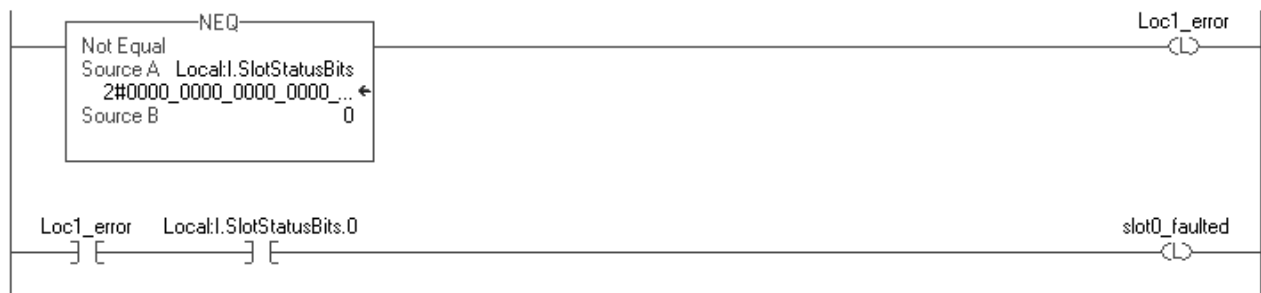
Monitoring a rack-optimized connection

The controller views the DIN rail as another module in the system. Each DIN rail has its own data. To view this data through the programming software:

1. In the Controller Organizer, select Controller Tags. Right-click to display the Data Monitor.



You can write logic to monitor the rack bits and take appropriate action if a fault occurs. For example, the following logic determines whether an error occurs on the Local rail. Then, the logic determines whether the error occurred at the module in slot 0. You can continue this logic to check each module on the rail.



Notes:

Configuring DriveLogix Motion

Using This Chapter

For information about:	See page
Configuring the Drive	5-1
Programming the Controller	5-5
Supported Motion Commands	5-12

This chapter introduces DriveLogix motion. The steps in this chapter provide the minimum settings required to begin testing of DriveLogix motion.

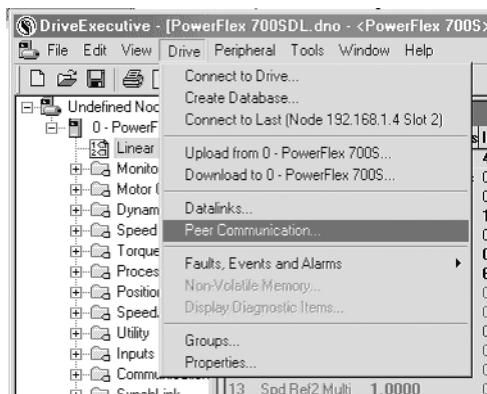
System Requirements

- PowerFlex 700S Drive with firmware revision 2.03 or higher
- DriveLogix controller with firmware revision 12.XX or higher
- DriveExecutive programming software version 2.02 or higher
- RSLogix 5000 programming software version 12 or higher

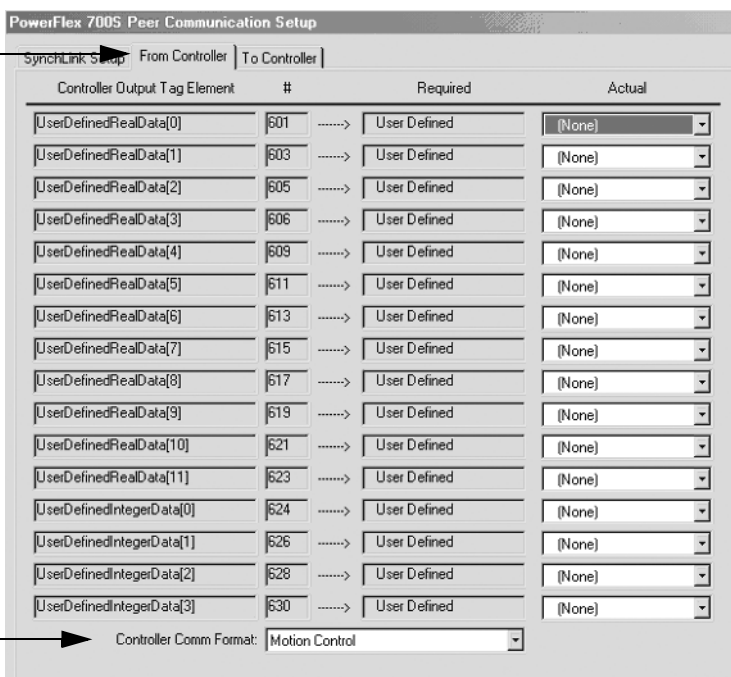
Configuring the Drive

In DriveExecutive software, connect to the drive and access the Peer Communications dialog as shown below:

1. From the File menu, select Peer Communication.



2. Click on the From Controller tab.



3. Select Motion Control from the list of possible Controller Comm Formats.

Next link the appropriate parameters to the words being produced and consumed by the controller.

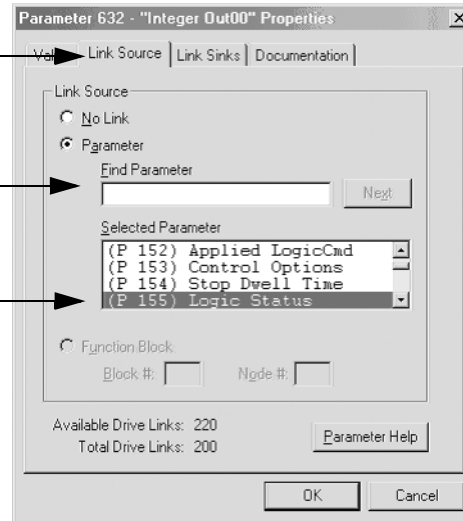
4. Double click on parameter 632 [Integer Out00], from the linear list view. This opens a dialog window for the parameter.

5. Click on the Link Source tab.

6. Type the parameter number (155) into the Find Parameter Window

or

Use the Select Parameter list to choose a parameter source.



Parameter 155 [Logic Status] is the source parameter and parameter 632 [Integer Out00] is the destination. The source will produce data and destination will consume it.

Create the additional links listed Table 5.1.

Table 5.1 Required Parameter Links

Destination Parameter	Source Parameter	Description
748 [CoarsePosit Trgt]	917 [Motn Posit Cmmd]	The position interpolator of the drive receives the coarse position target from the DriveLogix Motion Interpolator
750 [Coarse Spd Trgt]	918 [Motn Speed Cmmd]	The Speed loop of the drive receives the coarse velocity target from the DriveLogix Motion Interpolator
12 [Speed Ref 2]	751 [Interp Speed]	Speed Ref 2 receives the Speed reference from the Coarse to Fine Interpolator

Table 5.1 Required Parameter Links

Destination Parameter	Source Parameter	Description
1003 [Interp SyncInput]	919 [Motn Posit Sync]	The drive receives the synchronization pulse from the DriveLogix. This keeps the interpolators in synch.
632 [Integer Out00]	155 [Logic Status]	The DriveLogix controller receives the status of the drive.
22 [Speed Trim 2]	318 [Posit Spd Output]	This is a default link. The speed loop receives the trim value from the position regulator.

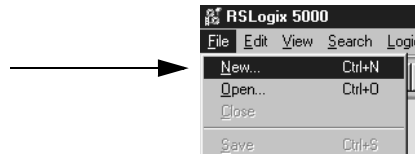
Next set up the parameters in the drive. Double click on the parameter to change and set the values to those listed in Table 5.2.

Table 5.2 Required Parameter Settings

Parameter	Value	Description
13 [Spd Ref2 Multi]	0.92	Sets the scale value for Speed Ref 2.
16 [Speed Ref Sel]	2	Selects Speed Ref 2 as the Speed Reference.
151 [Logic Command]	Bit 13 = 1	This enables the position loop within the drive.
664 [Lgx Comm Format]	19	This selects the format of the commands coming to and from the DriveLogix processor.
740 [Position Control]	Bit 1 = 1 Bit 6 = 0 Bit 8 = 0	This sets up the position regulator to work with the DriveLogix configuration.
742 [Posit Ref Sel]	0	This configures the drive position loop to receive position commands from DriveLogix via the Interpolator input.
1000 [SL Node Cnfg]	Bit 0 = 1	This sets up the SynchLink as the time keeper. This is used to synchronize the Drive and the DriveLogix controller. This must be used when the drive is a standalone system. If it is connected to other SynchLink nodes, only one needs this bit set.

Programming the Controller Using RSLogix 5000, create a new project. .

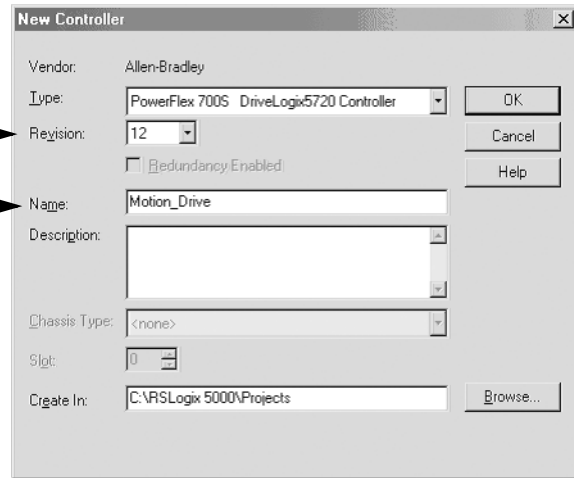
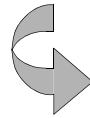
1. Select File →New.



2. Define the project.

Use Revision 12 or higher

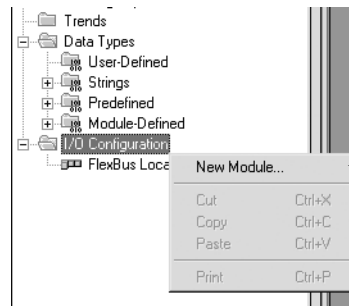
You must enter a name



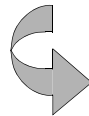
Click OK

3. Add the drive.

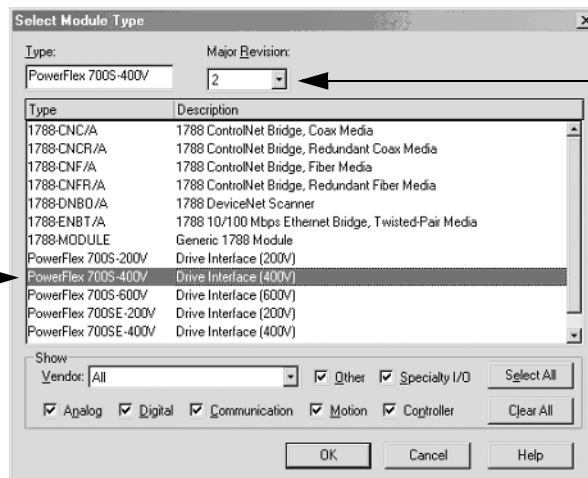
Right click on the I/O Configuration icon



Select New Module



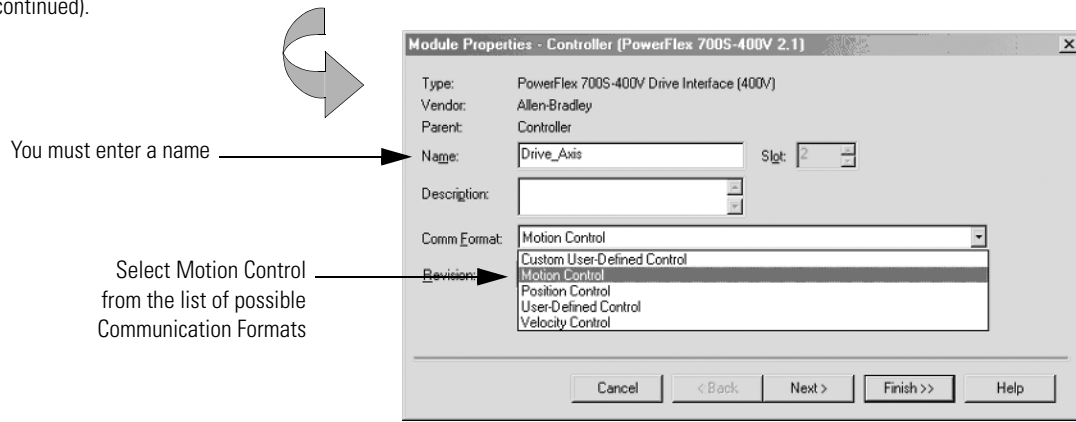
Select the correct drive



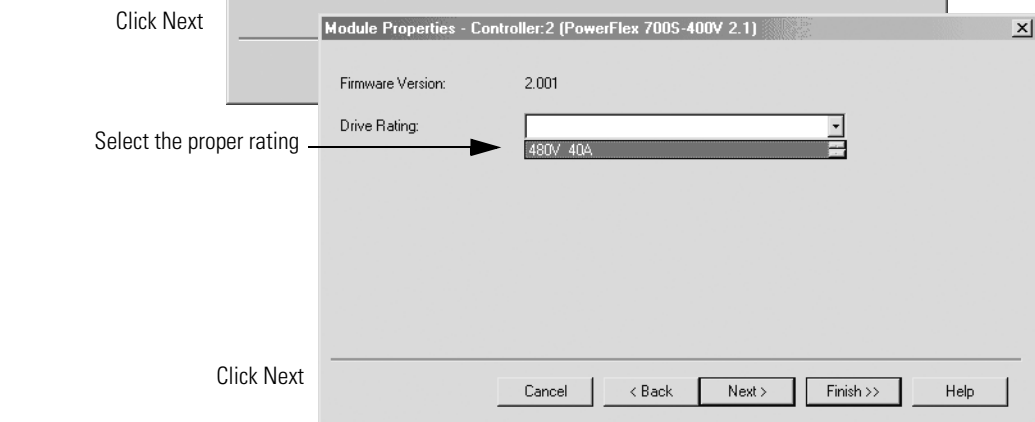
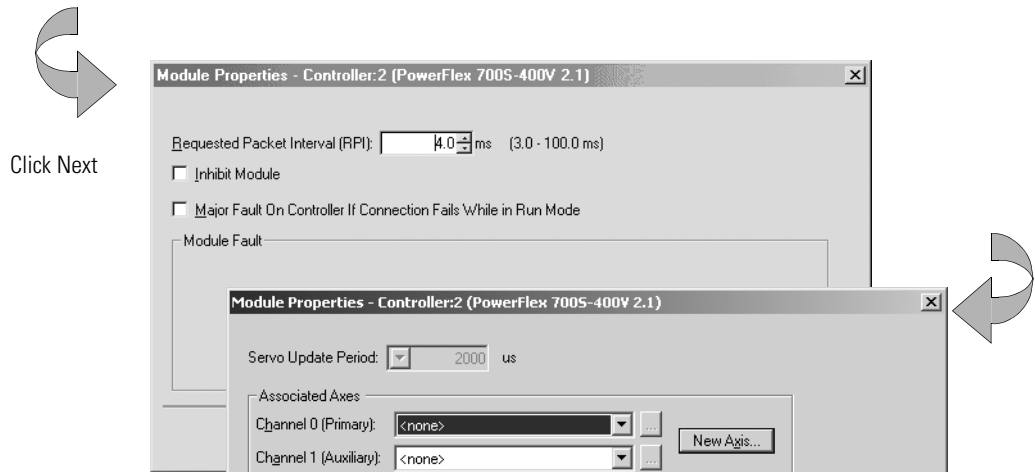
The Major Revision must be 2 or higher

Click OK

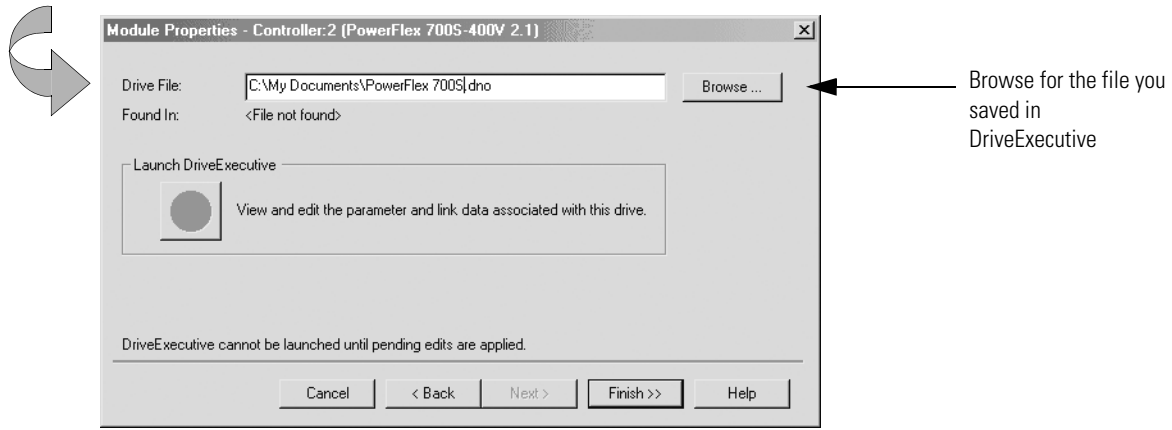
3. Add the drive (continued).



Click Next



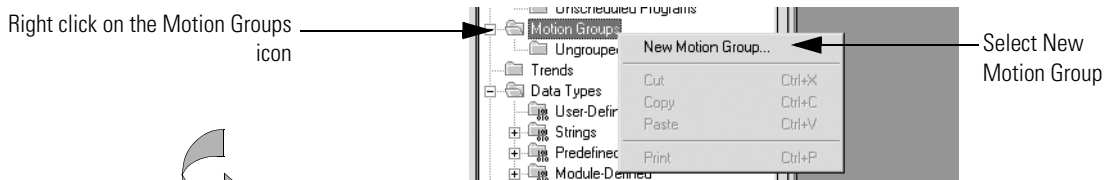
3. Add the drive (continued).



Browse for the file you saved in DriveExecutive

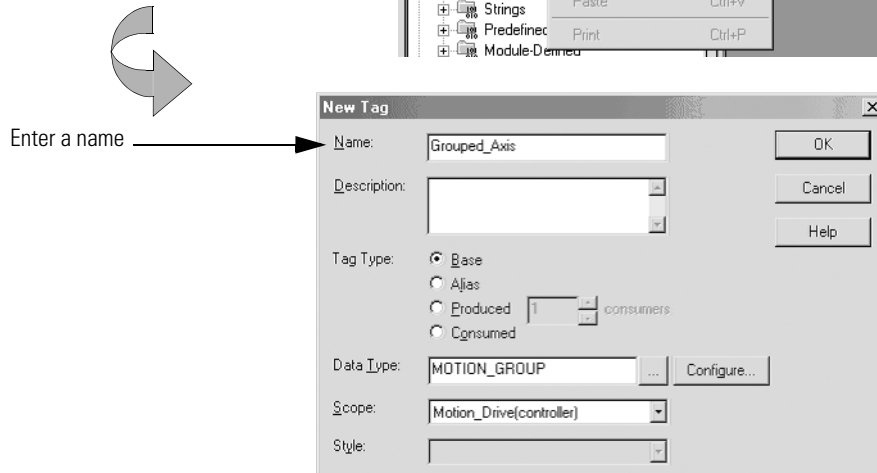
Click Finish

4. Adding a Motion Group.



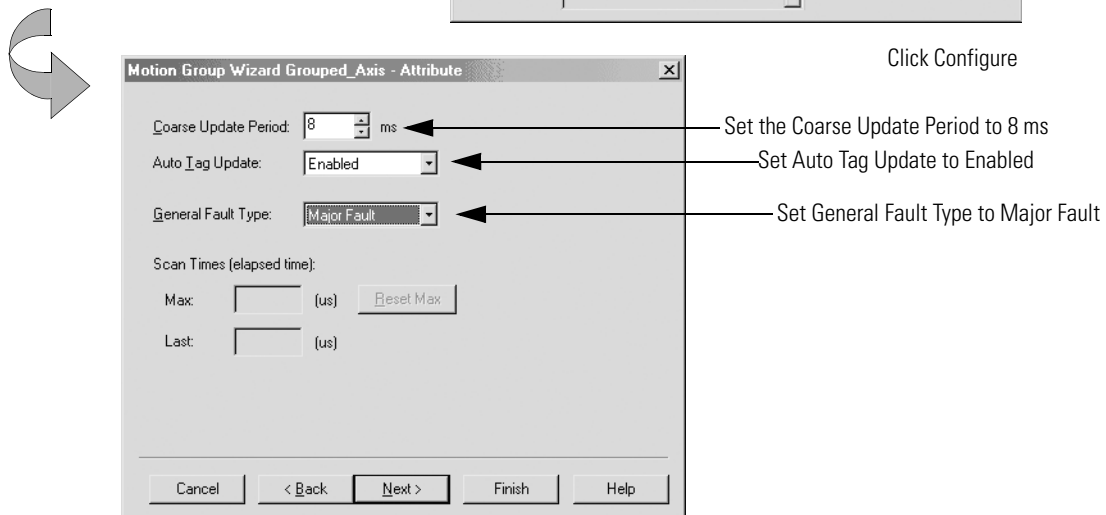
Right click on the Motion Groups icon

Select New Motion Group



Enter a name

Click Configure



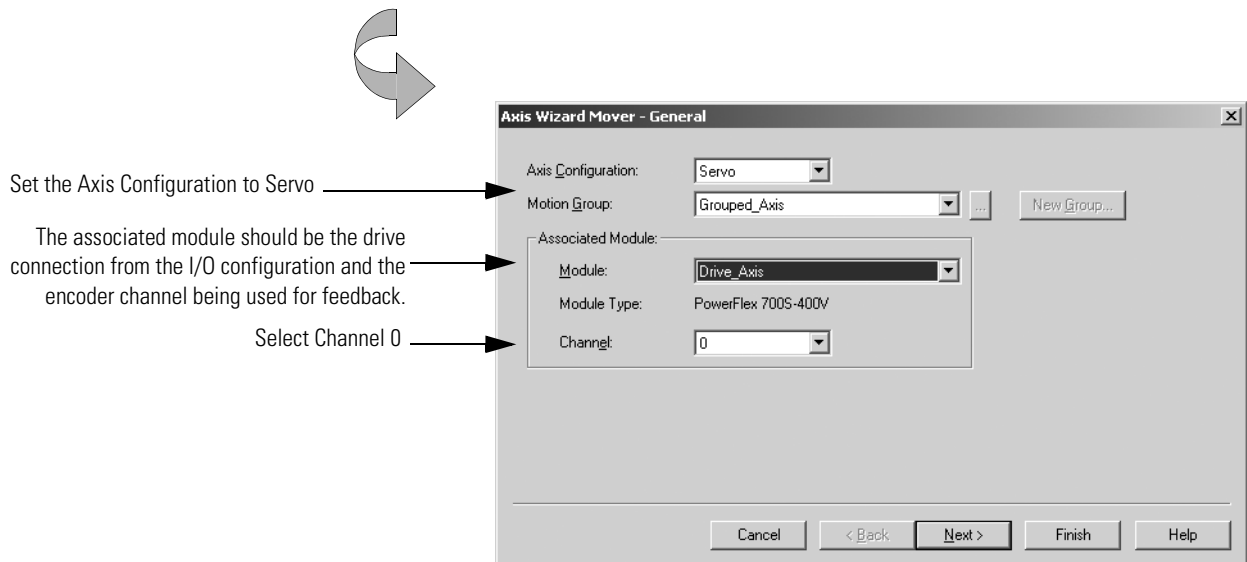
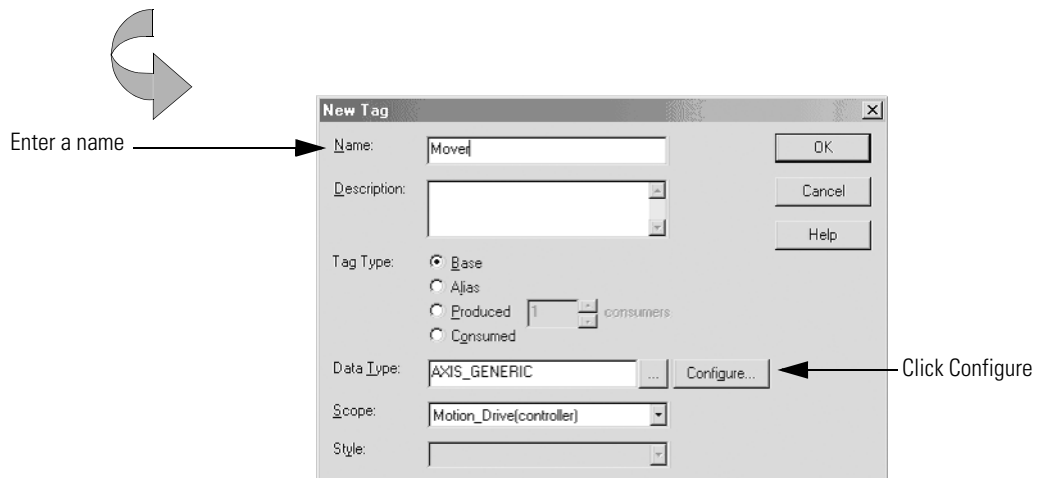
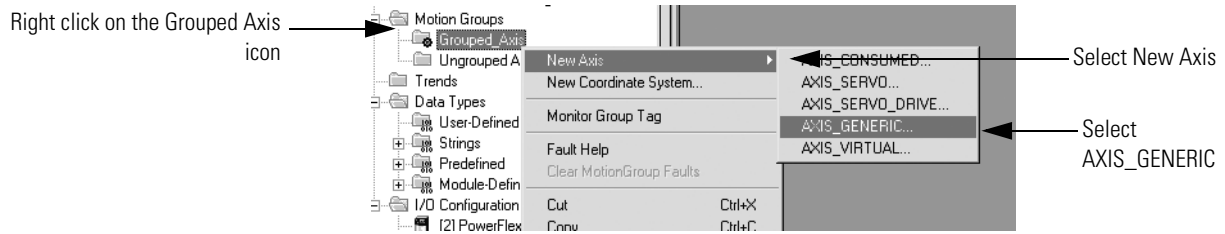
Set the Coarse Update Period to 8 ms

Set Auto Tag Update to Enabled

Set General Fault Type to Major Fault

Click Finish

5. Add the Axis

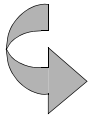


Click Next

IMPORTANT

Only Channel 0 will function for a Servo axis. Channel 1 may be used for a Feedback Only axis.

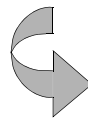
5. Adding the Axis (continued)



Determine how many Output Cam execution nodes (instances) are created for a specific axis.

The value specified for Execution Target in the MAOC instruction references a specific instance in which a value of zero selects the first instance.

Click Next

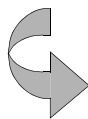


Enter the positioning units.

In this example inches are the units. Degrees, pallets, widgets, etc. could be used.

The Average Velocity Timebase is the sample rate that is used for the Average Velocity tag in the controller tags.

Click Next

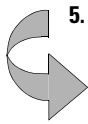


Select the Positioning Mode

Enter the Conversion Constant - this is the number of feedback counts for each Positioning Unit. In this example the encoder is attached to a ball screw with a pitch of 5 turns per inch. The encoder has a resolution of 1024 pulses per revolution that produces 4096 counts of feedback in quadrature mode. Therefore the Conversion Constant = $4096 \times 5 = 20480$ counts / inch.

Click Next

5. Adding the Axis (continued)



Referring to the tables below, enter the Homing Mode, Homing Position, Offset, and Homing Sequence.

Click Next

A	Homing Mode	<p>Active - the desired homing sequence is selected by specifying whether a home limit switch and/or the encoder marker are used for this axis. Active homing sequences always use the trapezoidal velocity profile.</p> <p>Passive - homing redefines the absolute position of the axis on the occurrence of a home switch or encoder marker event. Passive homing is most commonly used to calibrate uncontrolled axes, although it can also be used with controlled axes to create a custom homing sequence. Passive homing, for a given home sequence, works similar to the corresponding active homing sequence, except that no motion is commanded; the controller just waits for the switch and marker events to occur.</p>										
B	Homing Position	<p>Type the desired absolute position, in position units, for the axis after the specified homing sequence has been completed. In most cases, this position will be set to zero, although any value within the software travel limits can be used. After the homing sequence is complete, the axis is left in this position.</p> <p>If the Positioning Mode (set in the Conversion tab) of the axis is Linear, then the home position should be within the travel limits, if enabled. If the Positioning Mode is Rotary, then the home position should be less than the unwind distance in position units.</p>										
C	Offset	Type the desired offset (if any) in position units the axis is to move, upon completion of the homing sequence, to reach the home position. In most cases, this value will be zero.										
D	Homing Sequence	<p>Select the event that will cause the Home Position to be set:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Sequence Type</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>Immediate</td> <td>Sets the Home Position to the present actual position, without motion.</td> </tr> <tr> <td>Switch</td> <td>Sets the Home Position when axis motion encounters a home limit switch.</td> </tr> <tr> <td>Marker</td> <td>Sets the Home Position when axis encounters an encoder marker.</td> </tr> <tr> <td>Switch-Marker</td> <td>Sets the Home Position when axis first encounters a home limit switch, then encounters an encoder marker.</td> </tr> </tbody> </table>	Sequence Type	Description	Immediate	Sets the Home Position to the present actual position, without motion.	Switch	Sets the Home Position when axis motion encounters a home limit switch.	Marker	Sets the Home Position when axis encounters an encoder marker.	Switch-Marker	Sets the Home Position when axis first encounters a home limit switch, then encounters an encoder marker.
Sequence Type	Description											
Immediate	Sets the Home Position to the present actual position, without motion.											
Switch	Sets the Home Position when axis motion encounters a home limit switch.											
Marker	Sets the Home Position when axis encounters an encoder marker.											
Switch-Marker	Sets the Home Position when axis first encounters a home limit switch, then encounters an encoder marker.											

Limit Switch	If a limit switch is used, indicate the normal state of that switch (i.e., before being engaged by the axis during the homing sequence): Normally Open Normally Closed	
Direction	For active homing sequences, except for the Immediate Sequence type, select the desired homing direction:	
	Forward Uni-directional	The axis jogs in the positive axial direction until a homing event (switch or marker) is encountered, then continues in the same direction until axis motion stops (after decelerating or moving the Offset distance).
	Forward Bi-directional	The axis jogs in the positive axial direction until a homing event (switch or marker) is encountered, then reverses direction until motion stops (after decelerating or moving the Offset distance).
	Reverse Uni-directional	The axis jogs in the negative axial direction until a homing event (switch or marker) is encountered, then continues in the same direction until axis motion stops (after decelerating or moving the Offset distance).
	Reverse Bi-directional	The axis jogs in the negative axial direction until a homing event (switch or marker) is encountered, then reverses direction until motion stops (after decelerating or moving the Offset distance).

5. Adding the Axis (continued)

Axis Wizard Mover - Dynamics

Maximum Speed: 100 Inches/s Manual Adjust...

Maximum Acceleration: 50 Inches/s²

Maximum Deceleration: 50 Inches/s²

Cancel < Back Next > Finish Help

Click Finish

Enter the Dynamic Motion variables (Maximum Speed, Acceleration and Deceleration).

Do not leave zero values in these variables.

Do not exceed system limits, examine parameter 9 [Total Inertia].

6. Download the project to the controller.

ATTENTION

Running the system without proper tuning can cause unstable and unpredictable operation. To avoid potential personal injury and damage to machinery, determine the proper values for system dynamics and tune the system before beginning operation.

7. Test the system to determine proper dynamics for the system.

8. Write logic in the controller to move the axis.

Refer to Supported Motion Commands on page 5-12 and publication 1756-RM007D, *Reference Manual - Logix Controller Motion Instruction Set*.

ATTENTION

There is no default Position Error Fault logic in this system. To avoid potential personal injury and damage to machinery, detect Position Error faults, by using parameter links and ladder logic.

Supported Motion Commands

The following Logix Motion Instructions are supported by the DriveLogix controller:

Motion State

- MSO (Motion Servo On)
- MSF (Motion Servo Off)
- MASD (Motion Axis Shutdown)
- MASR (Motion Axis Shutdown Reset)
- MAFR (Motion Axis Fault Reset)

Motion Move

- MAJ (Motion Axis Jog)
- MAM (Motion Axis Move)
- MAS (Motion Axis Stop)
- MAH (Motion Axis Home)
- MAG (Motion Axis Gearing)
- MCD (Motion Change Dynamics)
- MRP (Motion Redine Position)
- MCCP (Motion Calculate Position Profile)
- MAPC (Motion Axis Position Cam)
- MATC (Motion Axis Time Cam)

Motion Event

- MAW (Motion Arm Watch)
- MDW (Motion Disarm Watch)
- MAR (Motion Arm Registration)
- MDR (Motion Disarm Registration)
- MAOC (Motion Arm Output Cam)
- MDOC (Motion Disarm Output Cam)

Motion Group

- MGS (Motion Group Stop)
- MGSD (Motion Group Shutdown)
- MGSR (Motion Group Shutdown Reset)
- MGSP (Motion Group Strobe Position)

Communicating with Devices on an EtherNet/IP Link

Using This Chapter

For information about:	See page
Configuring Your System for a EtherNet/IP Link	6-1
Configuring Remote I/O	6-8
Sending Messages	6-13
Producing and Consuming Data	6-20
Guidelines for Configuring Connections	6-23
Example 1: DriveLogix Controller and Remote I/O	6-23
Example 1: DriveLogix Controller and Remote I/O	6-25
Example 3: DriveLogix Controller to Other Devices	6-29

Configuring Your System for a EtherNet/IP Link

For the DriveLogix controller to operate on an Ethernet network, you need:

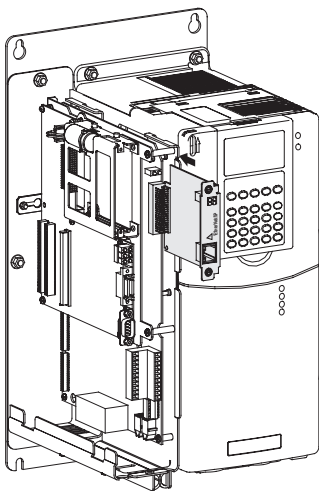
- a workstation with an appropriate EtherNet/IP communication daughtercard
- a 1788-ENBT communication daughtercard installed in the DriveLogix communication slot
- RSLinx software to configure the EtherNet/IP communication driver
- RSLogix 5000 programming software (Version 11 or later) to configure the 1788-ENBT communication daughtercard as part of the DriveLogix system

IMPORTANT

Unlike ControlNet, the EtherNet/IP network requires no scheduling.

Step 1: Configure the hardware

Before you can connect the DriveLogix system to the Ethernet network, you must configure the 1788-ENBT communication daughtercard and make sure it's properly installed in the DriveLogix controller. Refer to Access Procedures on page C-1 to understand how to gain access to the NetLinx daughtercard slot on the DriveLogix controller.



You'll need to configure the communication daughtercard slot number to 1 in the RSLogix 5000 programming software. The DriveLogix controller uses slot 0.

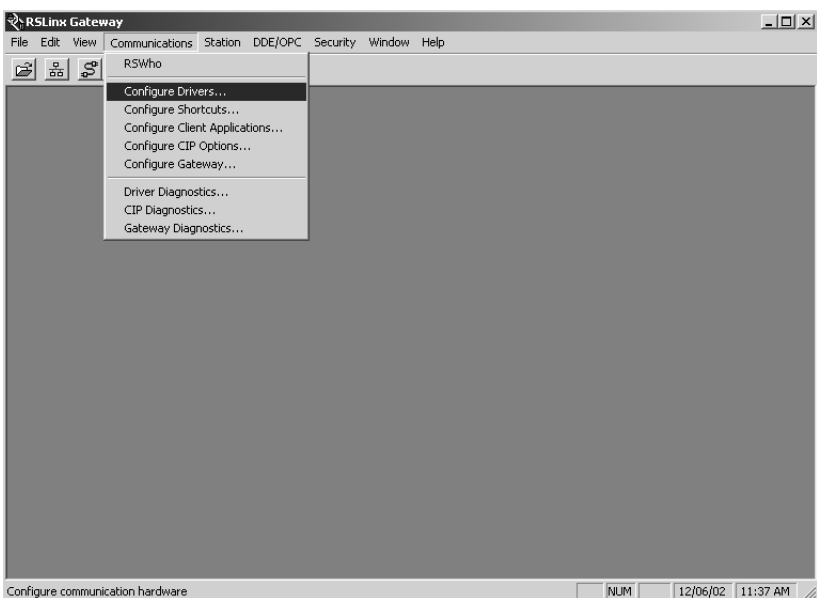
For more information about configuring a 1788-ENBT communication daughtercard, see:

For this card:	See this document:
1788-ENBT	1788-IN054

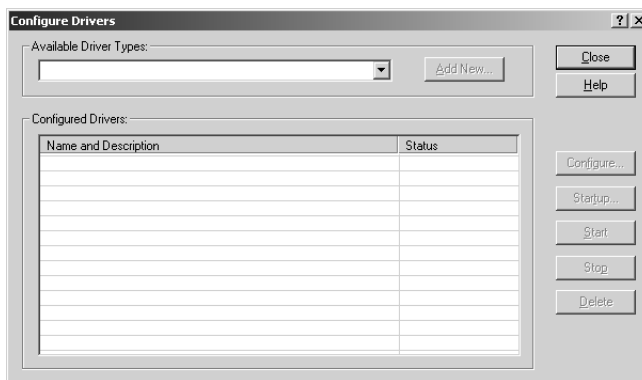
Step 2: Configuring the AB_ETH Driver

To configure the AB_ETH Ethernet communication driver perform the following steps:

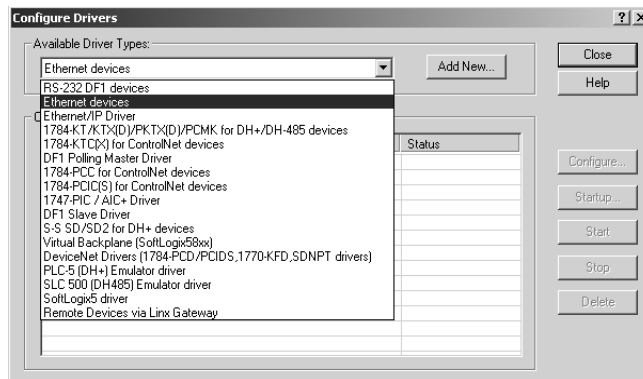
1. Start **RSLinx**.



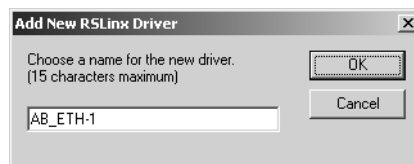
2. From the **Communications** menu, select **Configure Drivers**. The following window will open.



- Click on the arrow to the right of the **Available Driver Types** box. The **Available Driver Types** list will appear.

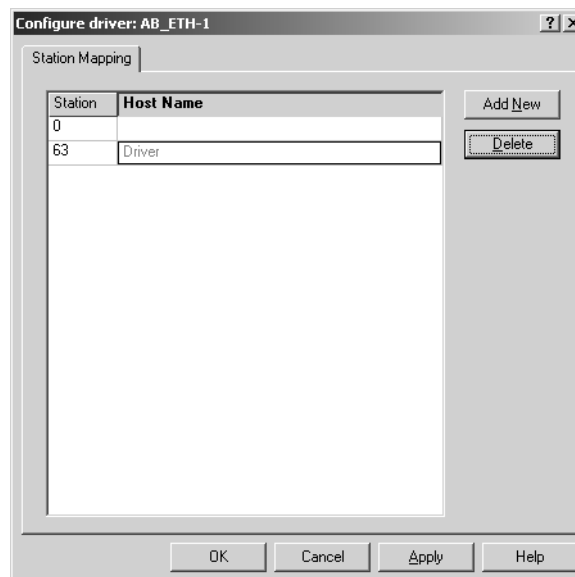


- Select **Ethernet Devices** and click on **Add/New**. You will be prompted to name the driver.



- Select the default driver name (e.g., AB_ETH-1) or type in your own name and click on **OK**.

The **Configure driver** window will appear with the **Station Mapping** page open.



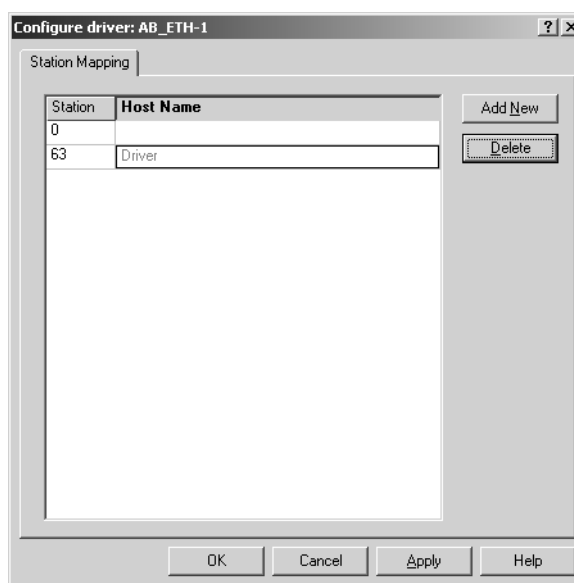
- Click on **Add New**.

7. Enter the IP address or Host Name of your 1788-ENBT module (e.g., 130.130.130.2, “Pump1”, etc.). Use of the IP address on this screen informs the controller of the daughtercard’s IP address for processes such as ladder logic and I/O data exchange. You can set the IP address in any of these ways:

- Rockwell BootP Utility
- RSLinx software
- Third-party BootP server
- DHCP network server

For more information on using these tools, see the EtherNet/IP Communication Daughtercard user manual, publication 1788-UM054.

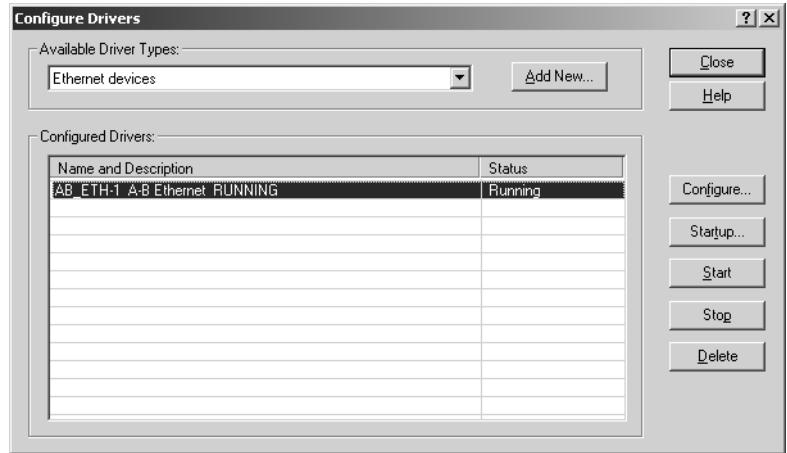
8. Repeat step 6 for each additional Ethernet module you need to access.



9. When you are done entering the IP addresses, click on **Apply**.

10. Click on **OK** to close the **Configure driver** window.

The new driver will appear in the list of configured drivers. (Your list will display the drivers you have configured on your workstation.)

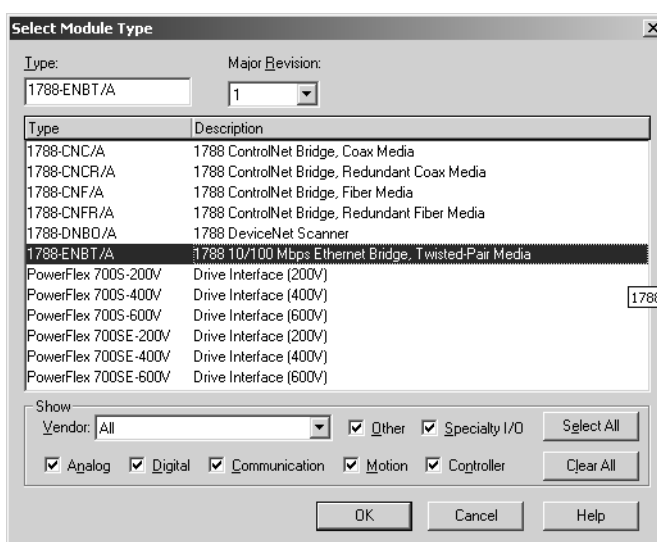
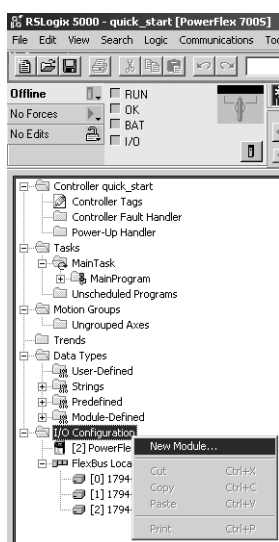


11. Close RSLinx

Step 3: Configure the daughtercard as part of the system

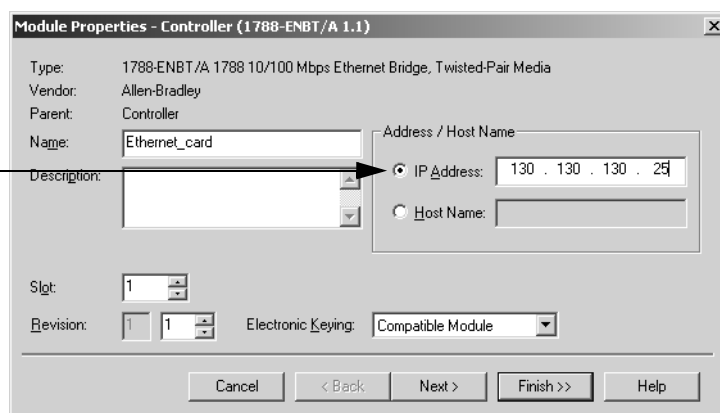
Use RSLogix 5000 programming software (Version 11 or later) to map the 1788-ENBT communication daughtercard as part of the DriveLogix system. In the Controller Organizer, add the communication daughtercard to the I/O Configuration folder.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.
2. Right-click to select New Module and add a 1788-ENBT communication daughtercard.
3. Specify the appropriate communication daughtercard settings.



4. Specify (while offline) the IP address of the communication daughtercard that you installed. Use of the IP address on this screen informs the controller of the daughtercard's IP address for processes such as ladder logic and I/O data exchange.

IMPORTANT: When the project is online, you can also specify the IP address on the Port Configuration screen, if you did not already use the Bootp tool to specify an IP address. When you specify an IP address on the Port Configuration screen, you assign the IP address to the device. If you specify an IP address on the Port Configuration screen, make sure it matches the IP address on the General screen.



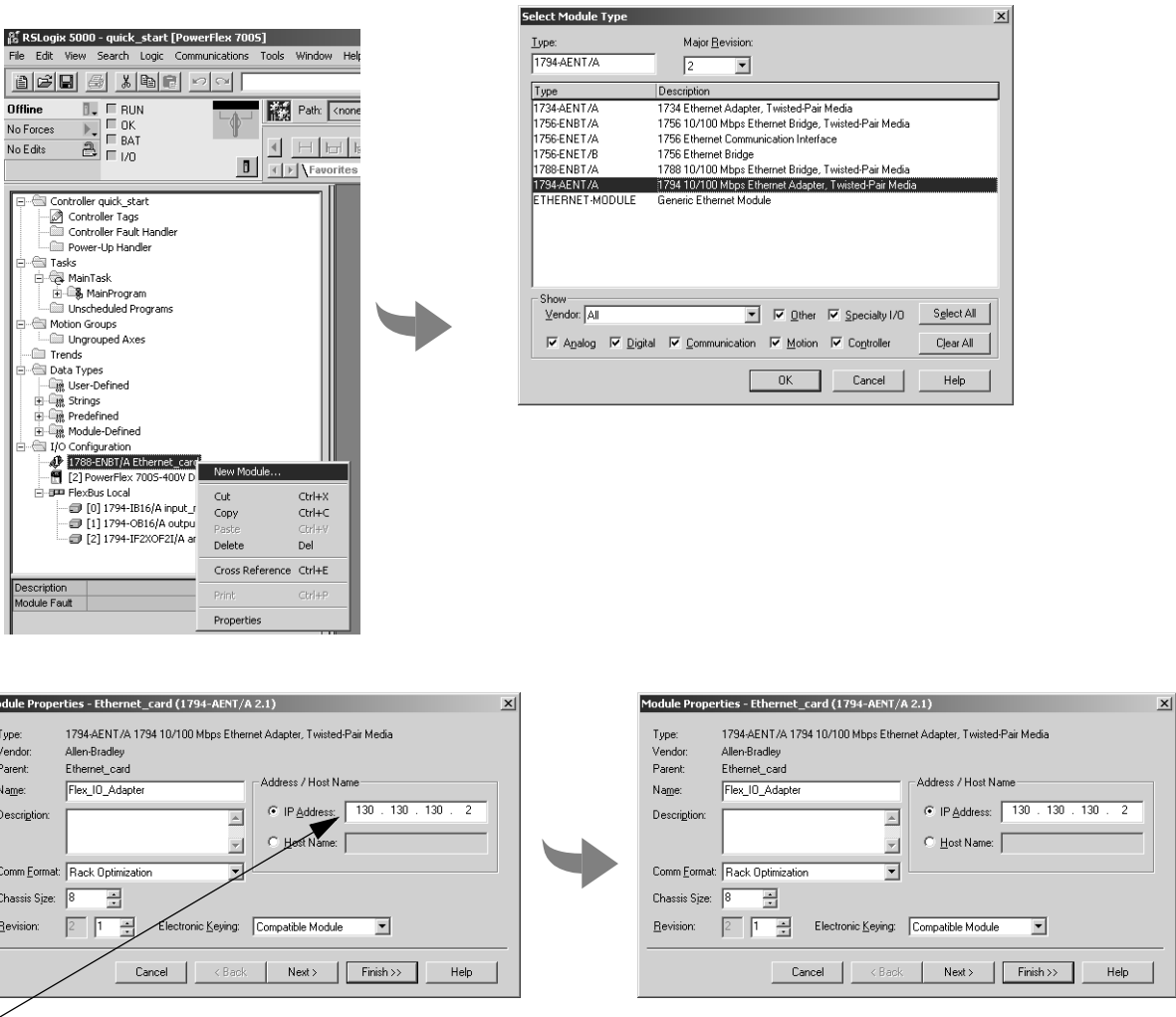
Complete your system configuration and develop your program logic. Then download the project to the DriveLogix controller.

Configuring Remote I/O

The DriveLogix controller supports remote I/O over an EtherNet/IP link. Configuring I/O in a remote chassis is similar to configuring local I/O. The difference is that you must also configure the communication daughtercard (1788-ENBT) in the local chassis and the communication module in the remote chassis.

Add the FLEX I/O Ethernet Adapter to the I/O Configuration

1. In RSLogix 5000 programming software, select 1788-ENBT communication daughtercard.
2. Right-click to select New Module and add a 1794-AENT Ethernet adapter.

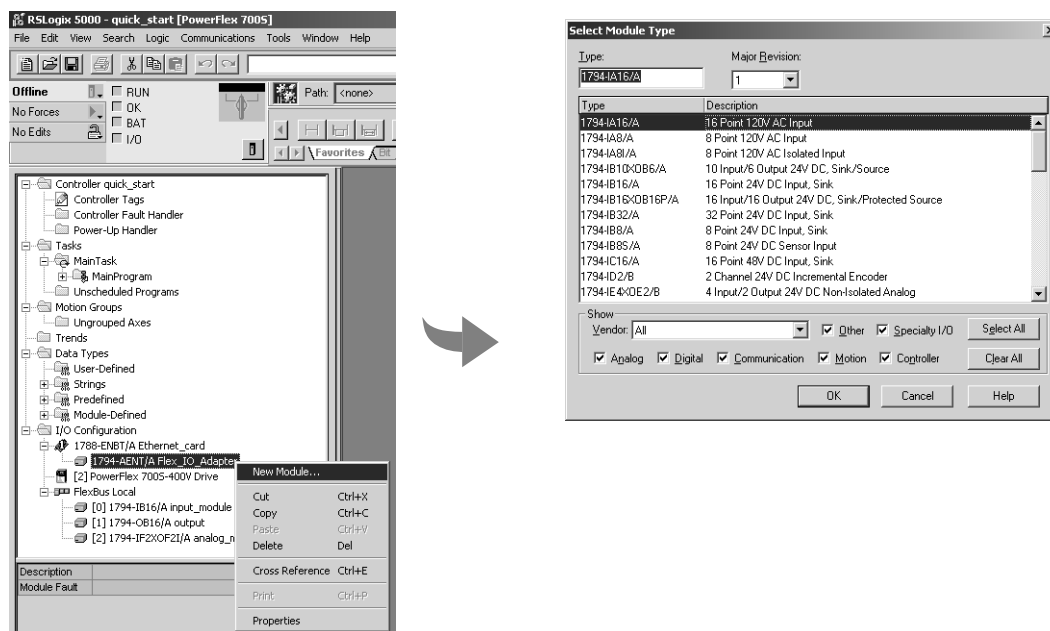


4. Specify (while offline) the IP address of the adapter that you installed. Use of the IP address on this screen informs the controller of the adapter's IP address for processes such as ladder logic and I/O data exchange.

IMPORTANT: When the project is online, you can also specify the IP address on the Port Configuration screen, if you did not already use the Bootp tool to specify an IP address. When you specify an IP address on the Port Configuration screen, you assign the IP address to the device. If you specify an IP address on the Port Configuration screen, make

Add FLEX I/O Modules to the I/O Configuration

1. In RSLogix 5000 programming software, select 1794-AENT Ethernet adapter.
2. Right-click to select New Module and add the appropriate FLEX I/O module.
3. Specify the appropriate module settings.



After you select the appropriate FLEX I/O module, the Module Properties window opens.

4. Configure the module.
5. Add additional modules as needed.

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

Accessing remote I/O

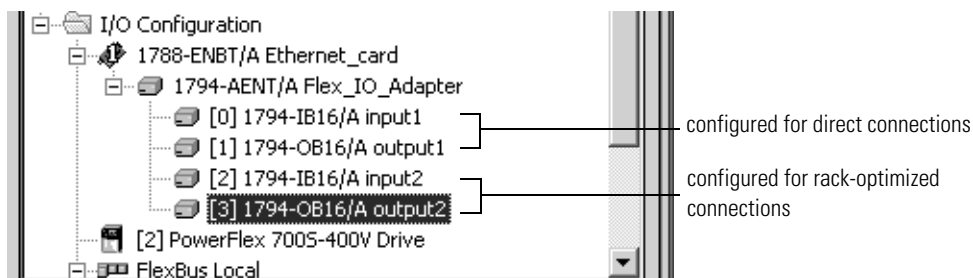
I/O information is presented as a structure of multiple fields, which depend on the specific features of the I/O module. The name of the structure is based on the location of the I/O module in the system. Each I/O tag is automatically created when you configure the I/O module through the programming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

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

where:

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

EXAMPLE



Device:	Example Tag Names (automatically created by the software):
remote adapter "FLEX_adapter"	FLEX_adapter:I FLEX_adapter:I.SlotStatusBits FLEX_adapter:I.Data FLEX_adapter:O FLEX_adapter:O.Data
remote "input1" in slot 0	FLEX_adapter:O:C
direct connection	FLEX_adapter:O:C.Config FLEX_adapter:O:C.Filter0_00_11 FLEX_adapter:O:C.Filter1_00_11 FLEX_adapter:O:C.Filter2_00_11 FLEX_adapter:O:C.Filter3_12_15 FLEX_adapter:O:C.Filter4_12_15 FLEX_adapter:O:C.Filter5_12_15 FLEX_adapter:O:C.ResetCounter FLEX_adapter:O:C.DisableFilter FLEX_adapter:O:I FLEX_adapter:O:I.Fault FLEX_adapter:O:I.Data FLEX_adapter:O:I.Counter

Device:	Example Tag Names (automatically created by the software):
remote "output1" in slot 1	FLEX_adapter:1:C
direct connection	FLEX_adapter:1:C.SSDate FLEX_adapter:1:I FLEX_adapter:1:I.Fault FLEX_adapter:1:O FLEX_adapter:1:O.Data
remote "input2" in slot 2	FLEX_adapter:2:C
rack-optimized connection	FLEX_adapter:2:C.Config FLEX_adapter:2:C.Filter0_00_11 FLEX_adapter:2:C.Filter1_00_11 FLEX_adapter:2:C.Filter2_00_11 FLEX_adapter:2:C.Filter3_12_15 FLEX_adapter:2:C.Filter4_12_15 FLEX_adapter:2:C.Filter5_12_15 FLEX_adapter:2:C.ResetCounter FLEX_adapter:2:C.DisableFilter FLEX_adapter:2:I
These tags are created as aliases into the FLEX_adapter:I tag	
remote "output2" in slot 3	FLEX_adapter:3:C
rack-optimized connection	FLEX_adapter:3:C.SSDate FLEX_adapter:3:O
These tags are created as aliases into the FLEX_adapter:O tag	

For examples of local I/O tags, see Chapter 4, Placing and Configuring Local I/O.

Sending Messages

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

This device:	Support this many unconnected messages:	Support this many connected messages:
1756-ENBT module (for a Logix5550 controller)	256	128
1788-ENBT daughtercard (for a DriveLogix controller)	5	32
1794-AENT adapter (for FLEX I/O)	The 1794-AENT adapter can support a total of 32 messages whether they be connected, unconnected or some combination of both.	
Ethernet PLC-5 controller	32	128

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

This type of message:	And this communication method:	Uses a connection:
CIP data table read or write		X
PLC2, PLC3, PLC5, or SLC (all types)	CIP	
	CIP with Source ID	
	DH+	X
CIP generic	CIP	Optional ⁽¹⁾
block-transfer read or write		X

⁽¹⁾ You can connect CIP generic messages, but for most applications, we recommend you leave CIP generic messages unconnected.

Connected messages are unscheduled connections on EtherNet/IP.

If a MSG instruction uses a connection, you have the option to leave the connection open (cache) or close the connection when the message is done transmitting.

If you:	Then:
Cache the connection	The connection stays open after the MSG instruction is done. This optimizes execution time. Opening a connection each time the message executes increases execution time.
Do not cache the connection	The connection closes after the MSG instruction is done. This frees up that connection for other uses.

The controller has the following limits on the number of connections that you can cache:

If you have this software and firmware revision:	Then you can cache:
11.x or earlier	<ul style="list-style-type: none"> • block transfer messages for up to 16 connections • other types of messages for up to 16 connections
12.x or later	up to 32 connections

Communicating with another Logix-based controller

All Logix-based controllers can use MSG instructions to communicate with each other. The following examples show how to use tags in MSG instructions between Logix-based controllers.

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

The source and destination tags:

- must be controller-scoped tags.
- can be of any data type, except for AXIS, MESSAGE, or MOTION_GROUP.

Communicating with other controllers over EtherNet/IP

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

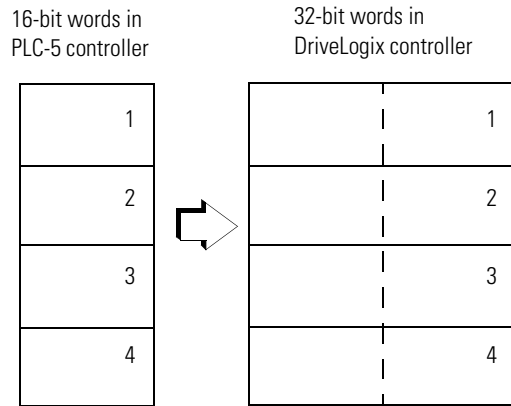
For MSG instructions originating from a DriveLogix controller to a PLC or SLC controller:

Type of MSG Instruction:	Supported Source File Types:	Supported Destination File Types:
DriveLogix writes to PLC-5 or SLC	In the DriveLogix controller, specify the source data type based on the destination device:	Specify the destination file type based on the destination device:
	PLC-5: SINT, INT, DINT, or REAL	PLC-5 typed write: S, B, N, or F
	SLC: INT, REAL	PLC-5 word-range write: S, B, N, F, I, O, A, or D
		SLC: B, N or F
	Example source element: <i>array_1</i>	Example destination tag: <i>N7:10</i>

Type of MSG Instruction:	Supported Source File Types:	Supported Destination File Types:
DriveLogix writes to PLC-2	<p>In the DriveLogix controller, select one of these data types:</p> <p>SINT, INT, DINT, or REAL</p> <p>Example source element: <i>array_1</i></p>	<p>Use the PLC-2 compatibility file.</p> <p>Example destination tag: <i>010</i></p>
DriveLogix reads from PLC-5 or SLC	<p>Specify the destination file type based on the destination device:</p> <p>PLC-5 typed read: S, B, N, or F</p> <p>PLC-5 word-range read: S, B, N, F, I, O, A, or D</p> <p>SLC: B, N or F</p> <p>Example source element: <i>N7:10</i></p>	<p>In the DriveLogix controller, specify the destination data type based on the destination device:</p> <p>PLC-5: SINT, INT, DINT, or REAL</p> <p>SLC: INT, REAL</p> <p>Example destination tag: <i>array_1</i></p>
DriveLogix reads from PLC-2	<p>Use the PLC-2 compatibility file.</p> <p>Example source element: <i>010</i></p>	<p>In the DriveLogix controller, select one of these data types:</p> <p>SINT, INT, DINT, or REAL</p> <p>Example destination tag: <i>array_1</i></p>

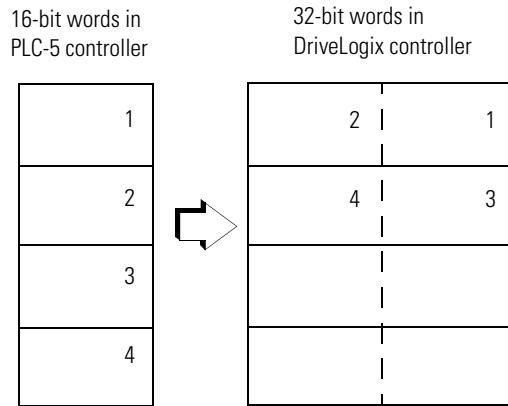
The DriveLogix controller can send typed or word-range commands to PLC-5 controllers. These commands read and write data differently. The following diagrams show how the typed and word-range commands differ.

Typed read command



The typed commands maintain data structure and value.

Word-range read command



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

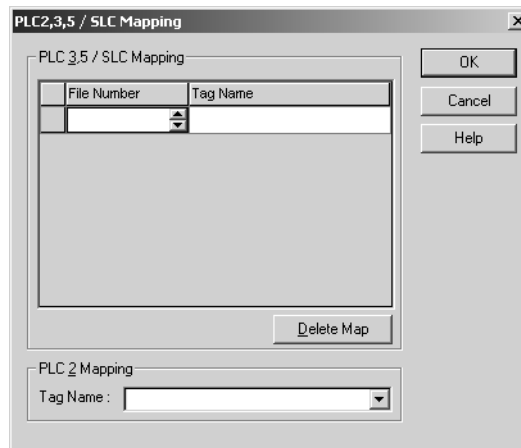
The DriveLogix controller can process messages initiated from PLC or SLC controllers. These messages use data table addresses. In order for these controllers to access tags within the DriveLogix controller, you map tags to data table addresses.

Mapping 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 controllers.

To map addresses:

1. From the Logic menu, select Map PLC/SLC Messages.



2. Specify this information:

For:	In this field:	Specify:	For example:
PLC-3, PLC-5, and SLC controllers	File Number	Type the file number of the data table in the PLC/SLC controller.	<i>10</i>
	Tag Name	Type the array tag name the local controller uses to refer to the PLC/SLC data table address. The tag must be an integer array (SINT, INT, or DINT) that is large enough for the message data.	<i>array_1</i>
PLC-2 controllers	Tag Name	Type the tag name to be the PLC-2 compatibility file.	<i>200</i>

TIP

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

The following table shows example source and destination tags and elements for different controller combinations.

Type of MSG Instruction:	Example Source and Destination:	
PLC-5 writes to DriveLogix	source element	<i>N7:10</i>
	destination tag	<i>"array_1"</i>
SLC writes to DriveLogix	The PLC-5, PLC-3, and SLC controllers 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 controller. Place the DriveLogix tag name in double quotes (").	
SLC 5/05		
SLC 5/04 OS402 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.	
SLC 5/03 OS303 and above		
PLC-2 writes to DriveLogix	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 DriveLogix	source tag	<i>"array_1"</i>
	destination element	<i>N7:10</i>
SLC reads from DriveLogix	The PLC-5, PLC-3, and SLC controllers 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 controller. Place the DriveLogix tag name in double quotes (").	
SLC 5/05		
SLC 5/04 OS402 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.	
SLC 5/03 OS303 and above		
PLC-2 reads from DriveLogix	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 DriveLogix 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 controllers, SLC 5/04 controllers (OS402 and above), and SLC 5/03 controllers (OS303 and above) support logical ASCII addressing and support PLC/SLC mapping (see the examples above). For all other SLC or MicroLogix1000 controllers, you must map a PLC-2 compatibility file (see the PLC-2 examples above).

Producing and Consuming Data

The DriveLogix controller supports the ability to produce (broadcast) and consume (receive) system-shared tags over an EtherNet/IP link. Produced and consumed data is accessible by multiple controllers over an Ethernet network. The controller sends or receives data at a predetermined RPI rate.

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

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

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

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

Maximum number of produced and consumed tags

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

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

Size limit of a produced or consumed tag

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

Producing a tag

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

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

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

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

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

Consuming a tag

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

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

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

All consumed tags are automatically controller-scope.

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

IMPORTANT

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

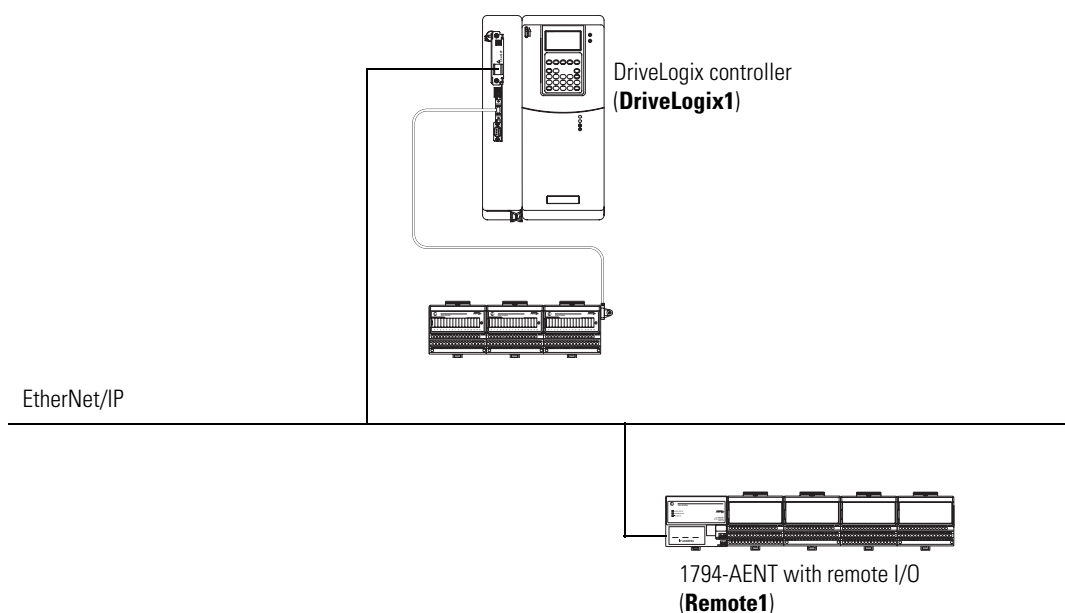
Guidelines for Configuring Connections

Each 1788-ENBT communication daughtercard supports 32 I/O connections. How you configure these connections determines how many devices the daughtercard can support.

If you have two communication daughtercards, use one for communication and the other for remote I/O. While one daughtercard can support both functions, performance can improve by separating these functions onto separate daughtercards.

Example 1: DriveLogix Controller and Remote I/O

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



Example 1: Controlling remote devices

This example has DriveLogix1 controlling the I/O connected to the remote 1794-AENT module. The data the DriveLogix controller receives from the remote I/O modules depends on how you configure the remote I/O modules. You can configure each module as a direct connection or as rack optimized.

One chassis can have a combination of some modules configured as a direct connection and others as rack optimized.

Example 1: Total connections required by DriveLogix1

The following table calculates the connections used in this example.

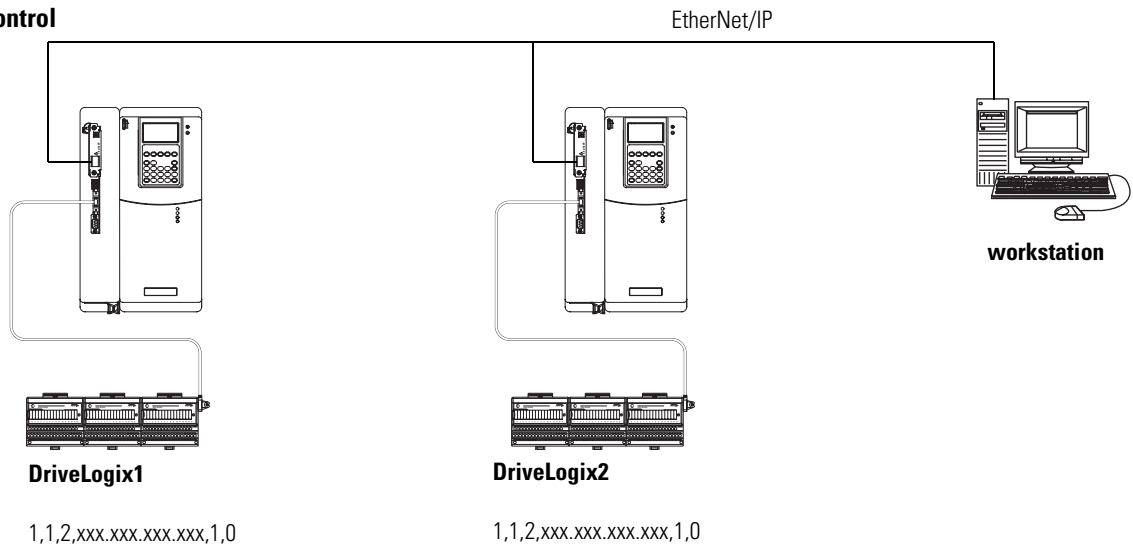
Connection:	Amount:
DriveLogix1 controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for each I/O module	3
DriveLogix1 controller to remote 1794-AENT	1
DriveLogix1 to 4 remote I/O modules (through 1794-AENT)	4
all I/O modules configured as direct connection	
no connection to the 1794-AENT	
total connections used:	9

If you configured the remote I/O modules as rack-optimized, you would only need a rack-optimized connection to the 1794-AENT, reducing the above example by 3 connections.

Example 2: DriveLogix Controller to DriveLogix Controller

In the following example, one DriveLogix controller communicates with another DriveLogix controller over EtherNet/IP. Each DriveLogix controller has its own local I/O

Distributed control

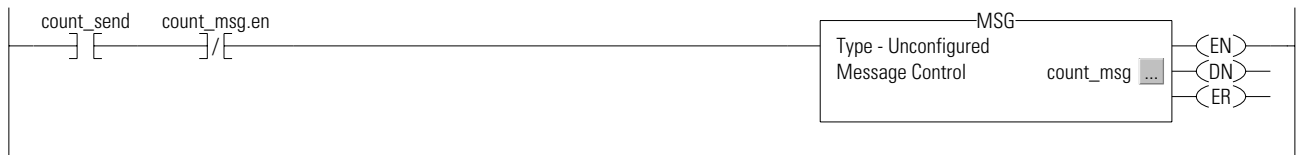


Example 2: Sending a MSG instruction

To send a MSG from DriveLogix1 to DriveLogix2:

1. For DriveLogix1, create a controller-scoped tag and select the MESSAGE data type.
2. Enter a MSG instruction.

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



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

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

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

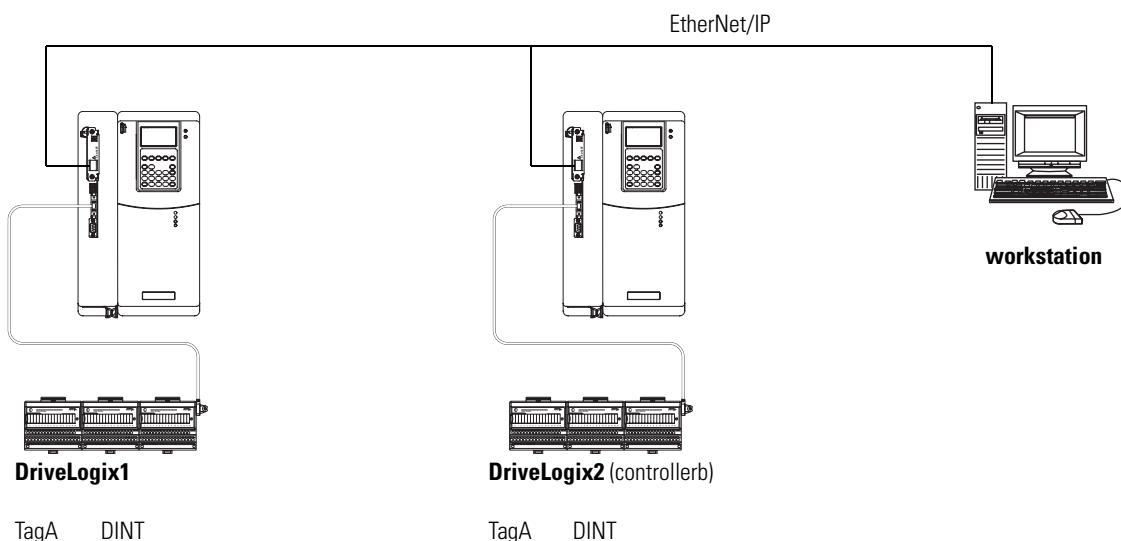
A communication path requires pairs of numbers. The first number in the pair identifies the port from which the message exits. The second number in the pair designates the node address of the next device.

For this item:	Specify:
Communication Path	1,1,2,xxx.xxx.xxx.xxx,1,0 where: 1 is the DriveLogix backplane of DriveLogix1 1 is 1788-ENBT daughtercard in slot 1 2 is the EtherNet/IP port xxx.xxx.xxx.xxx. is the IP address of DriveLogix2 1 is the DriveLogix backplane of DriveLogix2 0 is the controller slot of DriveLogix2

Example 2: Producing and consuming tags

Produced data must be of DINT or REAL data type or an array or structure. You can use a user-defined structure to group BOOL, SINT, and INT data to be produced. You can produce a base, alias, or consumed tag.

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



This example shows DriveLogix1 as producing TagA and consuming TagB:

<input type="checkbox"/>	<input checked="" type="checkbox"/>	+timer_1			TIMER	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	+tagA			DINT	Decimal
<input type="checkbox"/>	<input type="checkbox"/>	tagB		controllerb:tagB	REAL	Float
<input checked="" type="checkbox"/>	<input type="checkbox"/>	*				

TagA

The 'New Tag' dialog box for TagA shows the following configuration: Name: tagA, Description: (empty), Tag Type: Produced (2 consumers), Data Type: DINT, Scope: quick_start(controller), Style: Decimal.

TagB

The 'New Tag' dialog box for TagB shows the following configuration: Name: tagB, Description: (empty), Tag Type: Consumed (1 consumer), Producer: controllerb, Remote Tag Name: tagB, Data Type: REAL, Style: Float.

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

Example 2: Total connections required by DriveLogix1

The following table calculates the connections used in this example.

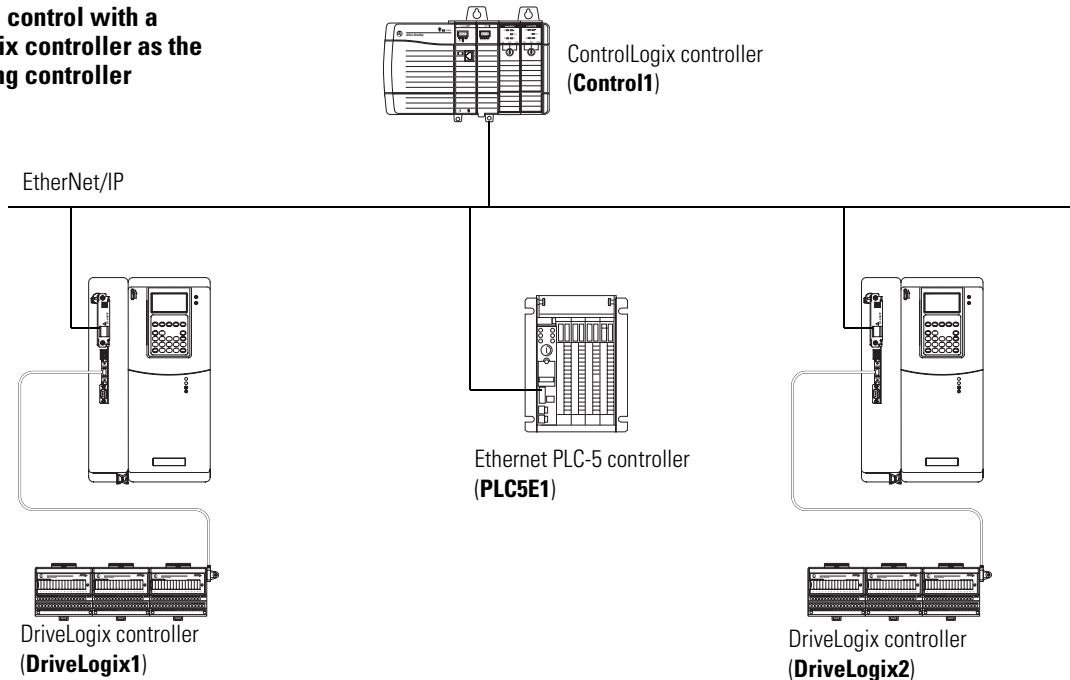
Connection:	Amount:
DriveLogix1 controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for each I/O module	3
DriveLogix1 controller to local 1788-ENBT	
	0
DriveLogix1 controller to remote 1788-ENBT	
	0
connected, cached MSG from DriveLogix1 to DriveLogix2	
	1
produced TagA	
produced from DriveLogix1 to DriveLogix2	1
other consumer (2 are configured)	1
consumed TagB	
	1
total connections used: 8	

If you configured the local I/O modules as rack-optimized, you would only need the DIN-rail connection to the I/O modules, reducing the above example by 3 connections.

Example 3: DriveLogix Controller to Other Devices

In the following example, one DriveLogix controller communicates with a Logix5550 controller and an Ethernet PLC-5 controller over EtherNet/IP.

Distributed control with a ControlLogix controller as the coordinating controller



Example 3: Sending MSG instructions

You configure a MSG instruction to a Logix5550 controller the same as you do for a DriveLogix controller. All Logix-based controllers follow the same MSG configuration requirements. See Example 2 above.

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

For MSG instructions originating from the DriveLogix controller to the Ethernet PLC-5 controller:

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

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

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

Type of MSG Instruction:	Example Source and Destination:	
PLC-5 writes to DriveLogix	source element	<i>N7:10</i>
	destination tag	<i>"array_1"</i>
PLC-5 reads from DriveLogix	source tag	<i>"array_1"</i>
	destination element	<i>N7:10</i>

Example 3: Total connections required by DriveLogix1

The following table calculates the connections used in this example.

Connection:	Amount:
DriveLogix1 controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for each I/O module	3
DriveLogix1 controller to local 1788-ENBT	0
connected, cached MSG from DriveLogix1 to Control1	1
connected, cached MSG from DriveLogix1 to PLC-5E1	1
total connections used:	6

If you configured the local I/O modules as rack-optimized, you would only need the DIN-rail connection to the I/O modules, reducing the above example by 3 connections.

Communicating with Devices on a ControlNet Link

Using This Chapter

For information about:	See page
Configuring your system for a ControlNet link	7-1
Configuring remote I/O	7-5
Sending messages	7-11
Producing and consuming data	7-17
Guidelines for configuring connections	7-21
Example 1: DriveLogix controller and remote I/O	7-22
Example 2: DriveLogix controller to DriveLogix controller	7-24
Example 3: DriveLogix controller to other devices	7-28

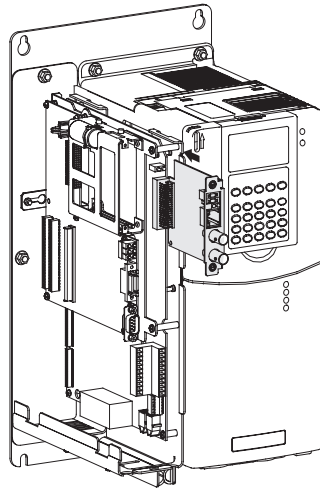
Configuring Your System for a ControlNet Link

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

- a workstation with an appropriate ControlNet communication daughtercard
- a 1788-CN_x communication daughtercard installed in the DriveLogix communication slot
- RSLinx software to configure the ControlNet communication driver
- RSLogix 5000 programming software to configure the 1788-CN_x communication daughtercard as part of the DriveLogix system
- RSNetWorx software to schedule the DriveLogix system on the network

Step 1: Configure the hardware

Before you can connect the DriveLogix system to the ControlNet network, you must configure the 1788-CN_x communication daughtercard and make sure it's properly installed in the DriveLogix controller. Refer to Access Procedures on page C-1 to understand how to gain access to the NetLinx daughtercard slot on the DriveLogix controller.



You'll need to configure the communication daughtercard slot number to 1 in the RSLogix 5000 programming software. The DriveLogix controller uses slot 0.

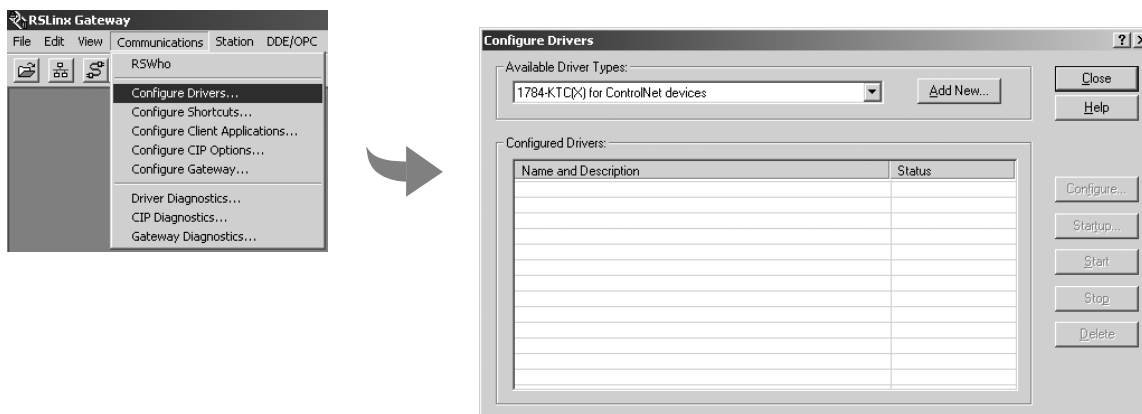
For more information about configuring a 1788-CN_x communication daughtercard, see:

For this card:	See this document:
1788-CNC, -CNCR	1788-IN002
1788-CNF, -CNFR	1788-IN005

Step 2: Configure the communication driver

Use RSLinx software to configure the ControlNet communication driver. Select the appropriate communication driver for the communication daughtercard in your workstation.

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



The installation instructions for the communications daughtercard should identify which communication driver to install.

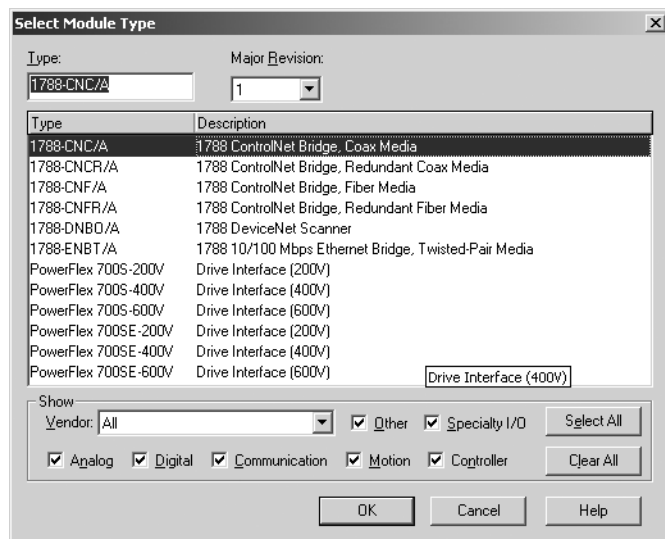
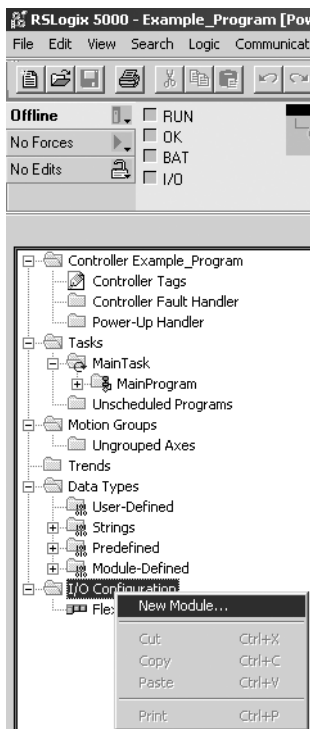
2. Specify the appropriate settings. For example:

If you are using this device:	Specify this information:
1784-KTCx card	memory address, which must match the switch setting on the card I/O base address, which must match the switch setting on the card ControlNet node address
1784-PCC card	ControlNet node address (MAC ID)
1784-PCIC card	ControlNet node address (MAC ID)

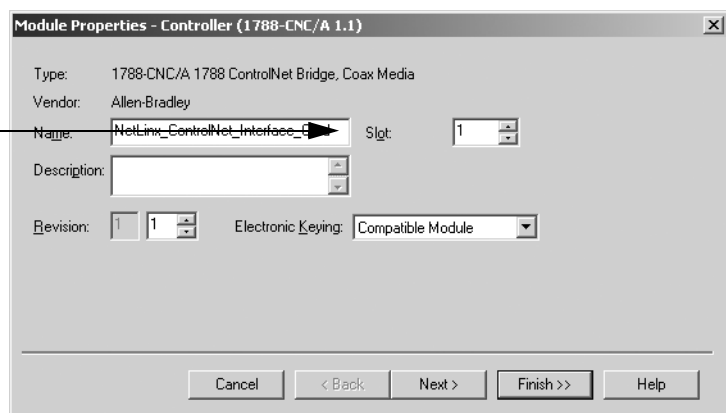
Step 3: Configure the daughtercard as part of the system

Use RSLogix 5000 programming software to map the 1788-CNx communication daughtercard as part of the DriveLogix system. In the Controller Organizer, add the communication daughtercard to the I/O Configuration folder.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.
2. Right-click to select New Module and add a 1788-CNx communication daughtercard.



4. Specify the slot number 1 for the communication daughtercard.



Complete your system configuration and develop your program logic. Then download the project to the DriveLogix controller.

Configuring Remote I/O

The DriveLogix controller supports remote I/O over a ControlNet link. Configuring I/O in a remote chassis is similar to configuring local I/O. The difference is that you must also configure the communication daughtercard (1788-CN_x) in the local chassis and the communication module in the remote chassis.

To configure a remote I/O module:

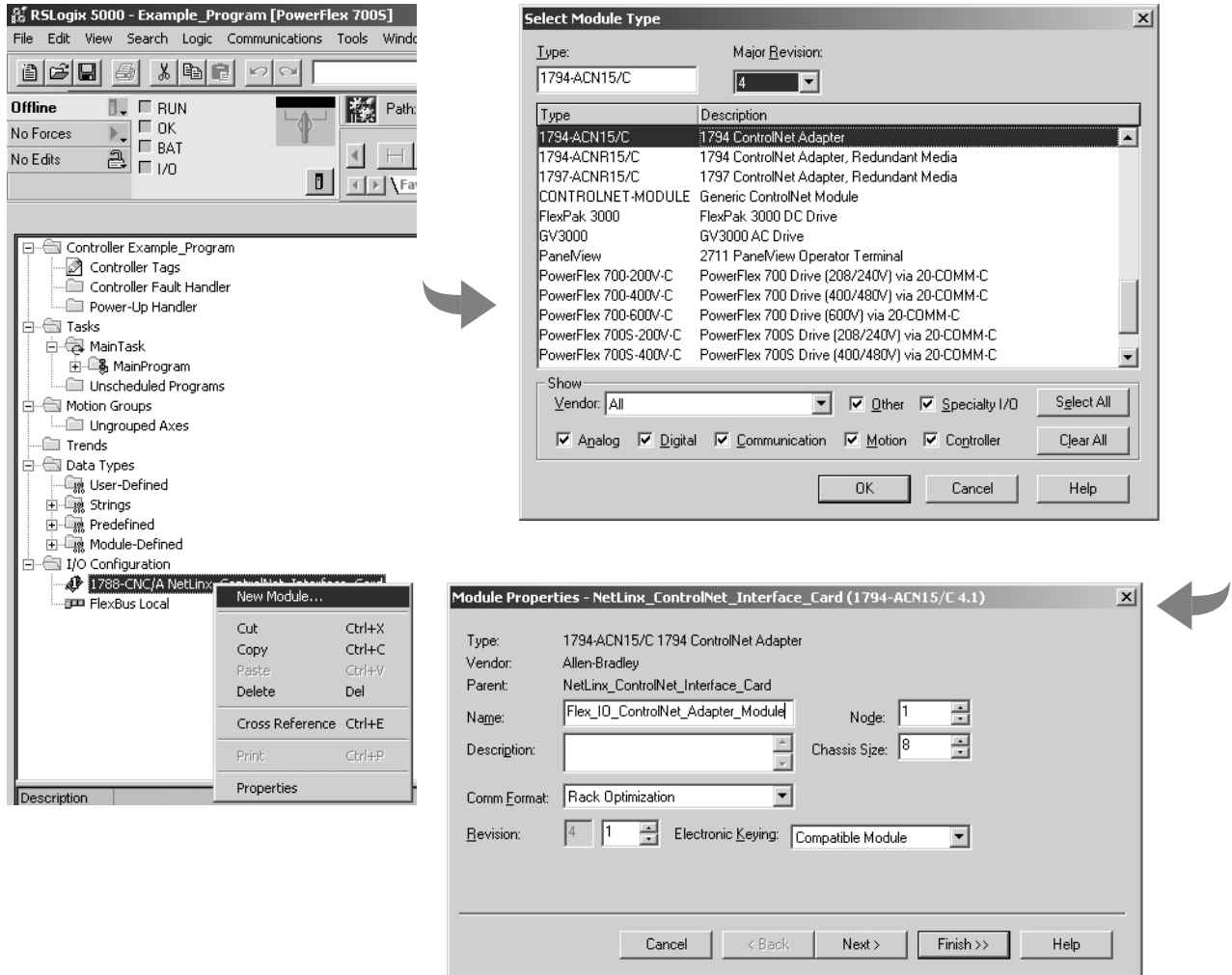
1. In the Controller Organizer, select the I/O Configuration Folder. Add and configure a 1788-CN_x communication daughtercard. This is the local communication daughtercard.

The image shows a sequence of three screenshots from the RSLogix 5000 software interface:

- Left Screenshot:** The Controller Organizer tree view is shown. The 'I/O Configuration' folder is selected, and a context menu is open with 'New Module...' highlighted.
- Middle Screenshot:** The 'Select Module Type' dialog box is displayed. The 'Type' field is set to '1788-CN/A'. A list of module types is shown, with '1788-CN/A' selected. The 'Major Revision' is set to '1'. The 'Drive Interface (400V)' option is selected in the bottom right.
- Right Screenshot:** The 'Module Properties - Controller (1788-CN/A 1.1)' dialog box is shown. The 'Name' field is 'NetLink_ControlNet_Interface_Card' and the 'Slot' is '1'. The 'Revision' is '1' and 'Electronic Keying' is 'Compatible Module'.

3. Specify the slot number (1 or 2) where you installed the communication daughtercard.

- In the Controller Organizer, select the local 1788-CNx communication daughtercard you just added. Add and configure the remote communication module (1794-ACN15 in this example)



- Add and configure the remote I/O modules on the remote communication module you just added.

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

Configure I/O modules for the remote communication module by adding them to the remote communication module (i.e., right-click the 1794-ACN15 module and select New Module). Configure the remote I/O modules the same way you do local I/O modules.

Accessing remote I/O

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

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

where:

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

EXAMPLE

Device:	Example Tag Names (automatically created by the software):
remote adapter "FLEX_adapter"	FLEX_adapter:I FLEX_adapter:I.SlotStatusBits FLEX_adapter:I.Data FLEX_adapter:O FLEX_adapter:O.Data
remote "input1" in slot 0	FLEX_adapter:O:C
direct connection	FLEX_adapter:O:C.Config FLEX_adapter:O:C.Filter0_00_11 FLEX_adapter:O:C.Filter1_00_11 FLEX_adapter:O:C.Filter2_00_11 FLEX_adapter:O:C.Filter3_12_15 FLEX_adapter:O:C.Filter4_12_15 FLEX_adapter:O:C.Filter5_12_15 FLEX_adapter:O:C.ResetCounter FLEX_adapter:O:C.DisableFilter FLEX_adapter:O:I FLEX_adapter:O:I.Fault FLEX_adapter:O:I.Data FLEX_adapter:O:I.Counter

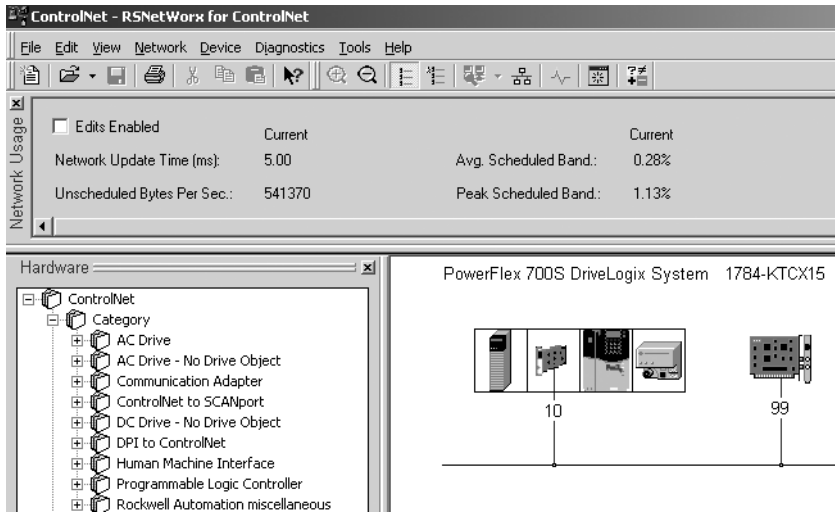
Device:	Example Tag Names (automatically created by the software):
remote "output1" in slot 1	FLEX_adapter:1:C
direct connection	FLEX_adapter:1:C.SSDate FLEX_adapter:1:I FLEX_adapter:1:I.Fault FLEX_adapter:1:O FLEX_adapter:1:O.Data
remote "input2" in slot 2	FLEX_adapter:2:C
rack-optimized connection	FLEX_adapter:2:C.Config FLEX_adapter:2:C.Filter0_00_11 FLEX_adapter:2:C.Filter1_00_11 FLEX_adapter:2:C.Filter2_00_11 FLEX_adapter:2:C.Filter3_12_15 FLEX_adapter:2:C.Filter4_12_15 FLEX_adapter:2:C.Filter5_12_15 FLEX_adapter:2:C.ResetCounter FLEX_adapter:2:C.DisableFilter
These tags are created as aliases into the FLEX_adapter:I tag	FLEX_adapter:2:I
remote "output2" in slot 3	FLEX_adapter:3:C
rack-optimized connection	FLEX_adapter:3:C.SSDate FLEX_adapter:3:O
These tags are created as aliases into the FLEX_adapter:O tag	

For examples of local I/O tags, see chapter 4 "Placing and Configuring Local I/O".

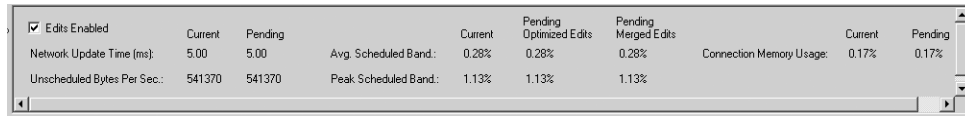
Scheduling the ControlNet Network

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

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



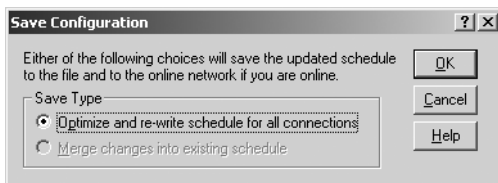
2. Specify the network update time (NUT)



The default NUT is 5ms.

The NUT you specify must be lower than or equal to the lowest RPI in your ControlNet network. The RPI numbers for the local and extended-local

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



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

Sending Messages

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

This device:	Support this many unconnected messages:	Support this many connected messages:
1756-CNB or 1756-CNBR module (for a Logix5550 controller)	20	64
1788-CNx daughtercard (for a DriveLogix controller)	5	32 with a maximum of 9 scheduled
ControlNet PLC-5 controller	32	128

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

This type of message:	And this communication method:	Uses a connection:
CIP data table read or write		X
PLC2, PLC3, PLC5, or SLC (all types)	CIP	
	CIP with Source ID	
	DH+	X
CIP generic	CIP	Optional ⁽¹⁾
block-transfer read or write		X

⁽¹⁾ You can connect CIP generic messages, but for most applications, we recommend you leave CIP generic messages unconnected.

Connected messages are unscheduled connections on ControlNet.

If a MSG instruction uses a connection, you have the option to leave the connection open (cache) or close the connection when the message is done transmitting.

If you:	Then:
Cache the connection	The connection stays open after the MSG instruction is done. This optimizes execution time. Opening a connection each time the message executes increases execution time.
Do not cache the connection	The connection closes after the MSG instruction is done. This frees up that connection for other uses.

The controller has the following limits on the number of connections that you can cache:

If you have this software and firmware revision:	Then you can cache:
11.x or earlier	<ul style="list-style-type: none"> • block transfer messages for up to 16 connections • other types of messages for up to 16 connections
12.x or later	up to 32 connections

Communicating with another Logix-based controller

All Logix-based controllers can use MSG instructions to communicate with each other. The following examples show how to use tags in MSG instructions between Logix-based controllers.

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

The source and destination tags:

- must be controller-scoped tags.
- can be of any data type, except for AXIS, MESSAGE, or MOTION_GROUP.

Communicating with other controllers over ControlNet

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

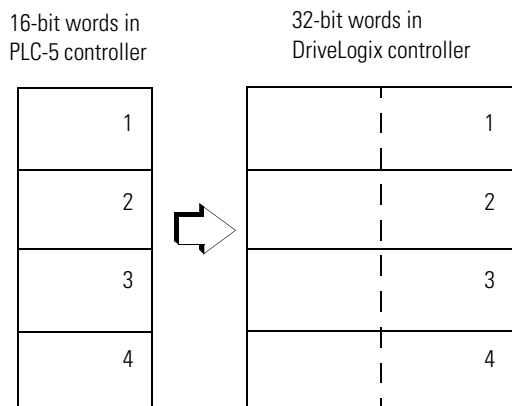
For MSG instructions originating from a DriveLogix controller to a PLC or SLC controller:

Type of MSG Instruction:	Supported Source File Types:	Supported Destination File Types:
DriveLogix writes to PLC-5 or SLC	In the DriveLogix controller, specify the source data type based on the destination device:	Specify the destination file type based on the destination device:
	PLC-5: SINT, INT, DINT, or REAL	PLC-5 typed write: S, B, N, or F
	SLC: INT	PLC-5 word-range write: S, B, N, F, I, O, A, or D
		SLC: B or N
	Example source element: <i>array_1</i>	Example destination tag: <i>N7:10</i>

Type of MSG Instruction:	Supported Source File Types:	Supported Destination File Types:
DriveLogix writes to PLC-2	<p>In the DriveLogix controller, select one of these data types:</p> <p>SINT, INT, DINT, or REAL</p> <p>Example source element: <i>array_1</i></p>	<p>Use the PLC-2 compatibility file.</p> <p>Example destination tag: <i>010</i></p>
DriveLogix reads from PLC-5 or SLC	<p>Specify the destination file type based on the destination device:</p> <p>PLC-5 typed read: S, B, N, or F</p> <p>PLC-5 word-range read: S, B, N, F, I, O, A, or D</p> <p>SLC: B or N</p> <p>Example source element: <i>N7:10</i></p>	<p>In the DriveLogix controller, specify the destination data type based on the destination device:</p> <p>PLC-5: SINT, INT, DINT, or REAL</p> <p>SLC: INT</p> <p>Example destination tag: <i>array_1</i></p>
DriveLogix reads from PLC-2	<p>Use the PLC-2 compatibility file.</p> <p>Example source element: <i>010</i></p>	<p>In the DriveLogix controller, select one of these data types:</p> <p>SINT, INT, DINT, or REAL</p> <p>Example destination tag: <i>array_1</i></p>

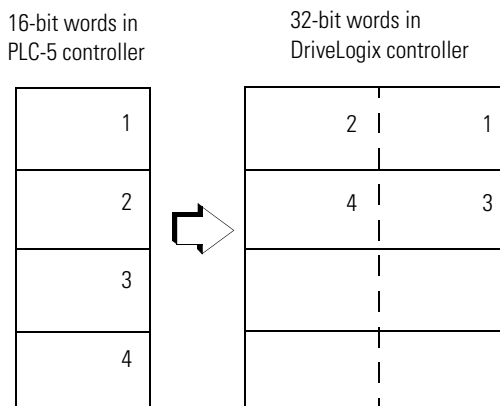
The DriveLogix controller can send typed or word-range commands to PLC-5 controllers. These commands read and write data differently. The following diagrams show how the typed and word-range commands differ.

Typed read command



The typed commands maintain data structure and value.

Word-range read command



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

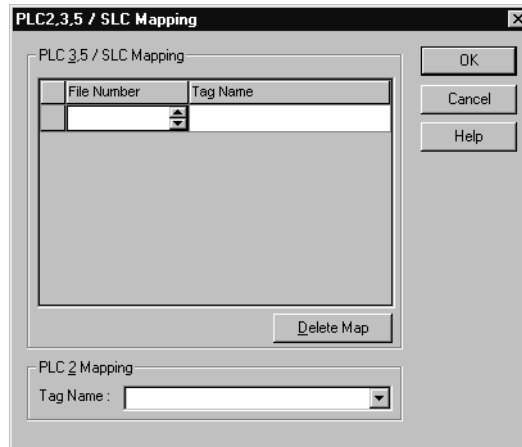
The DriveLogix controller can process messages initiated from PLC or SLC controllers. These messages use data table addresses. In order for these controllers to access tags within the DriveLogix controller, you map tags to data table addresses.

Mapping 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 controllers.

To map addresses:

1. From the Logic menu, select Map PLC/SLC Messages.



2. Specify this information:

For:	In this field:	Specify:	For example:
PLC-3, PLC-5, and SLC controllers	File Number	Type the file number of the data table in the PLC/SLC controller.	10
	Tag Name	Type the array tag name the local controller uses to refer to the PLC/SLC data table address. The tag must be an integer array (SINT, INT, or DINT) that is large enough for the message data.	array_1
PLC-2 controllers	Tag Name	Type the tag name to be the PLC-2 compatibility file.	200

TIP

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

The following table shows example source and destination tags and elements for different controller combinations.

Type of MSG Instruction:	Example Source and Destination:
PLC-5 writes to DriveLogix	source element <i>N7:10</i>
SLC writes to DriveLogix	destination tag <i>"array_1"</i>
SLC 5/05	The PLC-5, PLC-3, and SLC controllers 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 controller. Place the DriveLogix tag name in double quotes ("). 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.
SLC 5/04 OS402 and above	
SLC 5/03 OS303 and above	

Type of MSG Instruction:	Example Source and Destination:	
PLC-2 writes to DriveLogix	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 DriveLogix	source tag	<i>"array_1"</i>
SLC reads from DriveLogix	destination element	<i>N7:10</i>
SLC 5/05 SLC 5/04 OS402 and above	The PLC-5, PLC-3, and SLC controllers 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 controller. Place the DriveLogix tag name in double quotes ("").	
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 DriveLogix	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 DriveLogix 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 controllers, SLC 5/04 controllers (OS402 and above), and SLC 5/03 controllers (OS303 and above) support logical ASCII addressing and support PLC/SLC mapping (see the examples above). For all other SLC or MicroLogix1000 controllers, you must map a PLC-2 compatibility file (see the PLC-2 examples above).

Producing and Consuming Data

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

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

Tag type:	Description:	Specify:
produced	These are tags that the controller produced for other controllers to consume.	<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

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

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

Maximum number of produced and consumed tags

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

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

Size limit of a produced or consumed tag

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

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

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

- Reduce the Network Update Time (NUT). At a faster NUT, less connections have to share an update slot.
- Increase the Requested Packet Interval (RPI) of all connections. At a higher RPI, connections can take turns sending data during an update slot.

- For a ControlNet bridge module (CNB) in a remote chassis, select the most efficient communication format for that chassis:.

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

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

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

Producing a tag

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

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

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

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

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

Consuming a tag

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

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

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

All consumed tags are automatically controller-scope.

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

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

Guidelines for Configuring Connections

The 1788-CN_x communication daughtercard supports 9 scheduled connections. How you configure these connections determines how many devices the daughtercard can support.

The NUT and RPI also play a part in determining how many connections a 1788-CN_x can support in a given application, assuming the RPIs will be the same for all connections. You must also make sure that you do not exceed the maximum number of bytes per NUT.

- With the NUT = 5ms, the limit is 3 connections
- With the NUT = 10ms, the limit is 7 connections
- With the NUT \geq 20ms, the limit is 9 connections

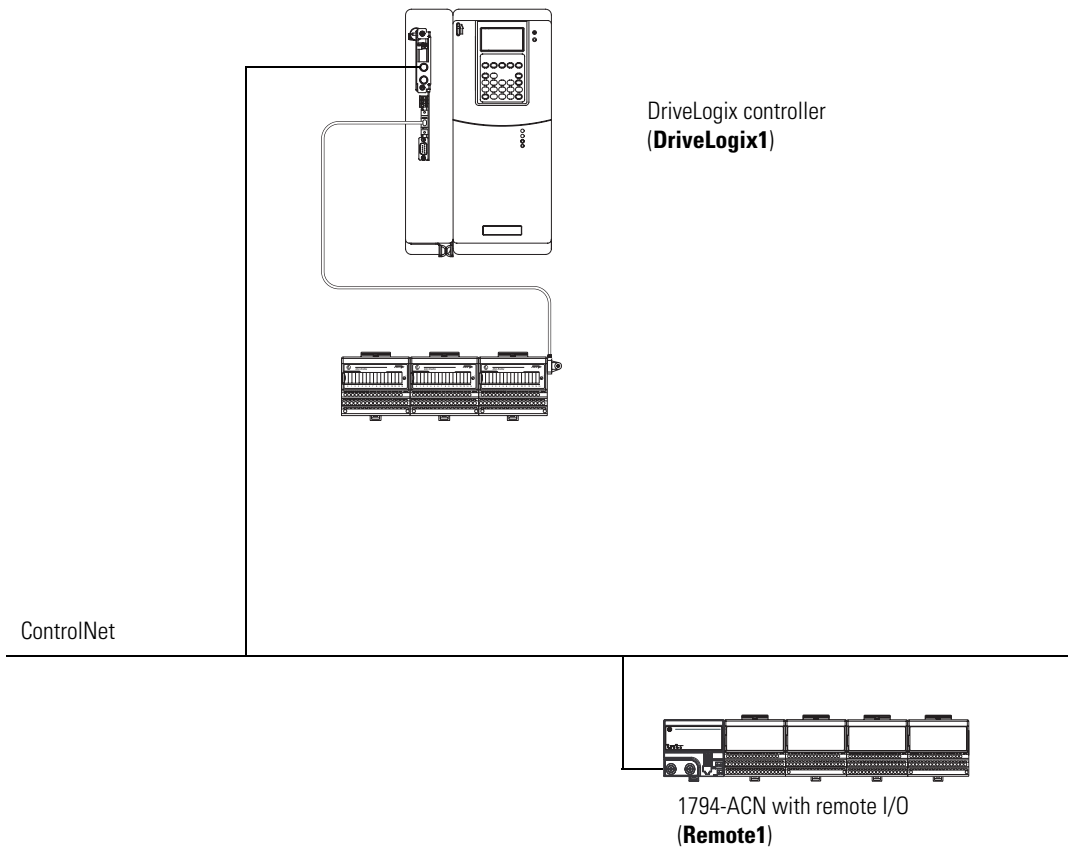
Determining the API

The API (actual packets per interval) is related to the RPI for the connection and the NUT of the network. Use this table to select the API to enter in the above worksheet:

If:	Enter this value for the API:
$RPI \geq NUT$ and $RPI < 2*NUT$	NUT
$RPI \geq 2*NUT$ and $RPI < 4*NUT$	2*NUT
$RPI \geq 4*NUT$ and $RPI < 8*NUT$	4*NUT
$RPI \geq 8*NUT$ and $RPI < 16*NUT$	8*NUT
$RPI \geq 16*NUT$ and $RPI < 32*NUT$	16*NUT
$RPI \geq 32*NUT$ and $RPI < 64*NUT$	32*NUT
$RPI \geq 64*NUT$ and $RPI < 128*NUT$	64*NUT
$RPI \geq 128*NUT$	128*NUT

Example 1: DriveLogix Controller and Remote I/O

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



Example 1: Controlling remote devices

This example has DriveLogix1 controlling the I/O connected to the remote 1794-ACN15 module. The data the DriveLogix controller receives from the remote I/O modules depends on how you configure the remote I/O modules. You can configure each module as a direct connection or as rack optimized. One chassis can have a combination of some modules configured as a direct connection and others as rack optimized.

Example 1: Total connections required by DriveLogix1

The following table calculates the connections used in this example.

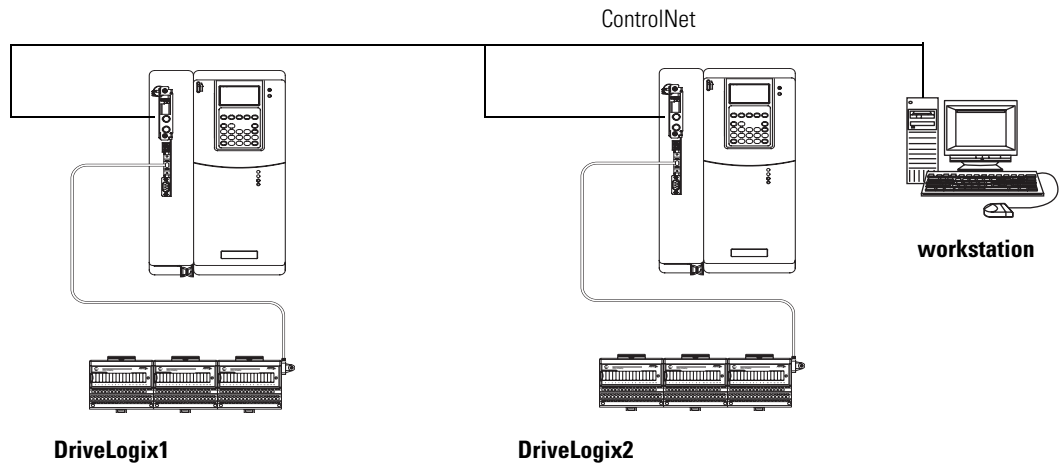
Connection:	Amount:
DriveLogix1 controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for each I/O module	3
DriveLogix1 controller to remote 1794-ACNR15	1
DriveLogix1 to 4 remote I/O modules (through 1794-ACNR15)	4
all I/O modules configured as direct connection	
no connection to the 1794-ACNR15	
total connections used: 9	

If you configured the remote I/O modules as rack-optimized, you would only need a rack-optimized connection to the 1794-ACNR15, reducing the above example by 3 connections.

Example 2: DriveLogix Controller to DriveLogix Controller

In the following example, one DriveLogix controller communicates with another DriveLogix controller over ControlNet. Each DriveLogix controller has its own local I/O

Distributed control



Example 2: Sending a MSG instruction

To send a MSG from DriveLogix1 to DriveLogix2:

1. For DriveLogix1, create a controller-scoped tag and select the MESSAGE data type.
2. Enter a MSG instruction.

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



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

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

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

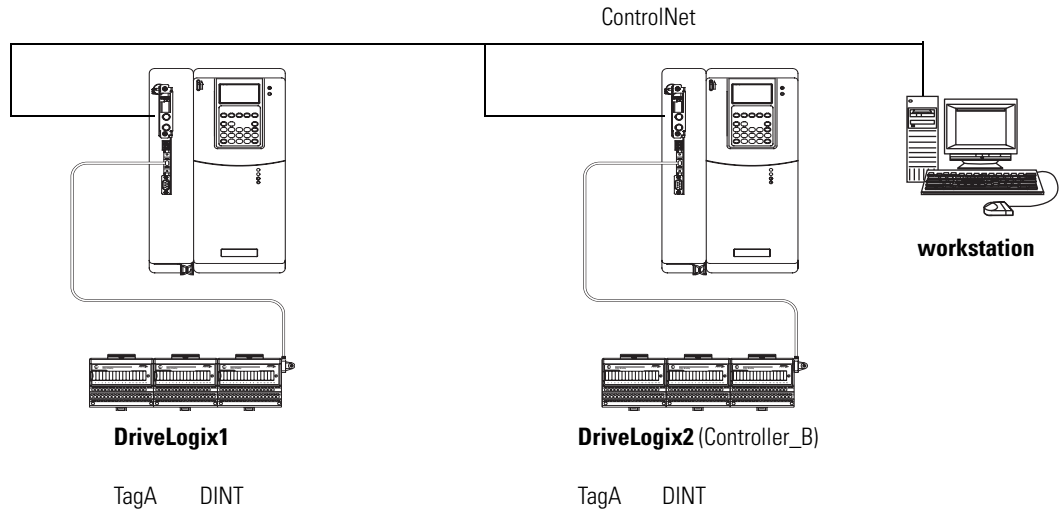
A communication path requires pairs of numbers. The first number in the pair identifies the port from which the message exits. The second number in the pair designates the node address of the next device.

For this item:	Specify:
Communication Path	1,1,2,27,1,0
	where:
	1 is the DriveLogix backplane of DriveLogix1
	1 is 1788-CNC daughtercard in slot 1
	2 is the ControlNet port
	27 is the ControlNet node of DriveLogix2
	1 is the DriveLogix backplane of DriveLogix2
	0 is the controller slot of DriveLogix2

Example 2: Producing and consuming tags

Produced data must be of DINT or REAL data type or an array or structure. You can use a user-defined structure to group BOOL, SINT, and INT data to be produced. You can produce a base, alias, or consumed tag.

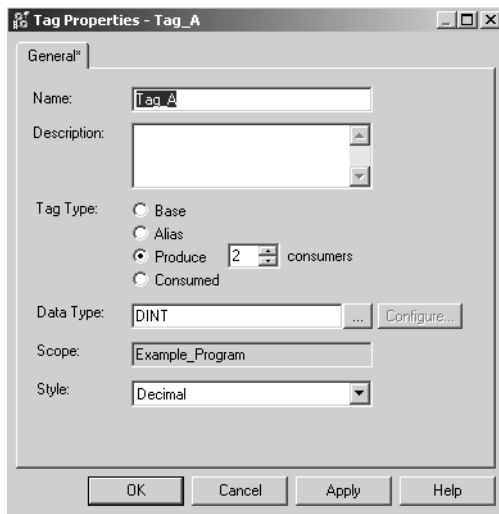
The consumed tag must have the same data type as the produced tag in the originating controller. The controller performs type checking to ensure proper data is being received.



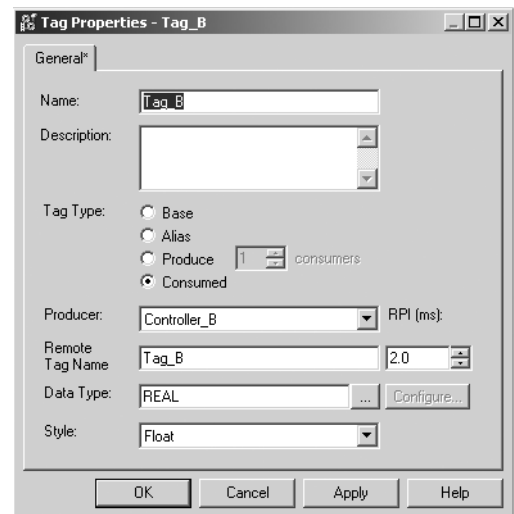
This example shows Flex1 as producing TagA and consuming TagB:

<input type="checkbox"/>	+tagA		DINT	Decimal
<input type="checkbox"/>	+tagB		DINT	Decimal
<input checked="" type="checkbox"/>	*			

TagA



TagB



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

Example 2: Total connections required by DriveLogix1

The following table calculates the connections used in this example.

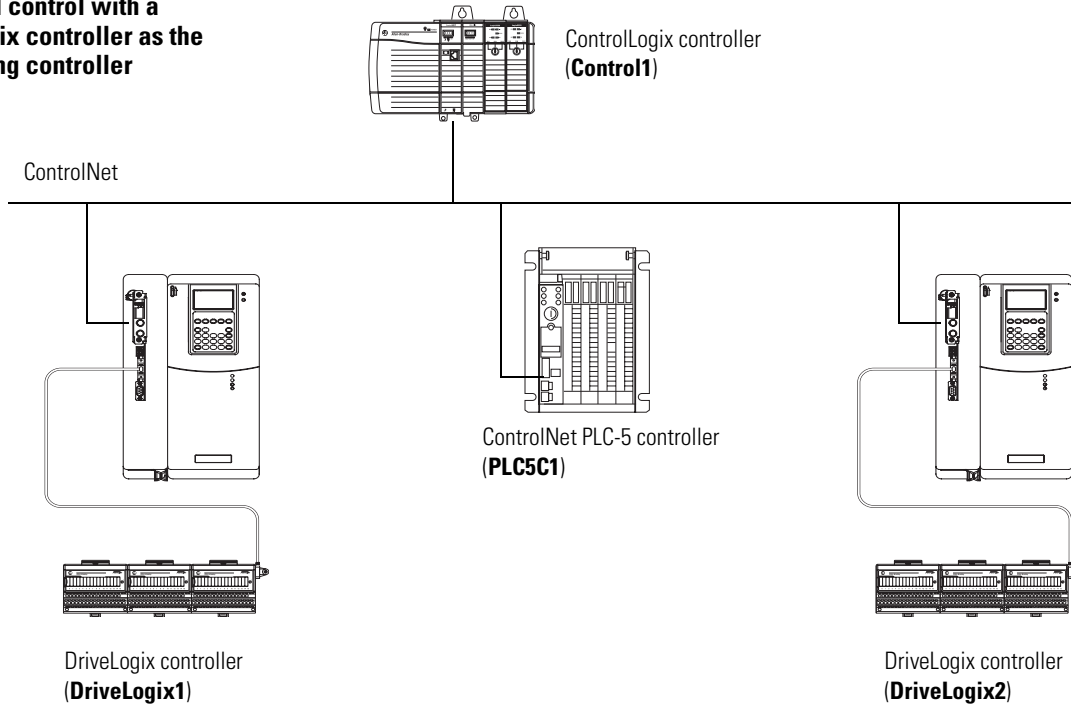
Connection:	Amount:
DriveLogix1 controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for each I/O module	3
DriveLogix1 controller to local 1788-CNC	0
DriveLogix1 controller to remote 1788-CNC	0
connected, cached MSG from DriveLogix1 to DriveLogix2	1
produced TagA	
produced from DriveLogix1 to DriveLogix2	1
other consumer (2 are configured)	1
consumed TagB	1
total connections used:	8

If you configured the local I/O modules as rack-optimized, you would only need the DIN-rail connection to the I/O modules, reducing the above example by 3 connections.

Example 3: DriveLogix Controller to Other Devices

In the following example, one DriveLogix controller communicates with a Logix5550 controller and a ControlNet PLC-5 controller over ControlNet.

Distributed control with a ControlLogix controller as the coordinating controller



Example 3: Sending MSG instructions

You configure a MSG instruction to a Logix5550 controller the same as you do for a DriveLogix controller. All Logix-based controllers follow the same MSG configuration requirements. See Example 2 above.

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

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

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

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

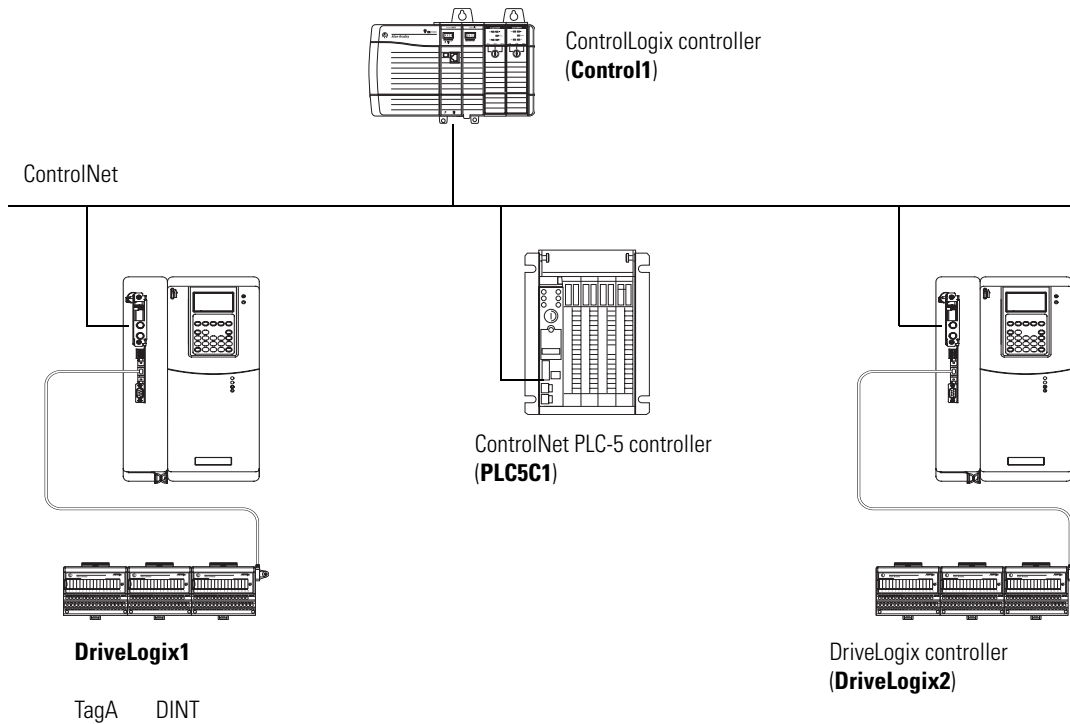
The PLC-5 controller supports logical ASCII addressing so you do not have to map a compatibility file for MSG instructions initiated by a PLC-5 controller. Place the DriveLogix tag name in double quotes (“”).

Type of MSG Instruction:	Example Source and Destination:	
PLC-5 writes to DriveLogix	source element	<i>N7:10</i>
	destination tag	<i>"array_1"</i>
PLC-5 reads from DriveLogix	source tag	<i>"array_1"</i>
	destination element	<i>N7:10</i>

Example 3: Producing and consuming tags

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

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



Producing a tag to a ControlNet PLC-5 controller

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

1. Determine the type of data to produce?

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

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

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

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

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

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

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

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

Consuming a tag from a ControlNet PLC-5 controller

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

- In RSNetWorx software, open the ControlNet configuration of the ControlNet PLC-5 controller, insert a Send Scheduled Message.
- In RSLogix 5000 software, add the ControlNet PLC-5 controller to the Controller Organizer.
- Create a user-defined data type that contains these members:

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

4. Create a consumed tag with the following properties:

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

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

Example 3: Total connections required by DriveLogix1

The following table calculates the connections used in this example.

Connection:	Amount:
DriveLogix1 controller to 3 local I/O modules	
rack-optimized connection for the DIN rail	1
direct connection for each I/O module	3
DriveLogix1 controller to local 1788-CNC	0
DriveLogix1 controller to remote 1756-CNB	1
DriveLogix1 controller to remote ControlNet PLC-5	1
connected, cached MSG from DriveLogix1 to Control1	1
connected, cached MSG from DriveLogix1 to PLC5C1	1
Produced TagA	
produced from DriveLogix1 to DriveLogix2	1
consumed by PLC5C1	1
Consumed TagB from DriveLogix2	1
Consumed INT from PLC5C1	1
total connections used:	12

If you configured the local I/O modules as rack-optimized, you would only need the DIN-rail connection to the I/O modules, reducing the above example by 3 connections.

You can configure the 1756-CNB module to use no connection. This is useful if you configure all direct connections to their associated I/O modules and do not need a rack-optimized connection.

Notes:

Communicating with Devices on a DeviceNet Link

Using This Chapter

For information about:	See page
Configuring your system for a DeviceNet link	8-1
Placing DeviceNet devices	8-5
Accessing DeviceNet devices	8-6
Placing the communication card in Run mode	8-9
Example 1: DriveLogix controller and DeviceNet devices	8-10
Example 2: Using a 1788-CN2DN Linking Device	8-11

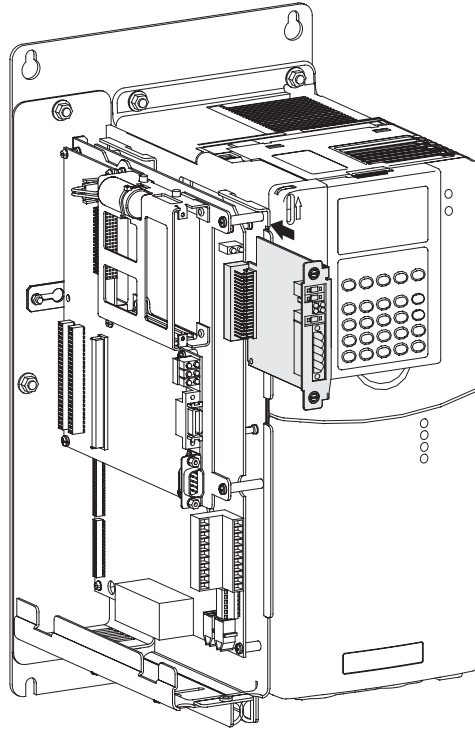
Configuring Your System for a DeviceNet Link

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

- a 1788-DNBO DeviceNet communication daughtercard.
- RSLogix 5000 programming software, version 10 or later, to configure the 1788-DNBO card as part of the DriveLogix system
- RSNetWorx for DeviceNet software to configure the 1788-DNBO card on the DeviceNet network

Step 1: Install the hardware

Before you can connect the DriveLogix system to the DeviceNet network, you must configure the 1788-DNBO communication card and make sure it's properly installed in the DriveLogix controller. Refer to Access Procedures on page C-1 to understand how to gain access to the NetLinX daughtercard slot on the DriveLogix controller.



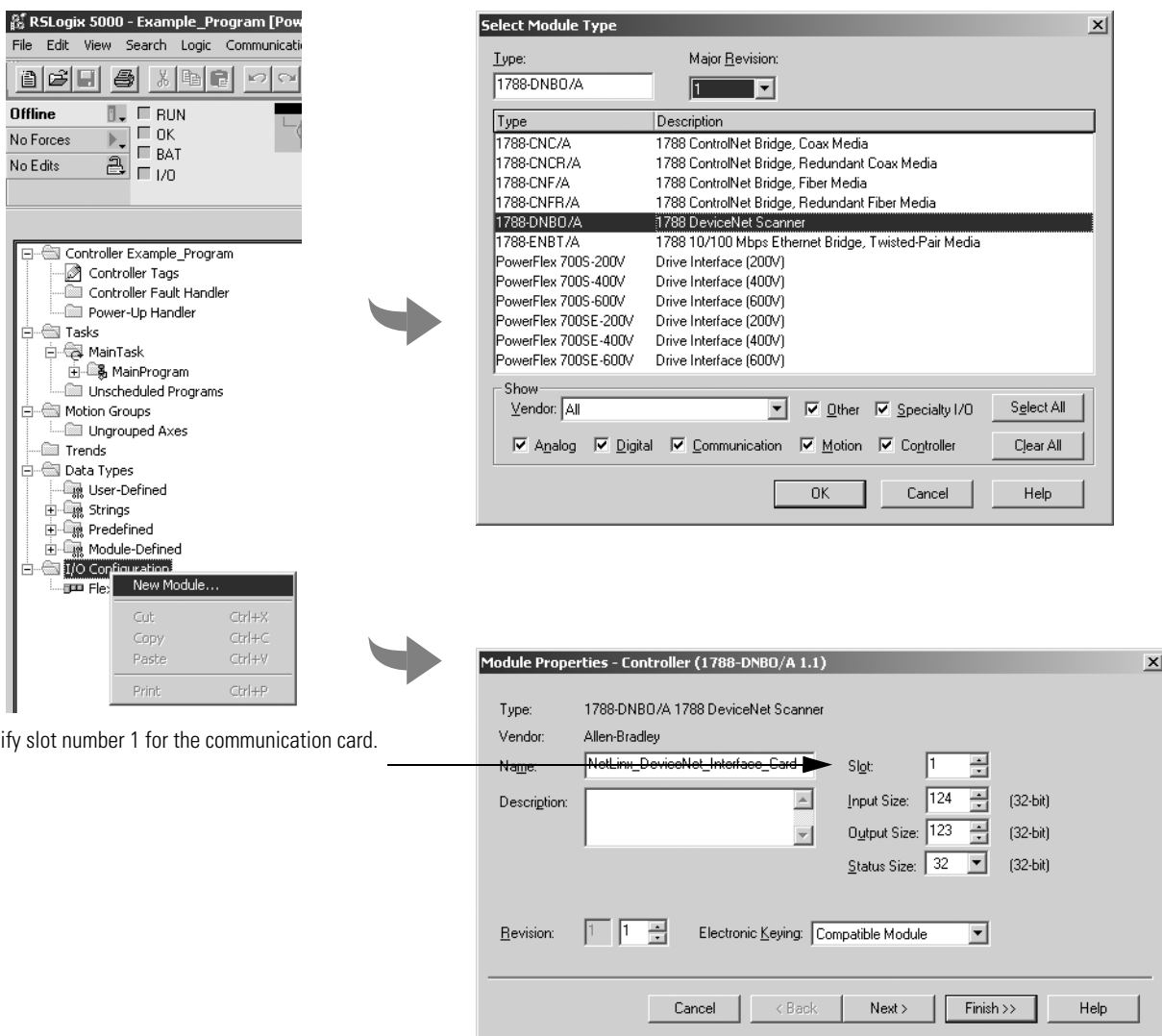
You'll need to configure the communication daughtercard slot number to 1 in the RSLogix 5000 programming software. The controller uses slot 0.

For more information about configuring a 1788-DNBO card, see the *DeviceNet Daughtercard Installation Instructions*, publication 1788-IN053.

Step 2: Configure the daughtercard as part of the system

Use RSLogix 5000 programming software to map the 1788-DNBO card as part of the DriveLogix system. In the Controller Organizer, add the card to the I/O Configuration folder.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.
2. Right-click to select New Module and add a 1788-DNBO communication daughtercard.



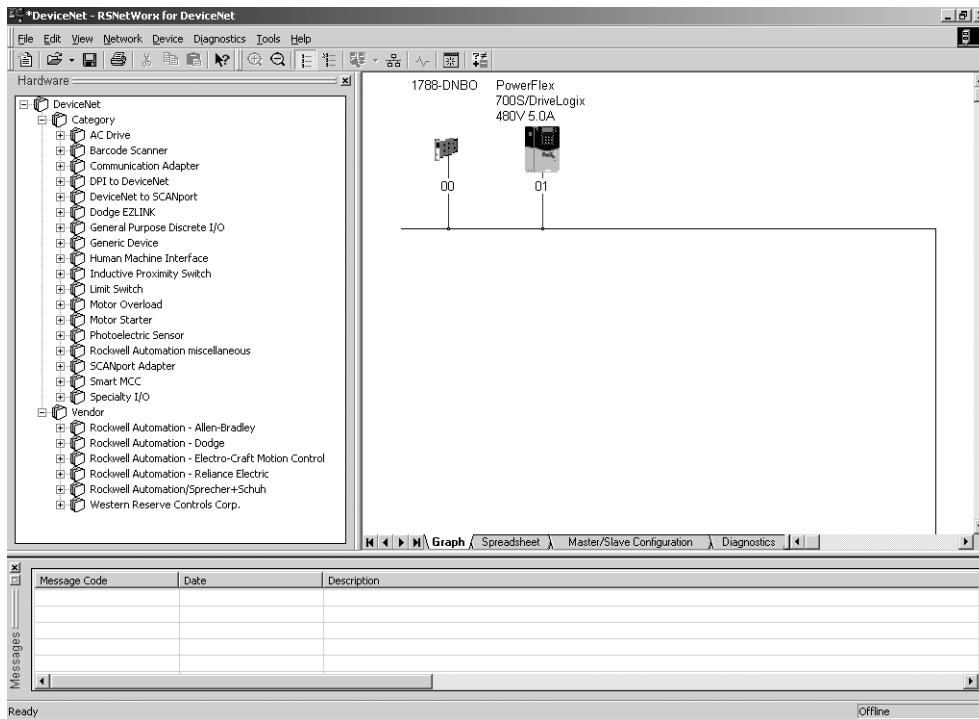
4. Specify slot number 1 for the communication card.

Complete your system configuration and develop your program logic. Then download the project to the controller.

Step 3: Define the DeviceNet scanlist

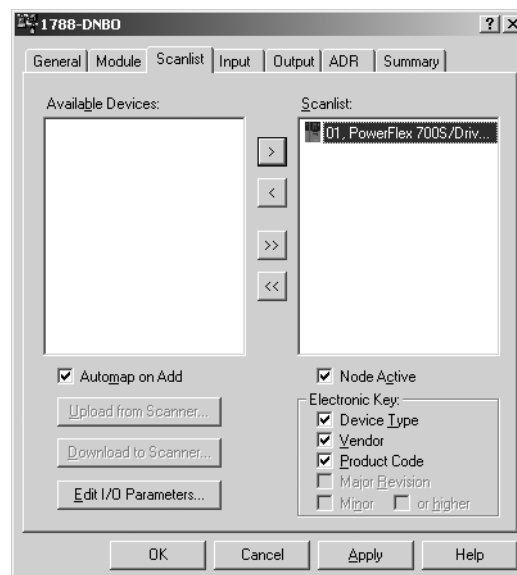
Use RSNetWorx for DeviceNet to create a scanlist of the DeviceNet devices connected to the 1788-DNBO card. If the DriveLogix controller is powered-up while connected to the 1788-DNBO card, the controller project does not have to be downloaded from RSLogix 5000 programming software to the controller and the controller must be in Program or Remote Program mode.

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



2. Assign a node address to each device.

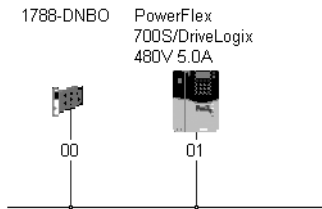
Configure each device.



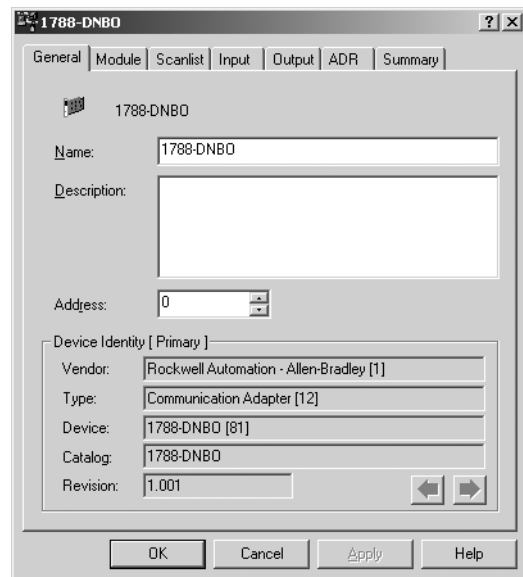
Placing DeviceNet Devices

Use RSNetWorx for DeviceNet to configure a scan list for the 1788-DNBO card. The scanlist and the associated input/output data tables set up the data you want the controller to send to and receive from the card.

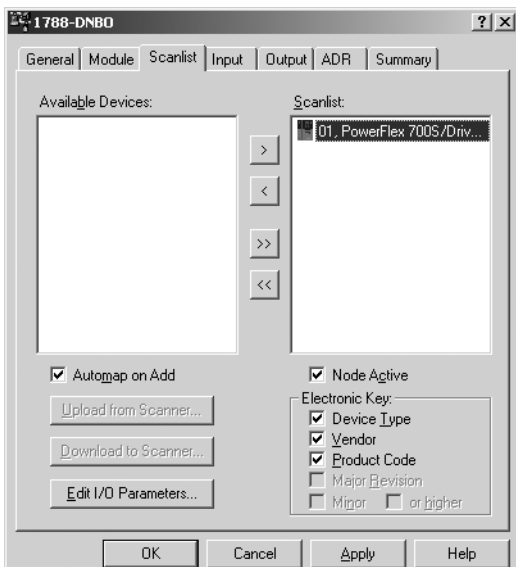
1. In RSNetWorx software, go online, enable edits, and survey the network.



2. Double-click the 1788-DNBO card and use the Module tab to configure the card. Upload the network information when prompted.



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

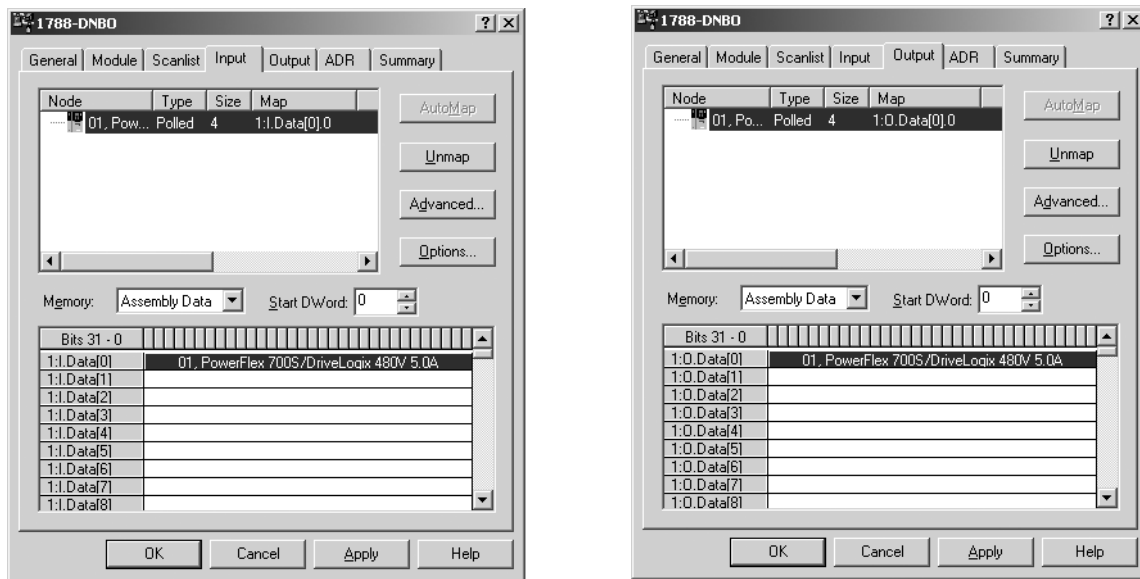


Defining the data blocks

How you configure the DeviceNet devices determines how many words you use per device. The 1788-DNBO card supports a maximum of:

- 124 32-bit words of input data
- 123 32-bit words of output data
- 32 32-bit words of status data

Once you define the scanlist, you define how the data for the devices maps into the input, output, and status data blocks. Use the Input, Output, or Status tabs to define the associated data block



Use the AutoMap button to simplify defining the data block for each DeviceNet device. The above screens show how many 32-bit words are mapped for the devices on this example network. These words map directly into the array tags that the software creates for the 1788-DNBO card.

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

Accessing DeviceNet Devices

I/O information is presented as a structure of multiple fields, which depend on the specific features of the I/O module. The name of the structure is based on the location of the I/O module in the system. Each I/O tag is

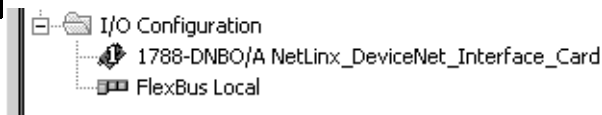
automatically created when you configure the I/O module through the programming software. Each tag name follows this format:

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

where:

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

EXAMPLE



← The 1788-DNBO card in this example is in named "DeviceNet_Interface_Card".

The data for the card is configured as a rack-optimized connection.

The rack-optimized connection creates a DINT element for mapped data for each DeviceNet module connected to the card “dnet.” The array dnet.I.Data contains the possible input elements; the dnet.O.Data contains the possible output elements.

Tag Name	Value	Force Mask	Style
Local:I	{...}	{...}	
Local:O	{...}	{...}	
NetLinx_DeviceNet_Interface_Card:I	{...}	{...}	
NetLinx_DeviceNet_Interface_Card:I.StatusRegister	{...}	{...}	
NetLinx_DeviceNet_Interface_Card:I.Data	{...}	{...}	Decimal
NetLinx_DeviceNet_Interface_Card:O	{...}	{...}	
NetLinx_DeviceNet_Interface_Card:O.CommandRegister	{...}	{...}	
NetLinx_DeviceNet_Interface_Card:O.Data	{...}	{...}	Decimal
NetLinx_DeviceNet_Interface_Card:S	{...}	{...}	
NetLinx_DeviceNet_Interface_Card:S.ScanCounter	2#0000_0000_...		Binary
NetLinx_DeviceNet_Interface_Card:S.DeviceFailureRegister	{...}	{...}	Binary
NetLinx_DeviceNet_Interface_Card:S.AutoverifyFailureRegister	{...}	{...}	Binary
NetLinx_DeviceNet_Interface_Card:S.DeviceIdleRegister	{...}	{...}	Binary
NetLinx_DeviceNet_Interface_Card:S.ActiveNodeRegister	{...}	{...}	Binary
NetLinx_DeviceNet_Interface_Card:S.StatusDisplay	{...}	{...}	Binary
NetLinx_DeviceNet_Interface_Card:S.ScannerAddress	16#00		Hex
NetLinx_DeviceNet_Interface_Card:S.ScannerStatus	16#00		Hex
NetLinx_DeviceNet_Interface_Card:S.ScrollingDeviceAddress	16#00		Hex
NetLinx_DeviceNet_Interface_Card:S.ScrollingDeviceStatus	16#00		Hex
NetLinx_DeviceNet_Interface_Card:S.DeviceStatus	{...}	{...}	Hex

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

Placing the Communication Card in Run Mode

To place the 1788-DNBO daughtercard in Run mode, your program logic needs to set the CommandRegister.Run bit in the output word for the 1788-DNBO card.

Set this bit →

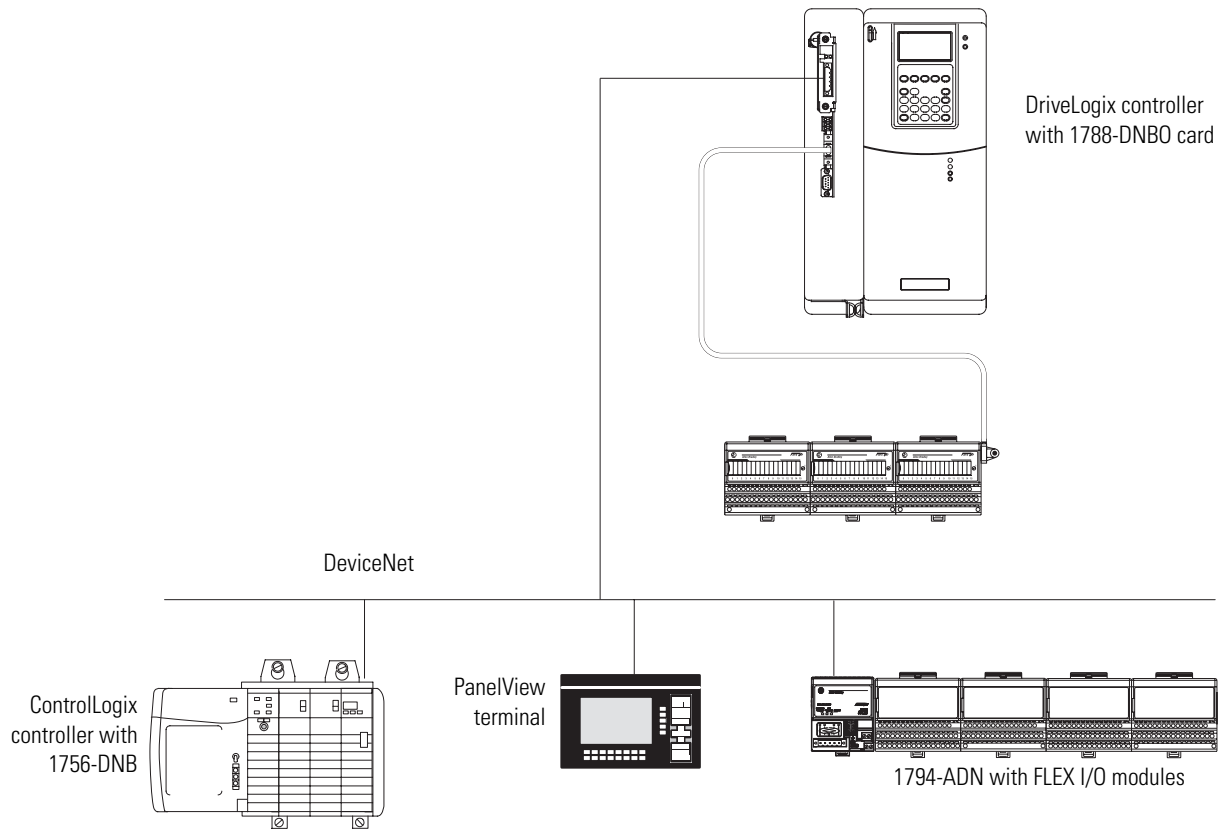
Tag Name	Value
Local:I	
Local:O	
NetLinx_DeviceNet_Interface_Card:I	
NetLinx_DeviceNet_Interface_Card:O	
NetLinx_DeviceNet_Interface_Card:O.CommandRegister	
NetLinx_DeviceNet_Interface_Card:O.CommandRegister.Run	
NetLinx_DeviceNet_Interface_Card:O.CommandRegister.Fault	
NetLinx_DeviceNet_Interface_Card:O.CommandRegister.DisableNetwork	
NetLinx_DeviceNet_Interface_Card:O.CommandRegister.HaltScanner	
NetLinx_DeviceNet_Interface_Card:O.CommandRegister.Reset	
NetLinx_DeviceNet_Interface_Card:O.Data	
NetLinx_DeviceNet_Interface_Card:S	
Stop	
Start	
Motor	

For example:



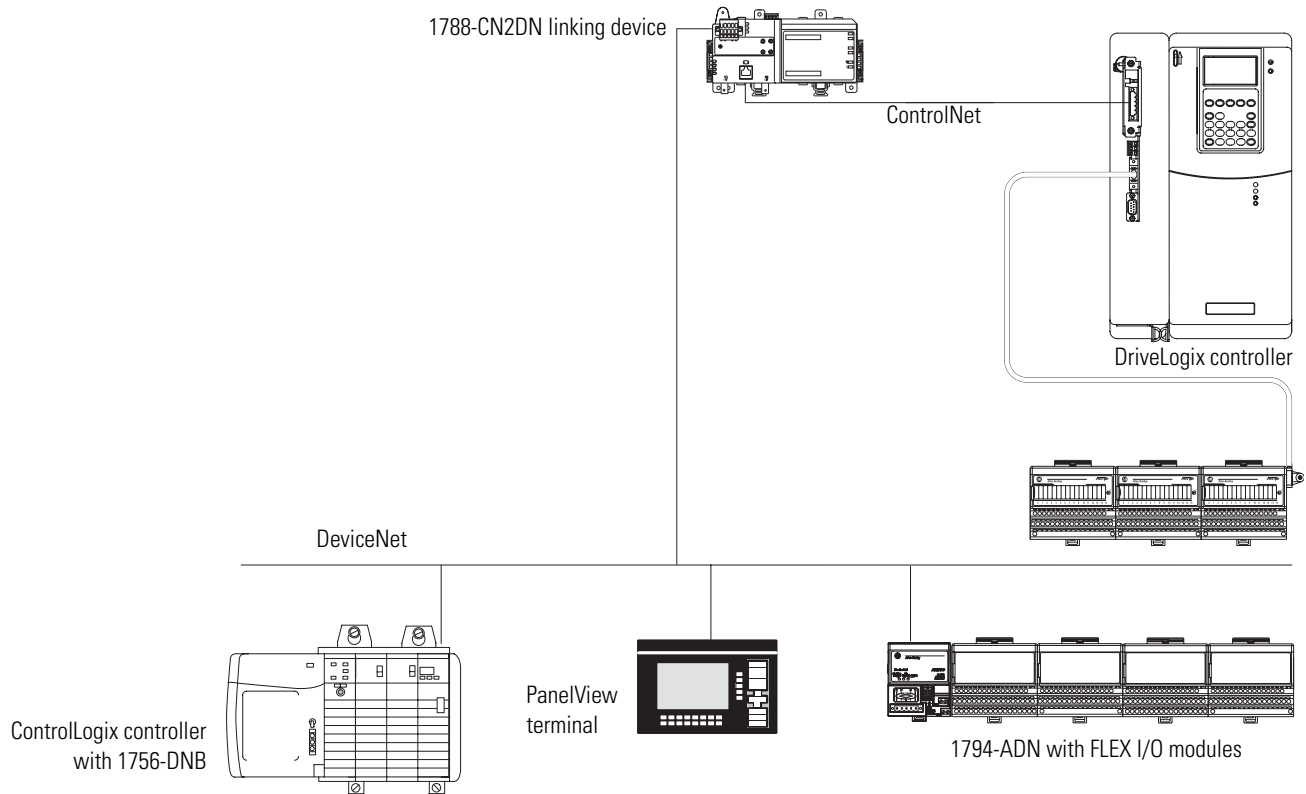
Example 1: DriveLogix Controller and DeviceNet Devices

In the following example, one DriveLogix controller controls remote DeviceNet devices through a 1788-DNBO card.



Example 2: Using a 1788-CN2DN Linking Device

In the following example, one DriveLogix controller controls remote DeviceNet devices through a 1788-CN2DN linking device.



This example has a DriveLogix controller controlling three DeviceNet devices through the linking device. The controller automatically creates a rack-optimized connection for the remote data based on the configuration of the linking device.

The tag name for the rack-optimized array tag is based on the name of the linking device. For example, if you name the linking device “cn_2_dnet,” the software automatically creates cn_2_dnet:I and cn_2_dnet:O data structures.

EXAMPLE



The 1788-CN2DN device in this example is named “ControlNet_Interface_Module”.

The data for the linking device is configured as a rack-optimized connection.

The rack-optimized connection creates a DINT element for mapped data for each DeviceNet module connected to the linking device “cnet_2_dnet.” The array cnet_2_dnet:I.Data contains the possible input elements; the cnet_2_dnet.O.Data contains the possible output elements.

Tag Name	Value	Force Mas	Style
Local:I	{...}	{...}	
Local:O	{...}	{...}	
ControlNet_to_DeviceNet_Linking_Device:I	{...}	{...}	
ControlNet_to_DeviceNet_Linking_Device:I.StatusRegister	{...}	{...}	
ControlNet_to_DeviceNet_Linking_Device:I.Data	{...}	{...}	Decimal
ControlNet_to_DeviceNet_Linking_Device:O	{...}	{...}	
ControlNet_to_DeviceNet_Linking_Device:O.CommandRegister	{...}	{...}	
ControlNet_to_DeviceNet_Linking_Device:O.Data	{...}	{...}	Decimal
ControlNet_to_DeviceNet_Linking_Device:S	{...}	{...}	
ControlNet_to_DeviceNet_Linking_Device:S.ScanCounter	2#0000_0000_00...		Binary
ControlNet_to_DeviceNet_Linking_Device:S.DeviceFailureRegister	{...}	{...}	Binary
ControlNet_to_DeviceNet_Linking_Device:S.AutoverifyFailureRegister	{...}	{...}	Binary
ControlNet_to_DeviceNet_Linking_Device:S.DeviceIdleRegister	{...}	{...}	Binary
ControlNet_to_DeviceNet_Linking_Device:S.ActiveNodeRegister	{...}	{...}	Binary
ControlNet_to_DeviceNet_Linking_Device:S.StatusDisplay	{...}	{...}	Binary
ControlNet_to_DeviceNet_Linking_Device:S.ScannerAddress	16#00		Hex
ControlNet_to_DeviceNet_Linking_Device:S.ScannerStatus	16#00		Hex
ControlNet_to_DeviceNet_Linking_Device:S.ScrollingDeviceAddress	16#00		Hex
ControlNet_to_DeviceNet_Linking_Device:S.ScrollingDeviceStatus	16#00		Hex
ControlNet_to_DeviceNet_Linking_Device:S.DeviceStatus	{...}	{...}	Hex

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

System requirements for using the linking device

If you want to use the linking device to connect to DeviceNet, you need:

- a 1788-CN2DN ControlNet to DeviceNet linking device

As a bridge, the linking device routes I/O and messaging data with a 5 ms delay. As a ControlNet device it offers a 2ms network update time. As a DeviceNet device, it provides full DeviceNet DML Scanner compatibility.

- a 1788-CN_x communication card installed in the DriveLogix communication slot for the ControlNet network
- RSLogix 5000 programming software to configure the linking device module as part of the DriveLogix system
- RSNetWorx for ControlNet software to configure the 1788-CN2DN device on the ControlNet network

- RSNetWorx for DeviceNet software to configure the 1788-CN2DN device on the DeviceNet network

Placing DeviceNet devices

The linking device supports 124, 32-bit words each of input data, output data, and status data. How you configure the DeviceNet devices determines how many words you use per device.

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

Notes:

Communicating with Devices on a Serial Link

Using This Chapter

For information about:	See page
Configuring your system for a serial link	9-1
Example 1: workstation directly connected to a DriveLogix controller	9-8
Example 2: workstation remotely connected to a DriveLogix controller	9-9
Example 3: DriveLogix controller communicating with a bar code reader	9-14

IMPORTANT

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

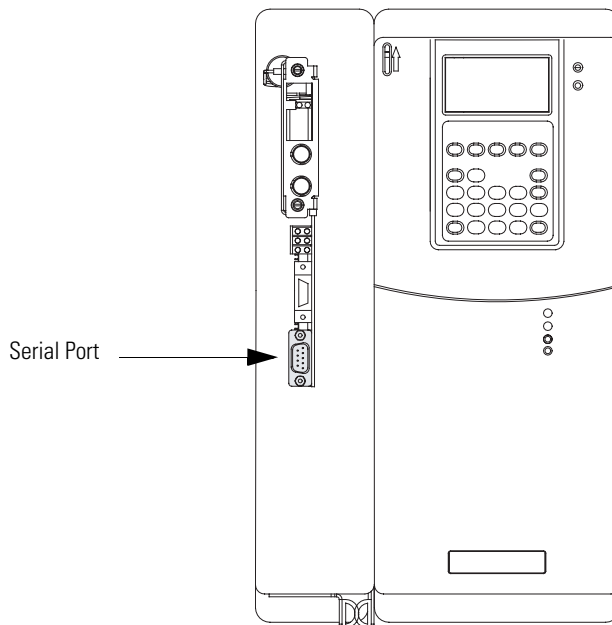
Configuring Your System for a Serial Link

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

- a workstation with a serial port
- RSLinx software to configure the serial communication driver
- RSLogix 5000 programming software to configure the serial port of the controller

Step 1: Configure the hardware

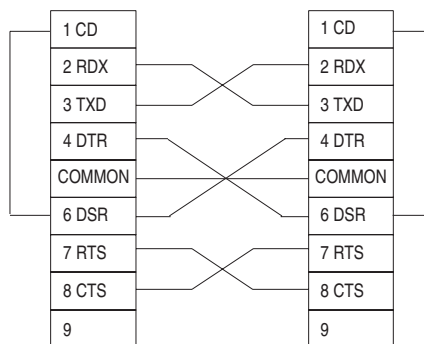
The RS-232 port is an isolated serial port built-in to the front of the controller. Refer to Access Procedures on page C-1 to understand how to gain access to the front of the DriveLogix controller.



To connect to the serial port:

1. Select the appropriate cable.

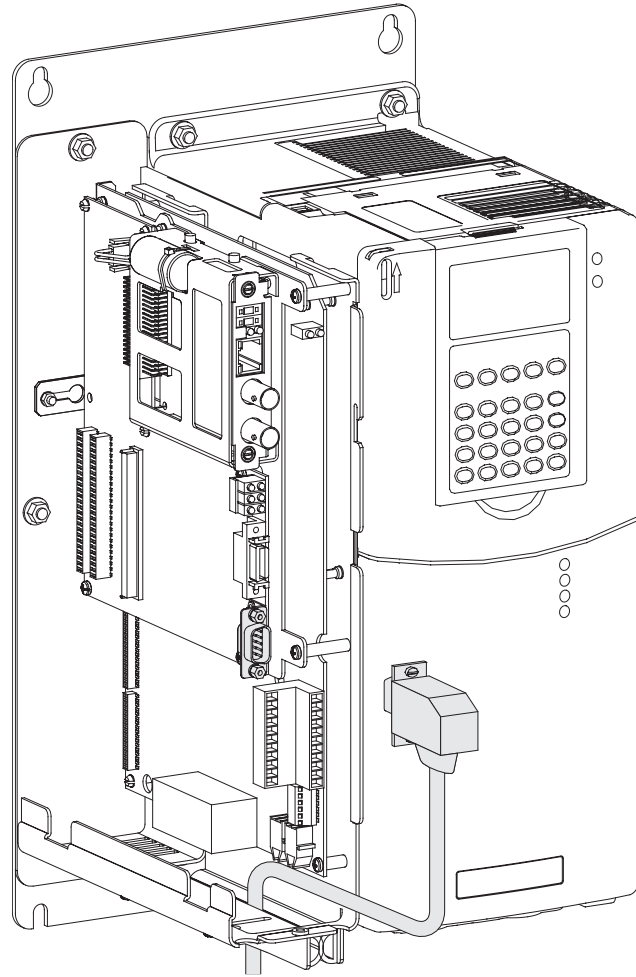
The 1756-CP3 cable attaches the controller directly to the controller.



If you make your own cable, it must be shielded and the shields must be tied to the metal shell (that surrounds the pins) on both ends of the cable.

You can also use a 1747-CP3 cable (from the SLC product family). This cable has a taller right-angle connector housing than the 1756-CP3 cable.

2. Connect the cable to the serial port on the controller..

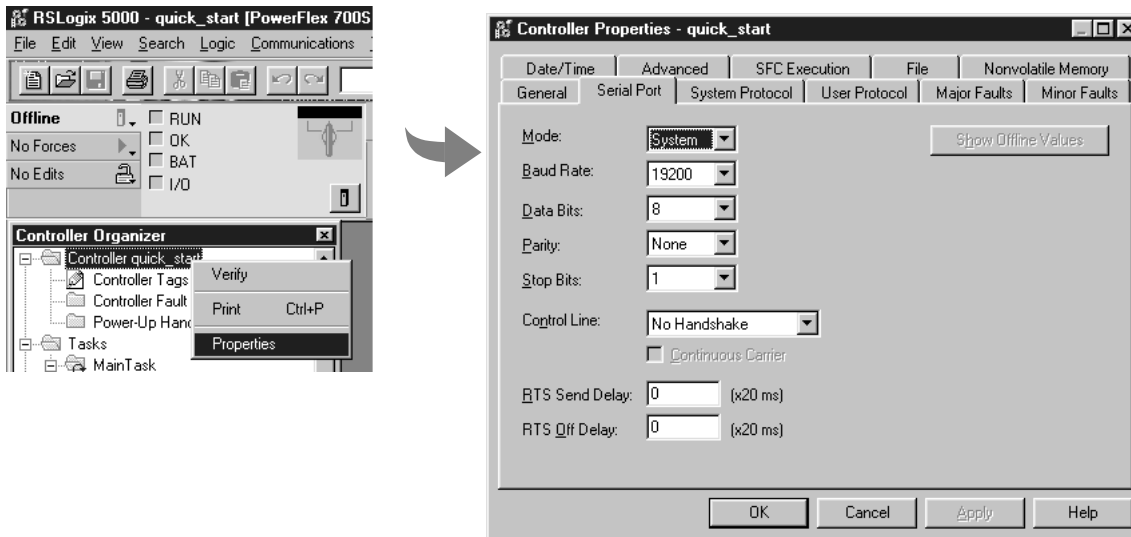


TIP

Remember to route cable through sliding access panel at the bottom of the Control Assembly.

Step 2: Configure the serial port of the controller

1. In RSLogix 5000 programming software, select the Controller folder. Right-click to select Properties.



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

Specifying serial port characteristics

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

Characteristic:	Description (default is shown in bold):
Mode	Select System (for DF1 communication) or User mode (for ASCII communication).
Baud rate	Specifies the communication rate for the serial port. Select a baud rate that all devices in your system support. Select 110, 300 600, 1200, 2400, 4800, 9600, 19200 , or 38400 Kbps.
Parity	Specifies the parity setting for the serial port. Parity provides additional message-packet error detection. Select None or Even.

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

IMPORTANT

Half-Duplex settings do not work as with other Logix controllers. RTS and CTS are not functional

Specifying system protocol characteristics

The available system modes are:

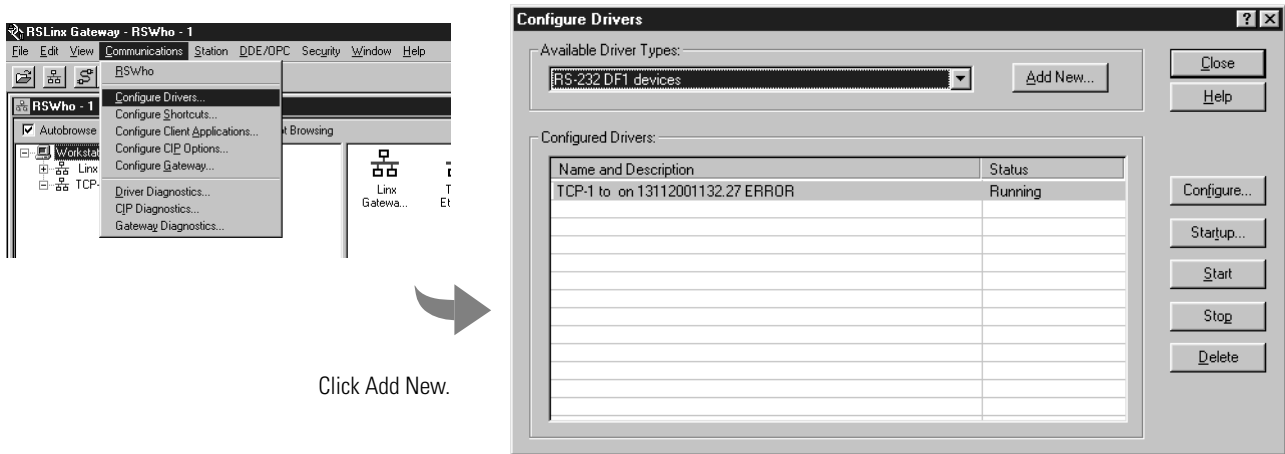
Use this mode:	For:	See page:
DF1 point-to-point	communication between the controller and one other DF1-protocol-compatible device. This is the default system mode. This mode is typically used to program the controller through its serial port.	9-8

Use this mode:	For:	See page:
DF1 master mode	control of polling and message transmission between the master and slave nodes.	9-11
	The master/slave network includes one controller configured as the master node and as many as 254 slave nodes. Link slave nodes using modems or line drivers.	
	A master/slave network can have node numbers from 0-254. Each node must have a unique node address. Also, at least 2 nodes must exist to define your link as a network (1 master and 1 slave station are the two nodes).	
DF1 slave mode	using a controller as a slave station in a master/slave serial communication network.	9-11
	When there are multiple slave stations on the network, link slave stations using modems or line drivers. When you have a single slave station on the network, you do not need a modem to connect the slave station to the master; you can configure the control parameters for no handshaking. You can connect 2-255 nodes to a single link. In DF1 slave mode, a controller uses DF1 half-duplex protocol.	
	One node is designated as the master and it controls who has access to the link. All the other nodes are slave stations and must wait for permission from the master before transmitting.	
User mode	communicating with ASCII devices	9-15
	This requires your program logic to use the ASCII instructions to read and write data from and to an ASCII device.	

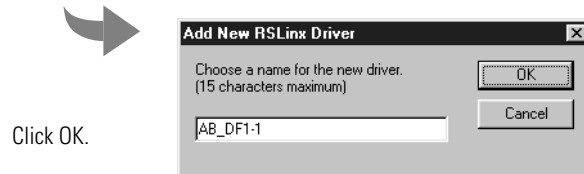
Step 3: Configure the communication driver

Use RSLinx software to configure the serial communication driver. Select the “DF1” driver.

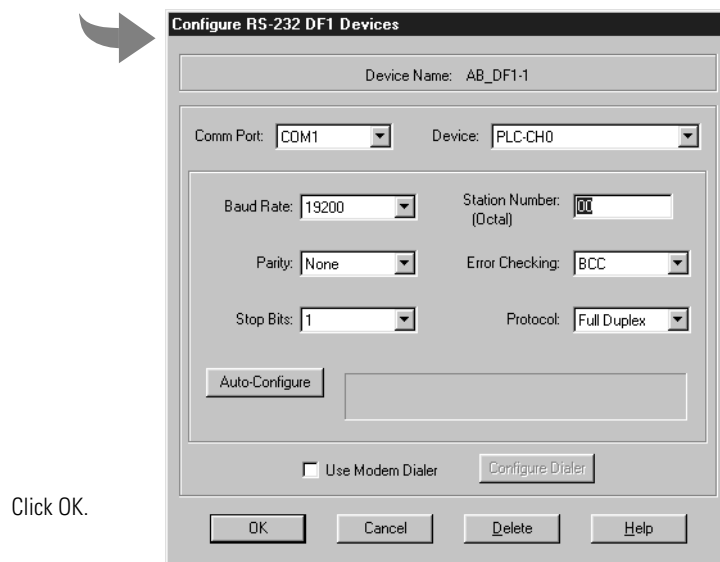
1. In RSLinx software, select Communication → Configure Driver. From the Available Driver Types list, select “RS-232 DF1 Devices”.



2. Specify a name for the driver

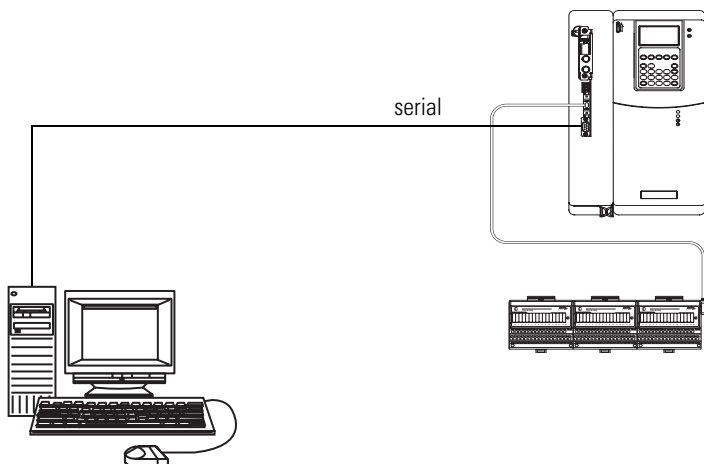


3. Specify the appropriate communication settings.



Example 1: Workstation Directly Connected to a DriveLogix Controller

In the following example, a workstation directly connects to a DriveLogix controller over a serial link. This is useful for downloading a controller project directly to the controller.



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

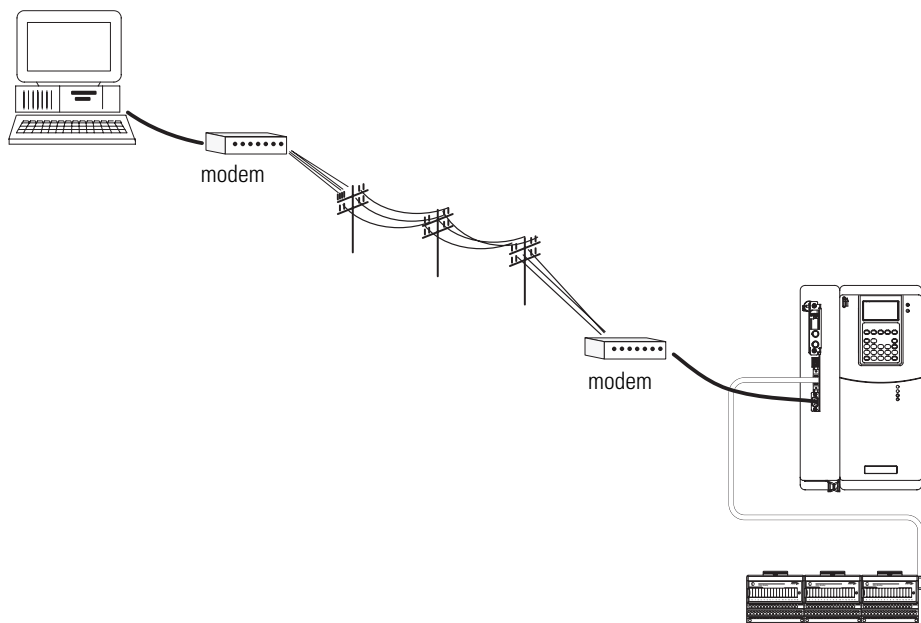
Configuring a DF1 point-to-point station

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).

This field:	Description:
Embedded response	<p>Specifies how to enable embedded responses.</p> <p>Select Autodetect (enabled only after receiving one embedded response) or Enabled. The default is Autodetect.</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>

Example 2: Workstation Remotely Connected to a DriveLogix Controller

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



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

the serial port of the controller for either the DF1 master or DF1 slave protocol (both half-duplex).

Master/slave communication methods

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

Name:	This method:	Benefits:
standard communication mode	<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 style="text-align: center;"><i>or</i></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 a DF1 slave station

This field:	Description:
Station address	<p>The station address for the serial port on the DF1 slave.</p> <p>Enter a valid DF1 address (0-254). Address 255 is reserved for broadcast messages. The default is 0.</p>
Transmit retries	<p>The number of times the remote station retries a message after the first attempt before the station declares the message undeliverable.</p> <p>Enter a value 0-127. The default is 3.</p>
Slave poll timeout	<p>Specifies the amount of time the slave station waits to be polled by a master before indicating a fault.</p> <p>Enter a value 0-32767. Limits are defined in 20ms intervals. The default is 3000 (60,000ms).</p>
EOT suppression	<p>Select whether or not to suppress sending EOT packets in response to a poll. The default is not to suppress sending EOT packets.</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>

Configuring a DF1 master station

This field:	Description:
Station address	<p>The station address for the serial port on the DF1 master.</p> <p>Enter a valid DF1 address (0-254). Address 255 is reserved for broadcast messages. The default is 0.</p>
Transmit retries	<p>Specifies the number of times a message is retried after the first attempt before being declared undeliverable.</p> <p>Enter a value 0-127. The default is 3.</p>
ACK timeout	<p>Specifies the amount of time you want the controller to wait for an acknowledgment to its message transmission.</p> <p>Enter a value 0-32767. Limits are defined in 20ms intervals. The default is 50 (1000ms).</p>
Reply message wait	<p>Message-based polling mode only</p> <p>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.</p> <p>Enter a value 0-65535. Limits are defined in 20ms intervals. The default is 5 (100ms).</p>

This field:	Description:
Polling mode	<p>Select one of these:</p> <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	<p>Standard polling modes only</p> <p>Select when the master station sends messages:</p> <ul style="list-style-type: none"> • between station polls (default) • in polling sequence
Normal poll node tag	<p>Standard polling modes only</p> <p>An integer tag array that contains the station addresses of the slave stations.</p> <p>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.</p> <p>This tag must be controller-scoped. The format is:</p> <p><i>list[0]</i> contains total number of stations to poll</p> <p><i>list[1]</i> contains address of station currently being polled</p> <p><i>list[2]</i> contains address of first slave station to poll</p> <p><i>list[3]</i> contains address of second slave station to poll</p> <p><i>list[n]</i> contains address of last slave station to poll</p>
Normal poll group size	<p>Standard polling modes only</p> <p>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.</p>
Priority poll node tag	<p>Standard polling modes only</p> <p>An integer tag array that contains the station addresses of the slave stations you need to poll more frequently.</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:</p> <p><i>list[0]</i> contains total number of stations to be polled</p> <p><i>list[1]</i> contains address of station currently being polled</p> <p><i>list[2]</i> contains address of first slave station to poll</p> <p><i>list[3]</i> contains address of second slave station to poll</p> <p><i>list[n]</i> contains address of last slave station to poll</p>

This field:	Description:
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

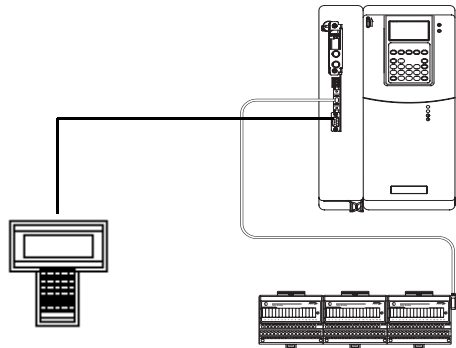
The master station polls the slave stations in this order:

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.

Example 3: DriveLogix Controller to a Bar Code Reader

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



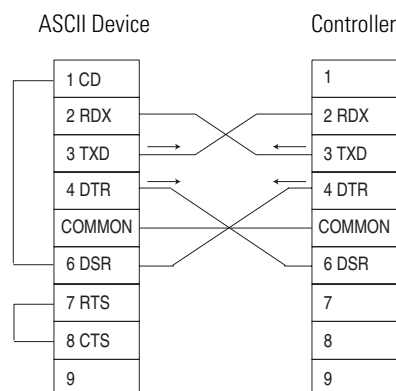
Connect the ASCII device to the controller

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

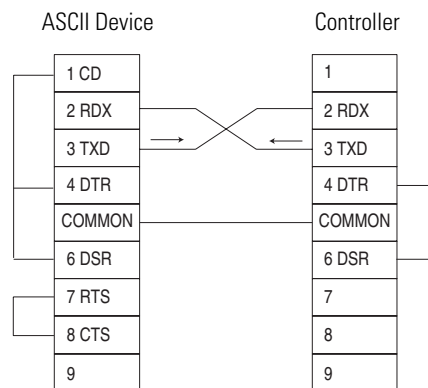
1. For the serial port of the ASCII device, determine which pins send signals and which pins receive signals.
2. Connect the sending pins to the corresponding receiving pins and attach jumpers:

If the communications: **Then wire the connectors as follows:**

handshake



do not handshake



3. Attach the cable shield to both connectors and tie the cable to both connectors.
4. Connect the cable to the controller and the ASCII device.

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

Configuring user mode

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

Programming ASCII instructions

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

For information about using these examples, see the *Logix5000 Controllers Reference Manual*, publication 1756-UM001.

Communicating with Devices on a DH-485 Link

Using This Chapter

The DH-485 protocol uses RS-485 half-duplex as its physical interface. (RS-485 is a definition of electrical characteristics; it is *not* a protocol.) You can configure the RS-232 port of the DriveLogix controller to act as an DH-485 interface.

For information about:	See page
Configuring your system for a DH-485 link	10-1
Planning a DH-485 network	10-4
Installing a DH-485 network	10-6
Example 1: DriveLogix controller, ControlLogix controller, and SLC controller on the same DH-485 network	10-9

IMPORTANT

A DH-485 network consists of multiple cable segments. Limit the total length of all the segments to 1219m (4000 ft.).

Configuring Your System for a DH-485 Link

For the DriveLogix controller to operate on a DH-485 network, you need:

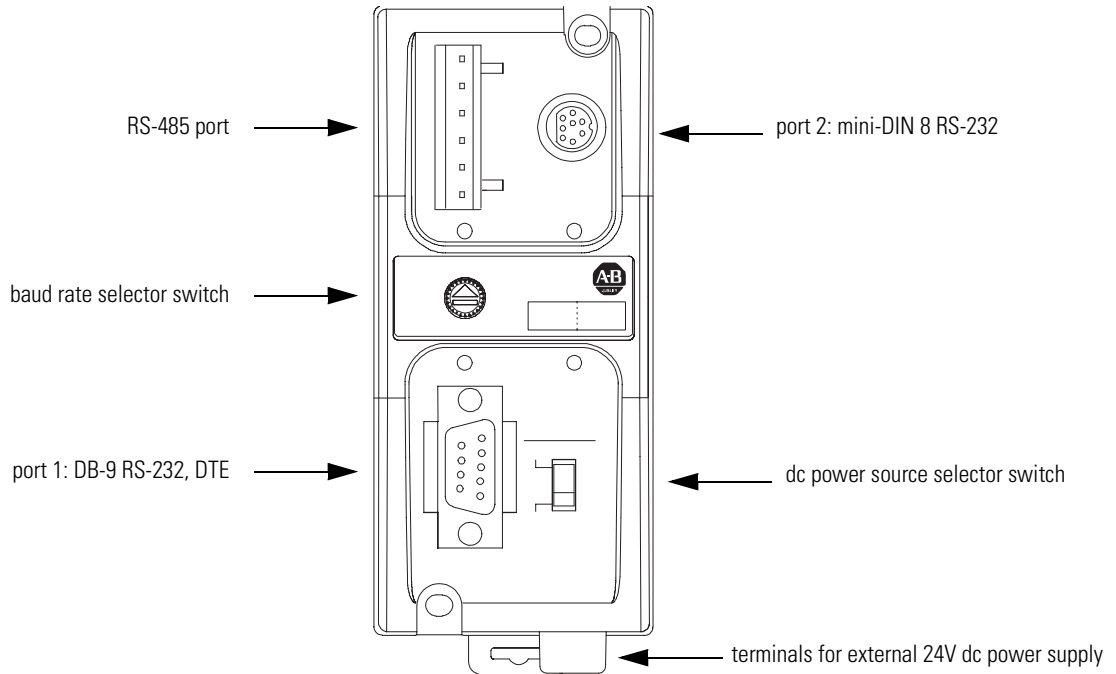
- a 1761-NET-AIC converter for each DriveLogix controller you want to put on the DH-485 network.
- RSLogix 5000 programming software to configure the serial port of the controller for DH-485 communications

Step 1: Configure the hardware

The RS-232 port on the front of the DriveLogix controller supports the requirements you need for the DH-485 network connection.

Connect the controller to an RS-232-to-RS-485 isolator. One possible isolator is the 1761-NET-AIC interface converter. Refer to Access Procedures on

page C-1 to understand how to gain access to the front of the DriveLogix controller.



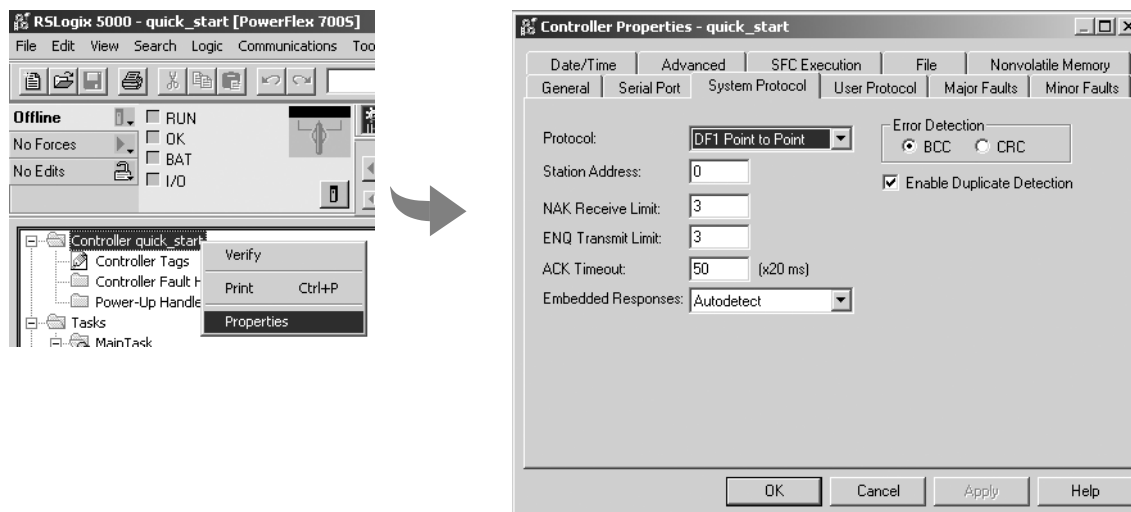
Connect the serial port of the DriveLogix controller to either port 1 or port 2 of the 1761-NET-AIC converter. Use the RS-485 port to connect the converter to the DH-485 network.

The cable you use to connect the controller depends on the port you use on the 1761-NET-AIC converter.

If you connect to this port:	Use this cable:
port 1	1747-CP3
DB-9 RS-232, DTE connection	or 1761-CBL-AC00
port 2	1761-CBL-AP00
mini-DIN 8 RS-232 connection	or 1761-CBL-PM02

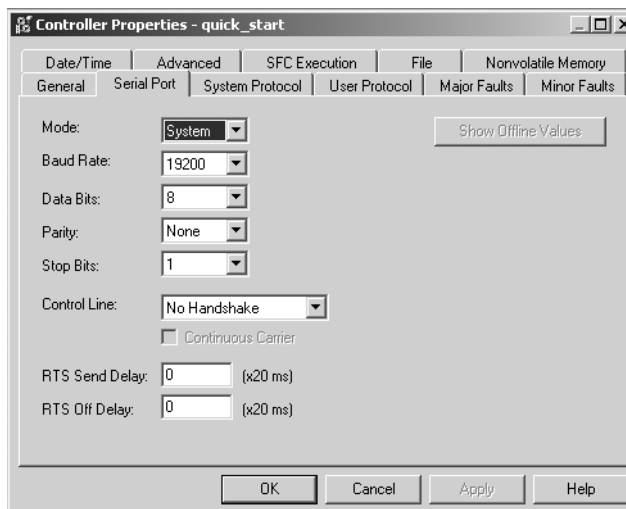
Step 2: Configure the DH-485 port of the controller

1. In RSLogix 5000 programming software, select the Controller folder. Right-click to select Properties.



3. On the Serial Port tab, specify the appropriate communication settings.

The grayed out settings are selections that do not apply to a DH-485 network.



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

Characteristic:	Description (default is shown in bold):
Baud Rate	Specifies the communication rate for the DH-485 port. All devices on the same DH-485 network must be configured for the same baud rate. Select 9600 or 19200 Kbps.
Node Address	Specifies the node address of the DriveLogix controller on the DH-485 network. Select a number 1-31 decimal, inclusive.
	To optimize network performance, assign node addresses in sequential order. Initiators, such as personal computers, should be assigned the lowest address numbers to minimize the time required to initialize the network.
Token Hold Factor	Number of transmissions (plus retries) that a node holding a token can send onto the data link each time that it receives the token. Enter a value between 1-4. The default is 1.
Maximum Node Address	Specifies the maximum node address of all the devices on the DH-485 network. Select a number 1- 31 decimal, inclusive.
	To optimize network performance, make sure: <ul style="list-style-type: none"> • the maximum node address is the highest node number being used on the network • that all the devices on the same DH-485 network have the same selection for the maximum node address.

Planning a DH-485 Network

The DH-485 network offers:

- interconnection of 32 devices
- multi-master capability
- token passing access control
- the ability to add or remove nodes without disrupting the network
- maximum network length of 1219 m (4000 ft.)

The DH-485 protocol supports two classes of devices: initiators and responders. All initiators on the network get a chance to initiate message transfers. The DH-485 protocol uses a token-pass algorithm to determine which initiator has the right to transmit

DH-485 token rotation

A node holding the token can send any valid packet onto the network. Each node gets only one transmission (plus two retries) each time it receives the token. After a node sends one message packet, it attempts to give the token to its successor by sending a “token pass” packet to its successor.

If no network activity occurs, the initiator sends the token pass packet again. After two retries (a total of three tries) the initiator attempts to find a new successor.

IMPORTANT

The maximum address that the initiator searches for before starting again with zero is the value in the configurable parameter “maximum node address.” The default value for this parameter is 31 for all initiators and responders.

The allowable range of the node address of an initiator is 0 to 31. The allowable address range for all responders is 1 to 31. There must be at least one initiator on the network.

Network initialization

The network requires at least one initiator to initialize it. Network initialization begins when an initiator on the network detects a period of inactivity that exceeds the time of a link dead timeout. When the link dead timeout is exceeded, usually the initiator with the lowest address claims the token. When an initiator has the token it will begin to build the network.

Building a network begins when the initiator that claimed the token tries to pass the token to the successor node. If the attempt to pass the token fails, or if the initiator has no established successor (for example, when it powers up), it begins a linear search for a successor starting with the node above it in the addressing.

When the initiator finds another active initiator, it passes the token to that node, which repeats the process until the token is passed all the way around the network to the first node. At this point, the network is in a state of normal operation.

Number of nodes and node addresses

The number of nodes on the network directly affects the data transfer time between nodes. Unnecessary nodes (such as a second programming terminal that is not being used) slow the data transfer rate. The maximum number of nodes on the network is 32.

If the node addresses for controllers are assigned in sequence, starting at node 1 (with node 0 left for a programming terminal), it is as efficient to leave the maximum node address at 31 as it is to decrease it to the highest node address on the network. Then, adding devices to the network at a later time will not require modifying the maximum node address in every device on the network.

The maximum node address should be the same for all devices on a DH-485 network for optimal operation.

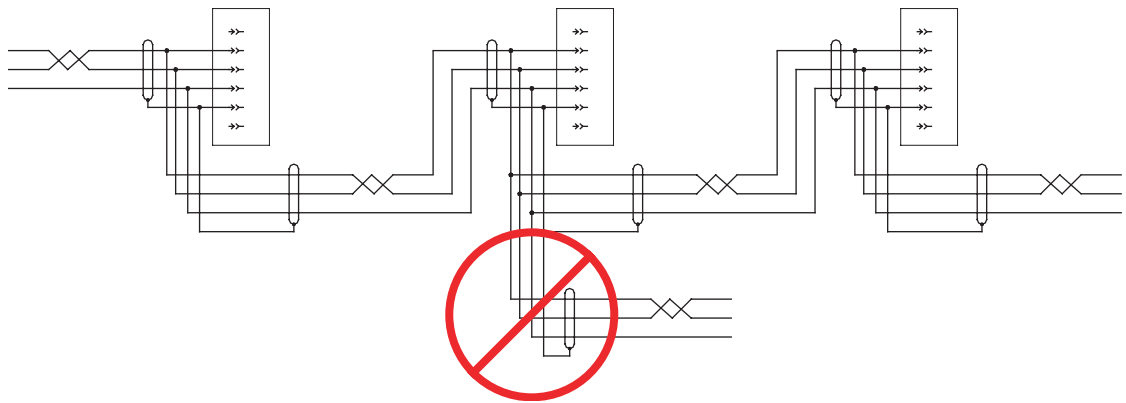
The best network performance occurs when node addresses start at 0 and are assigned in sequential order. The controller defaults to node address 1 (controllers cannot be node 0). Initiators, such as personal computers, should be assigned the lowest numbered addresses to minimize the time required to initialize the network.

Installing a DH-485 Network

A DH-485 network consists of a number of cable segments daisy-chained together. The total length of the cable segments cannot exceed 1219 m (4000 ft).

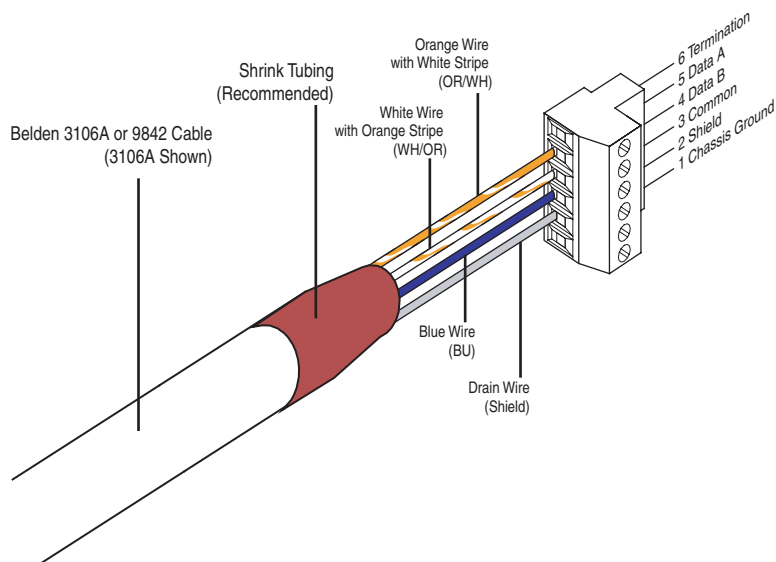
IMPORTANT

Use shielded, twisted-pair cable - either Belden 3106A or Belden 9842. A daisy-chained network is recommended. Star connections are not recommended



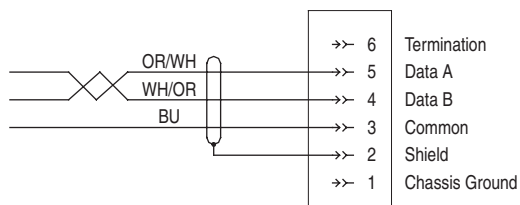
When cutting cable segments, make them long enough to route them from one link coupler to the next with sufficient slack to prevent strain on the connector. Allow enough extra cable to prevent chafing and kinking in the cable.

Single Cable Connection



Connections Using Belden 3106 Cable

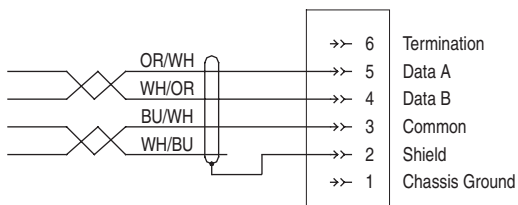
The table and schematic diagram below shows wire/terminal connections for Belden 3106A cable.



For this Wire/Pair	Connect this Wire	To this Terminal
white/orange	orange with white stripe	5 - Data A)
	white with orange stripe	4 - (Data B)
blue	blue	3 - (Common)
shield/drain	non-jacketed	2 - Shield

Connections Using Belden 9842 Cable

The table and schematic diagram below shows wire/terminal connections for Belden 9842 cable.



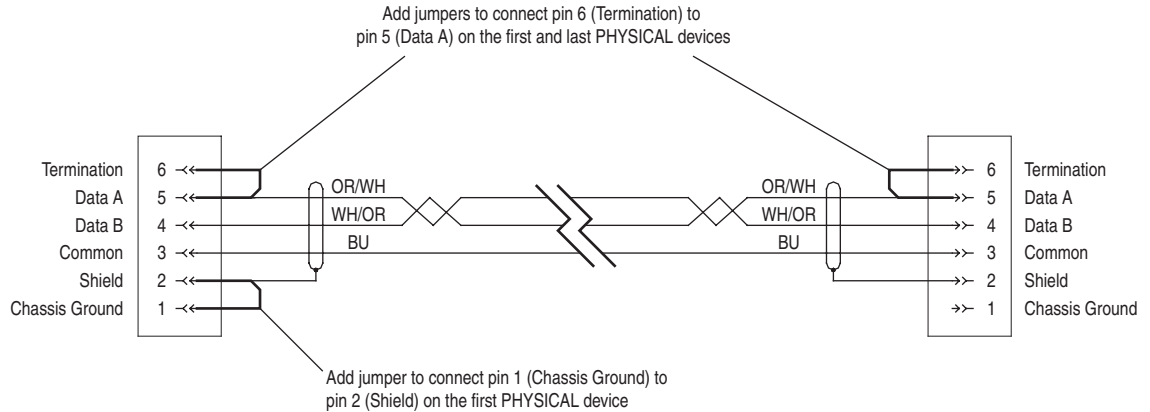
For this Wire/Pair	Connect this Wire	To this Terminal
white/orange	orange with white stripe	5 - (Data A)
	white with orange stripe	4 - (Data B)
blue/white	white with blue stripe	cut back - no connection ⁽¹⁾
	blue with white stripe	3 - (Common)
shield/drain	non-jacketed	2 - Shield

⁽¹⁾ To prevent confusion when installing the communication cable, cut back the white with blue stripe wire immediately after the insulation jacket is removed. This wire is not used by DH-485.

Grounding and terminating a DH-485 network

You must terminate the network at the first and last PHYSICAL devices, by connecting pin 6 (Termination) to pin 5 (Data A).

You must ground the network at the first PHYSICAL device by connecting pin 1 (Chassis Ground) to pin 2 (Shield).



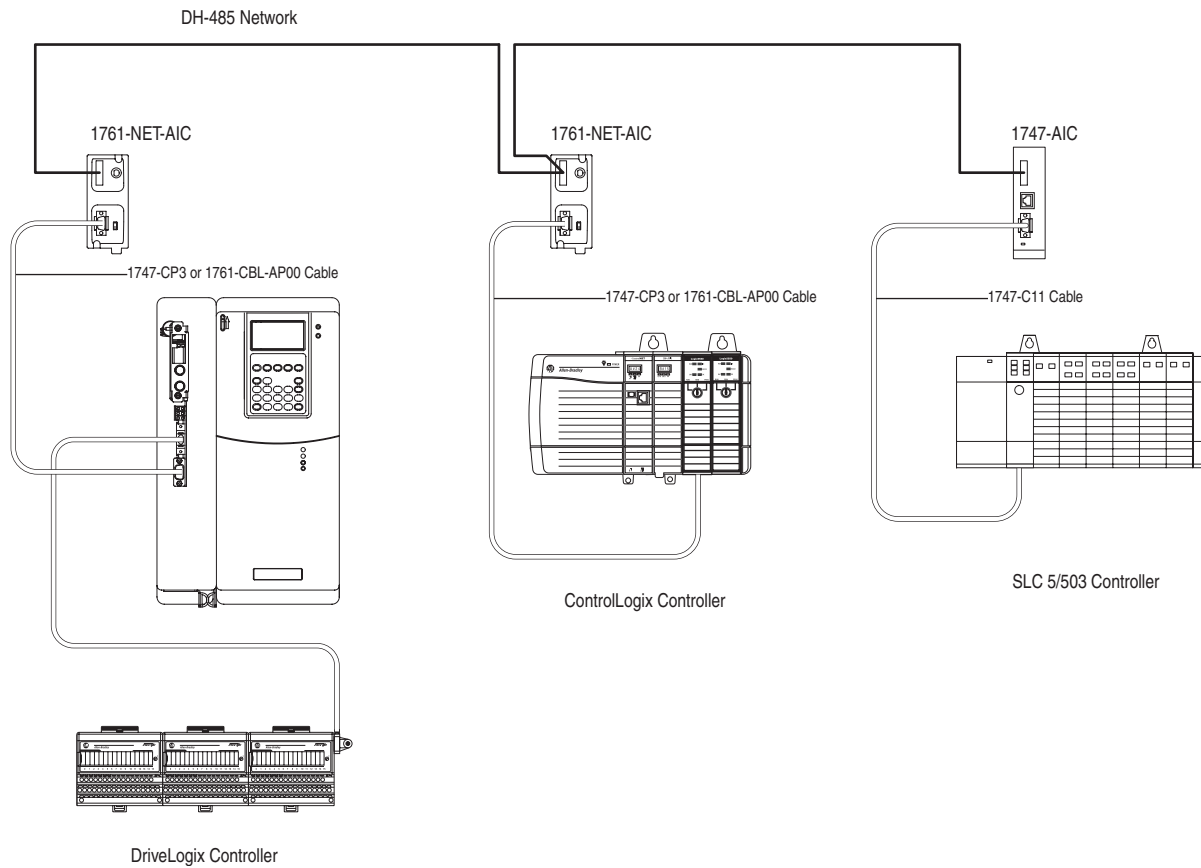
IMPORTANT

A device’s physical location may be independent of its node address. Make sure you ground and terminate the proper PHYSICAL locations.

Example: DriveLogix Controller, ControlLogix Controller, and SLC Controller on the Same DH-485 Network

In the following example, both a DriveLogix controller and a ControlLogix controller use its own 1761-NET-IAC+ converter to connect to a DH-485

network. In addition, an SLC 5/03 controller uses a 1747-AIC converter to connect to the same DH-485 network.



On the DH-485 network, the DriveLogix controller can send and receive messages to and from other controllers on the network.

Communicating with Devices on a Third-Party Link

Using This Chapter

For information about:	See page
Configuring Your System for a Third-Party Link	9-1

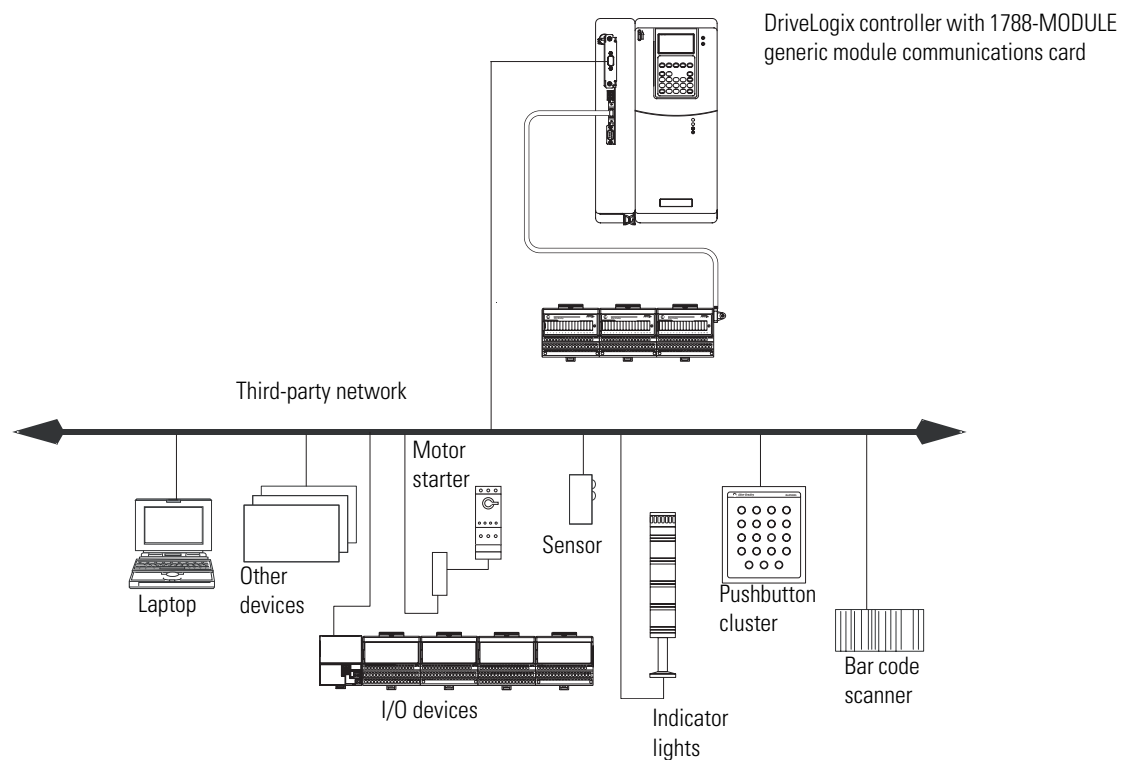
Configuring Your System for a Third-Party Link

For the DriveLogix controller to operate on a third-party network, you need:

- a 1788-MODULE generic module communication daughtercard.
- RSLogix 5000 programming software (Version 12 or later) to configure the 1788-MODULE card as part of the DriveLogix system
- Software that configures the 1788-MODULE card on the third-party network

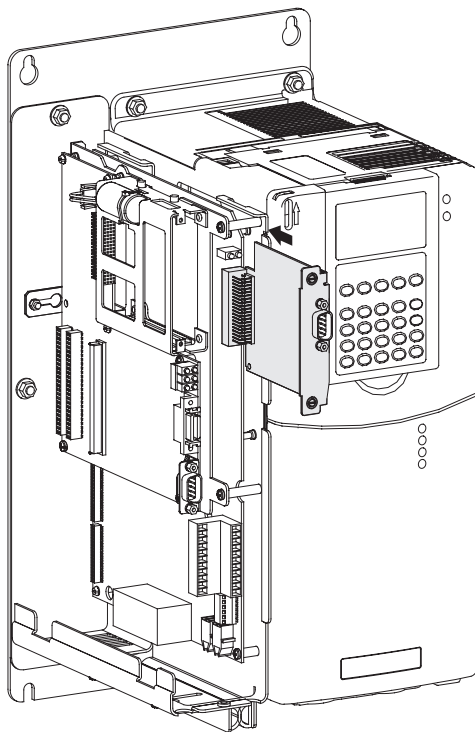
Figure 11.1 shows an example system on third-party link.

Figure 11.1



Step 1: Install the hardware

Before you can connect the DriveLogix system to the third-party network, you must configure the 1788-MODULE communication card and make sure it is properly installed in the DriveLogix controller. Refer to Access Procedures on page C-1 to understand how to gain access to the NetLinX daughtercard slot on the DriveLogix controller.

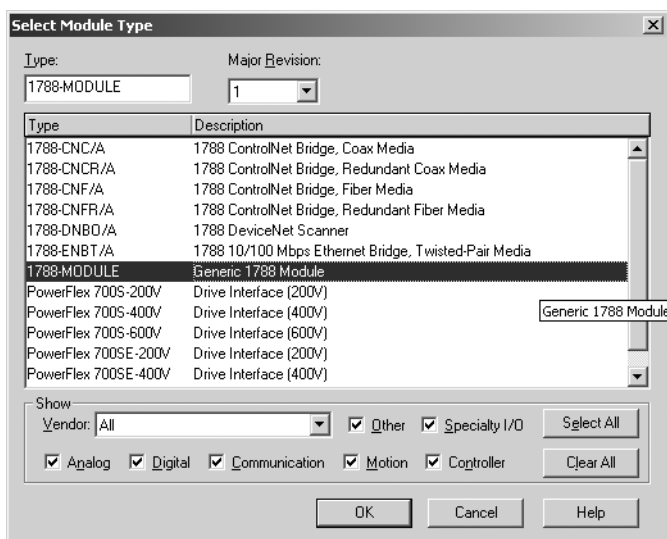
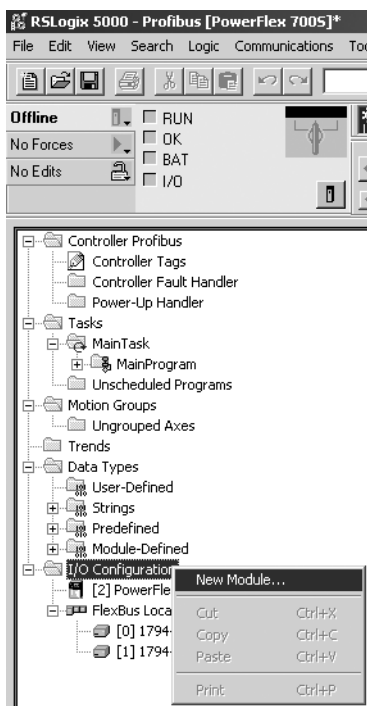


Remember which slot you use for which communication card. You'll need the slot number to configure the communication card in the RSLogix 5000 programming software. The controller uses slot 0.

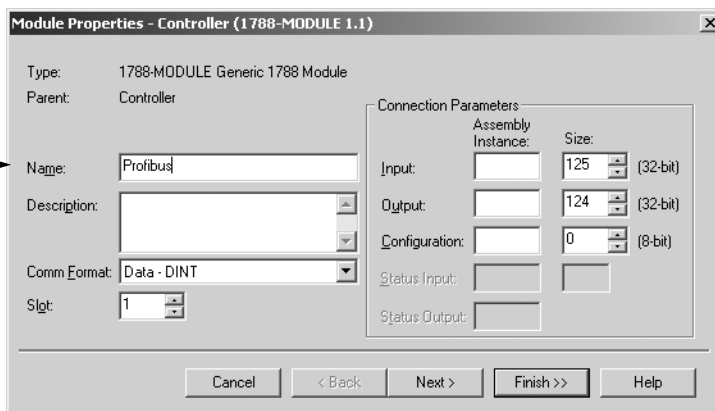
Step 2: Configure the daughtercard as part of the system

Use RSLogix 5000 programming software to map the 1788-MODULE card as part of the DriveLogix system. In the Controller Organizer, add the card to the I/O Configuration folder.

1. In RSLogix 5000 programming software, select the I/O Configuration folder.
2. Right-click to select New Module and add a 1788-MODULE communication daughtercard.
3. Specify the appropriate communication settings.



4. Specify the module's properties. For more information on communications format the connection parameters, see the next section.



Communication Format

The Communication Format field chooses a data type for information transmitted between the controller and a remote device connected to the 1788-MODULE communication card. This format creates an array in the controller of whatever data type you choose for the input and output data.

Connection Parameters

You must set connection parameters to define data identification and connection size. An Assembly Instance and Data Size must be assigned for each:

- Input
- Output
- Configuration.

Assembly Instance

The Assembly Instance is a number that identifies what data transferred between the owner-controller and I/O module looks like. You must create a map that defines your assembly instance entries.

Size

The size field determines how large the connections are between the owner-controller and the I/O module. Connections are sent in sizes matching the communications format data type selected. The default, DINT, results in 32-bit quantities.

Complete your system configuration and develop your program logic. Then download the project to the controller.

DriveLogix Back-Up on DeviceNet

Using This Chapter

For information about:	See page
How the Back-up Works	12-2
Power-Up and System Start-up	12-4
Developing the DriveLogix Back-Up Application	12-6
Using Indicators to Check Status	12-13
Development and Debugging Tips	12-13

This chapter offers a solution to back-up your DriveLogix controller on DeviceNet. DriveLogix Back-Up on DeviceNet is a simple, low-cost, back-up system most effective when used in smaller applications that require fast switchovers from a primary to a secondary controller.

This back-up solution will:

- minimize downtime in case of controller failure when the same program is used in both programs.
- mitigate the risk of changes adversely affecting the application (use old, proven program in one controller and new, untested program in other controller). If the new untested program causes a problem, a forced switchover can be made to the older proven program without downloading the program again.

The DriveLogix Back-Up on DeviceNet solution takes advantage of *Shared DeviceNet Mastership of Slave I/O Devices* technology. Typically, only a single DeviceNet master exists for any particular slave. With Shared DeviceNet Mastership, two masters can exist. Heartbeat communications between primary and secondary controllers determines which scanner is the master and which scanner remains in stand-by mode.

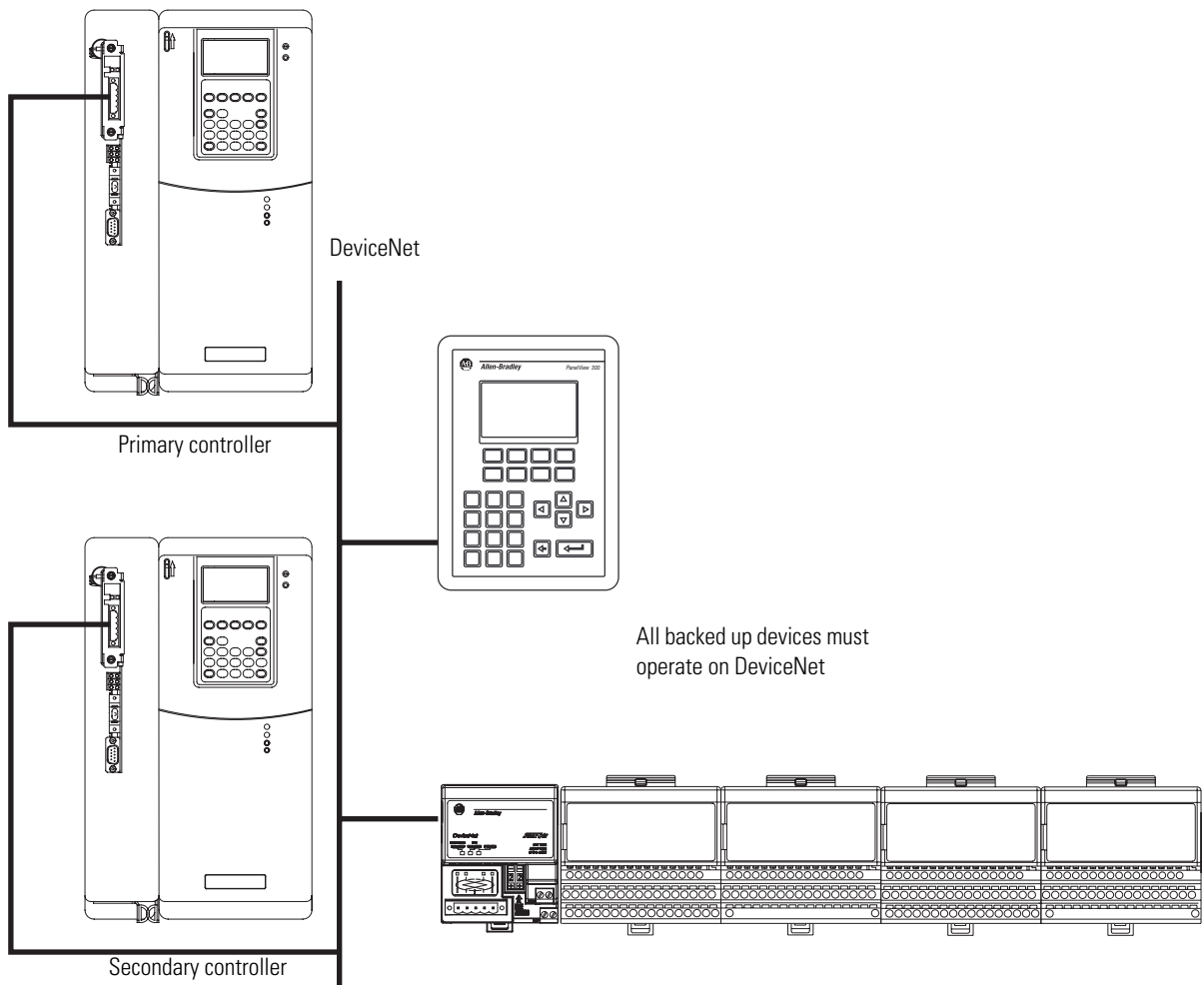
How the Back-up Works

Figure 11.2 shows an example back-up system. In the back-up system, the following occurs:

- Both controllers/scanners simultaneously receive all inputs.
- Both controllers execute in parallel but are NOT synchronized.
- Only the primary controller sends output data to the I/O devices. A virtual switch in the 1788-DNBO cards is used to switch outputs between primary and secondary controllers.
- After failure or forced switchover, outputs are automatically switched by the 1788-DNBO card from the primary controller to secondary. When the switch occurs, the secondary controller becomes the primary controller.

The switchover occurs so quickly that the I/O devices do not timeout; these devices are unaware that redundant controllers/scanners exist and are unaware of the switchover.

Figure 11.2



Requirements of the Back-Up

The DriveLogix Back-Up on DeviceNet solution requires that you use the following:

- RSLogix 5000, version 10 or higher
- 2 DriveLogix controllers, firmware revision 10.x or higher
- 2 1788-DNBO communication cards, firmware revision 2.x or higher

IMPORTANT

Many applications use multiple communications cards in a DriveLogix controller to communicate with several networks. This solution requires the software and DriveLogix controllers use version 10.x or higher.

However, if you are using the 1788-ENBT card in your application, remember that you must use software and DriveLogix controllers of version 11.x or higher.

Additional requirements are as follows:

- When setting up the DeviceNet network, you must set the primary and secondary 1788-DNBO cards to the same node address and reserve the next node address.

We recommend you set the primary and secondary 1788-DNBO node addresses to 0 and reserve node 1. However, you can use any successive node numbers (e.g. 30 and 31).

- All I/O and operator interfaces that required back-up must be on DeviceNet.
- The scanlists in the two DeviceNet scanner must be identical.

Power-Up and System Start-up

To configure a DriveLogix Back-up system on DeviceNet, you can take the following steps. Some of these steps are described in greater detail in the rest of the appendix.

1. Install all I/O and operator interfaces that you need to back-up on DeviceNet.

We recommend that you reserve node addresses 0 and 1 for the two DriveLogix controllers used in the back-up. If you do not use 0 and 1, make sure you reserve two consecutive numbers for the controllers when you install I/O and other devices on DeviceNet.

2. Connect a DriveLogix controller with a 1788-DNBO scanner to the DeviceNet network.
3. Set the controller node address to 0 (or the lower of the 2 node addresses reserved for the DriveLogix controllers).
4. Power-up the controller and the network.
5. Use RSNetWorx for DeviceNet to download the network's scanlist to the 1788-DNBO card.

You can use either a scanlist from a new configuration or previously-used configuration. If the scanlist is a new configuration, we recommend you save it to a new project for later use.

6. Use RSLogix 5000 software to download the appropriate user program to the DriveLogix controller.

The program should contain the explicit message(s) that enable the back-up feature for this controller and scanner. The messages are described in the Developing the DriveLogix Back-Up Application section beginning on page 12-6.

7. Put controller into RUN mode.
8. Either disable power to the controller or disconnect the scanner from DeviceNet. This controller will be the secondary controller.

9. Connect the other DriveLogix controller with a 1788-DNBO scanner on the network.
10. Set the node address to 0.
11. Power-up the controller and scanner.
12. Use RSNetWorx for DeviceNet to download the same scanlist used in step 5.

It may be necessary to browse the network again before downloading the scanlist. This second browsing of the network allows RSNetWorx for DeviceNet to establish communication to the new scanner at the same node number as the previous scanner.

13. Use RSLogix 5000 to download the user program to the second DriveLogix controller as performed in step 6.

Typically, the same user program is downloaded to the second DriveLogix controller as the first. However, unlike the scanlists, the user programs in the controllers do not have to be identical.

14. Put the controller into RUN mode.

This controller is now ready to go and is the primary controller.

15. Reapply power to the secondary controller and/or reconnect the secondary scanner to the DeviceNet subnet.

This completes the back-up process. For more detailed information on some of the steps listed previously, see the next section.

Developing the DriveLogix Back-Up Application

The DriveLogix back-up is enabled from an RSLogix 5000 user program with a few simple ladder rungs (or equivalent). The following rungs are used in the DriveLogix back-up:

- Back-up Heartbeat Configuration Rungs - required
- Reading Back-up State Rung - optional
- Reading Back-up Status - optional

Back-up Heartbeat Configuration Rungs

The first, and most critical, step is to set the back-up “heartbeat” constant in the DeviceNet scanner. The heartbeat constant enables the back-up feature and determines the switchover time (2 x heartbeat).

By default, the heartbeat is zero; this default value disables the back-up mode. Your user program must set the heartbeat to a non-zero value to enable back-up.

The heartbeat occurs in multiples of 8ms (i.e. 8, 16, 24, etc.). We recommend a value of 16-48ms for most applications. The recommended heartbeat times result in switchover times of 32-96ms. However, these times do not include controller scan delays.

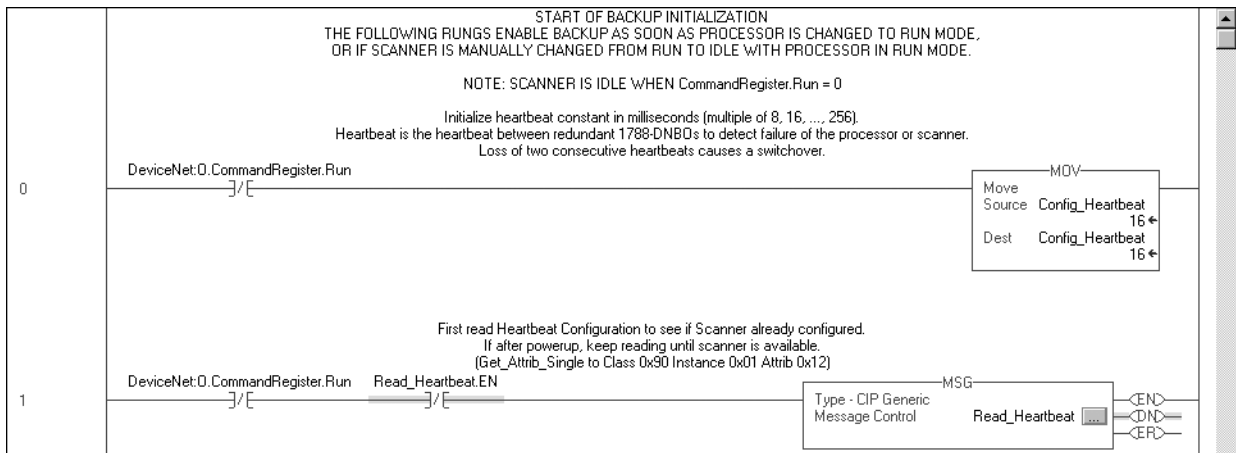
IMPORTANT

If multiples of 8 are not used for the requested heartbeat, then the DeviceNet scanner uses the next higher supported heartbeat value that can be read from the scanner. For example, if you set the heartbeat to 10, the scanner uses a 16ms heartbeat.

Setting the Heartbeat Constant

You can set the heartbeat constant with five rungs of ladder logic. Figure 11.3 shows rungs 0 & 1 and the message set-up used in rung 1. The message in rung 1 uses the INT data type.

Figure 11.3



Rung 1 message configuration and communication tabs

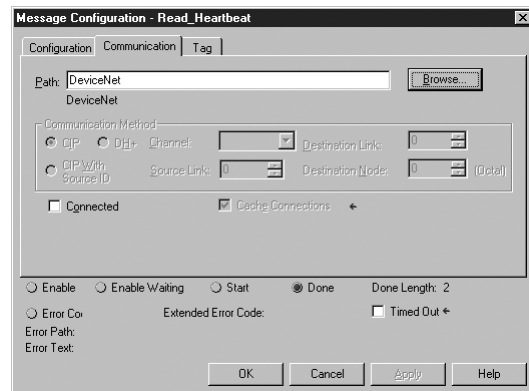
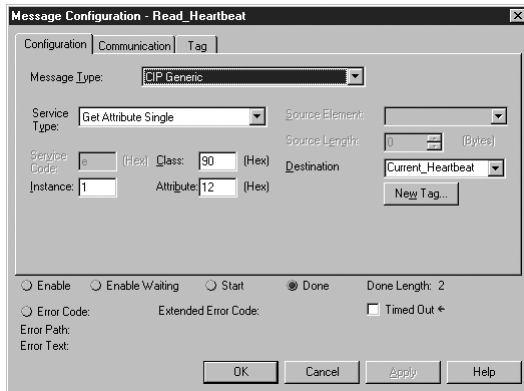
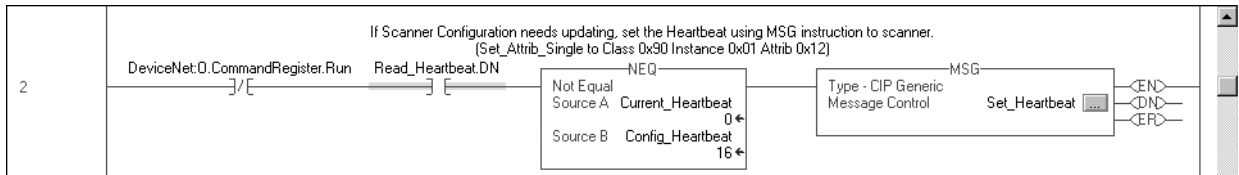


Figure 11.4 shows rung 2 and the message set-up used on it. The message in rung 2 uses the INT data type.

Figure 11.4



Rung 2 message configuration and communication tabs

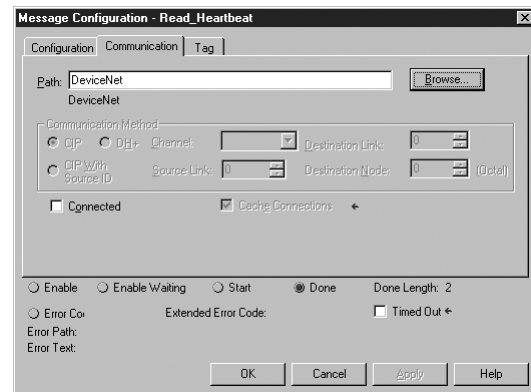
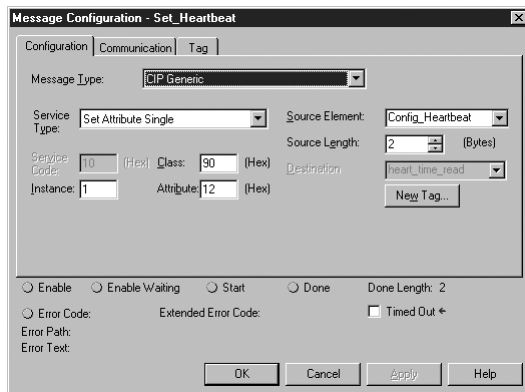
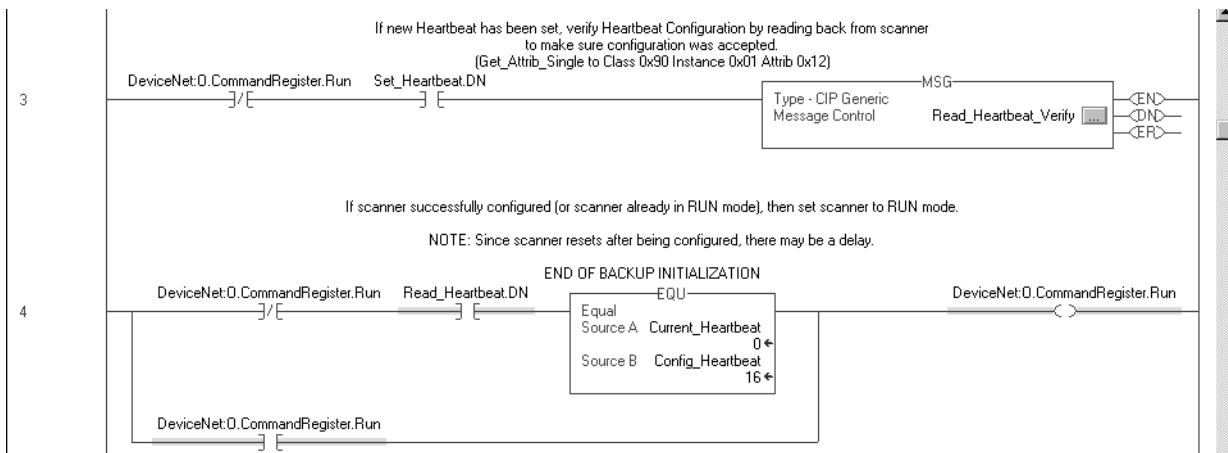
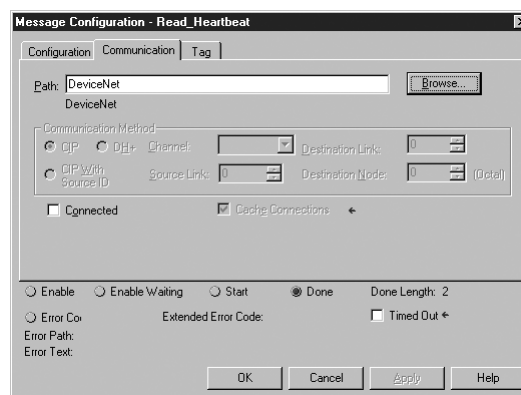
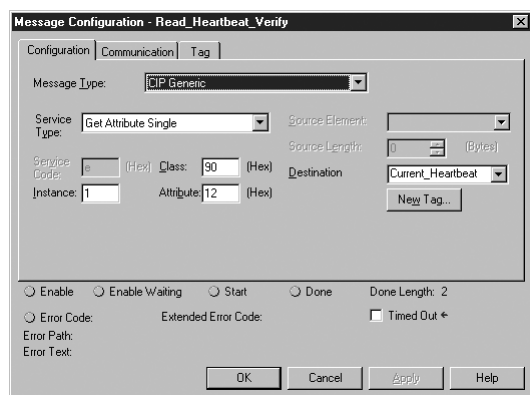


Figure 11.5 shows rungs 3 & 4 and the message set-up used on it. The message in rung 3 uses the INT data type.

Figure 11.5



Rung 3 message configuration and communication tabs



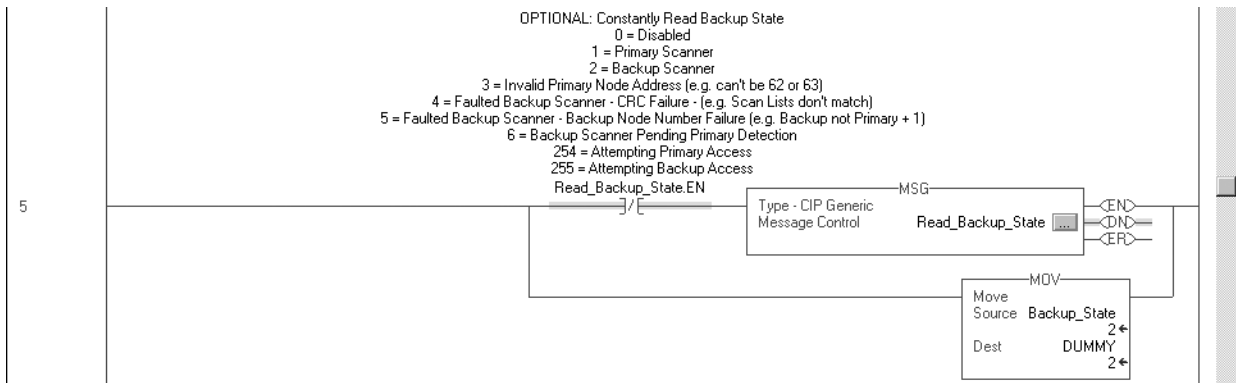
This completes the required portion of ladder logic to enable the DriveLogix back-up on DeviceNet. The following sections describe how to use additional ladder logic to read back-up state and status. However, these sections are not required to complete the back-up solution.

Reading Back-up State Rung

You can read the back-up state of the DeviceNet scanner with a single rung of ladder logic. The back-up state is useful for debug or more sophisticated back-up schemes. The message in this rung uses the SINT data type.

Figure 11.6 shows the rung you can use to read the back-up state.

Figure 11.6



Rung 5 message configuration and communication tabs

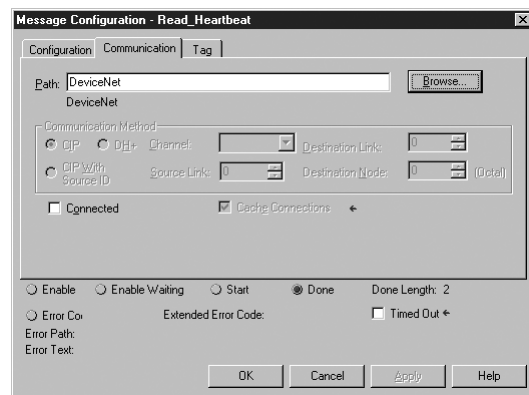
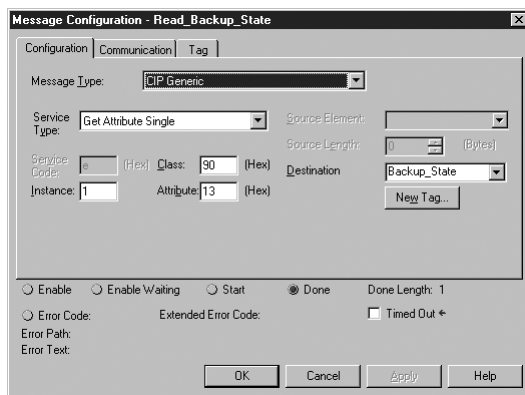


Table 11.1 describes the possible values this message may return when reading the back-up state of the DeviceNet scanner.

Table 11.1

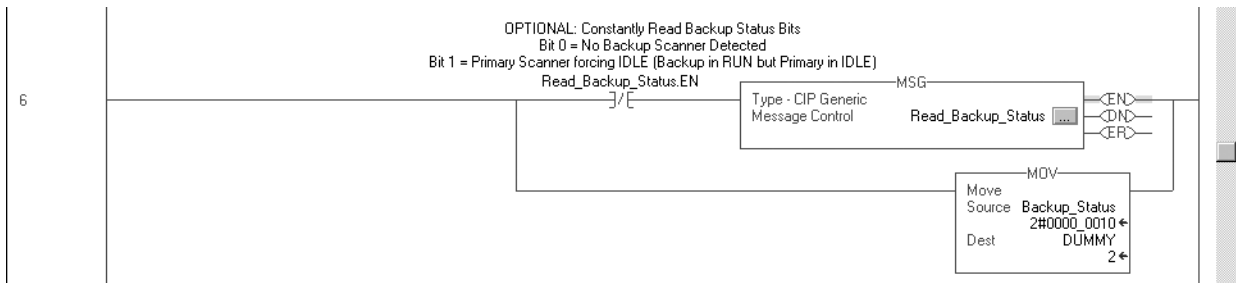
If the message reads this value:	the back-up state of the DeviceNet scanner is:
0	Disabled
1	Primary scanner
2	Back-up scanner
3	Invalid primary node address (e.g. the node address cannot be 62 or 63)
4	Faulted back-up scanner - CRC failure (e.g. the scanlists in the scanners do not match)
5	Faulted back-up scanner - back-up node number failure (e.g. the back-up scanner is not using a node number = the primary node number + 1)
6	Back-up scanner pending primary detection
254	Attempting primary access
255	Attempting back-up access

Reading Back-up Status

You can read the back-up status of the DeviceNet scanner with a single rung of ladder logic. The back-up state is useful for debugging or more sophisticated back-up schemes. The message in this rung uses the SINT data type.

Figure 11.7 shows the rung you can use to read the back-up state.

Figure 11.7



Rung 6 message configuration and communication tabs

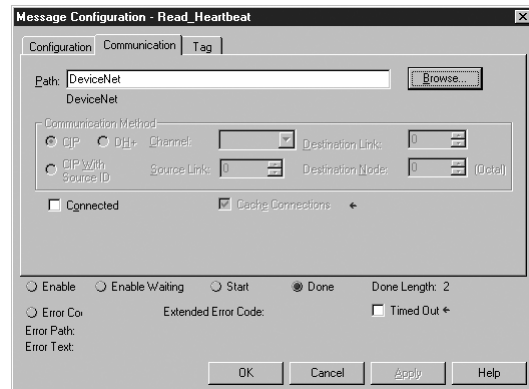
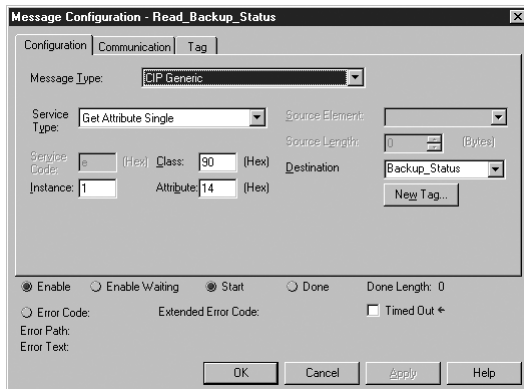


Table 11.1 describes the possible values this message may return when reading the back-up status of the DeviceNet scanner.

Table 11.2

If the message reads this value:	the back-up state of the DeviceNet scanner is:
0	No back-up scanner detected
1	Primary scanner forcing IDLE (back-up in RUN but primary in IDLE)

Using Indicators to Check Status

The 1788-DNBO card's status indicators provide useful information (e.g. determining which controller is primary) about back-up scanner status. Table 11.3 lists the indicators to monitor when checking back-up status.

Table 11.3

If this indicator	exhibits this behavior	this condition exists:
Module status (MS)	Flashing red	A secondary controller was not found (or other minor fault detected)
Back-up status (BS) ⁽¹⁾	Solid green	This scanner is the primary controller.
	Flashing green	This scanner is a qualified secondary controller.
	Off	This scanner is not configured for back-up mode.

⁽¹⁾ The BS status indicator may not be labelled on current 1788-DNBO communication cards.

Development and Debugging Tips

When you implement the DriveLogix Back-Up on DeviceNet solution, we recommend you consider the following development and debugging tips:

- Develop and debug the entire application with only the primary controller and scanner present. When the application is totally verified, then download the program and exact same scanlist to the secondary controller, without the primary controller present. Verify that the secondary is also functioning properly, and then both primary and secondary can be added to the network at the same time.
- No configuration parameters are entered from RSNetworkx for DeviceNet or RSLogix 5000 to enable Back-up. All configuration occurs in the user program. Almost your entire application (e.g. except for a few ladder rungs) can be developed without knowledge that the application will have a back-up controller and scanner.
- Local I/O still works when this solution is used but the Local I/O is not backed up.
- Switchover time depends on the user configurable heartbeat. After two heartbeats are lost between primary and secondary the switchover occurs. This time can be as little as 50ms with a heartbeat of 16ms.
- The I/O during switchover is NOT bumpless. Since the programs and I/O updates are not synchronized, it is possible for the secondary controller to be either slightly faster or slower than the primary.

For example, if output changes during a switchover, the fact that the primary and secondary controllers are unsynchronized can cause the

output to momentarily switch between an older and newer value. If you configure the switchover time slower than the program scan and I/O update, the secondary lags behind the primary and eliminates this.

- State variables, such as counters or timers are NOT synchronized. The user program must synchronize the primary and secondary controllers, typically over an EtherNet/IP or ControlNet link between controllers. If the outputs are dependent on a state variable, the lack of synchronization can also cause a bumpy switchover.
- As with all back-up and redundancy systems, the I/O must change at a slower rate than the switchover time. If the inputs change faster than the switchover, the change of state is lost.
- Either the user program or user action determine the primary controller. In its simplest mode, the first scanner to power-up or become available on DeviceNet first is the primary.
- Unlike some back-up systems (i.e. PLC5), the primary controller still maintain control of the I/O and switchover does NOT occur if the primary controller is set to Program/Idle mode. The secondary 1788-DNBO scanner also indicates that it is in Idle Mode.
- By default, a switchover will NOT occur if the default fault routine or user fault routine is executed in the primary controller. However, the user fault routine can force a switchover if so desired.
- If an operator interface is on DeviceNet, then it can work without knowledge which controller is primary or secondary.
- Online edits are not automatically performed on both Primary and Secondary since no synchronization exists between Primary and Secondary. Once an online edit occurs on the Primary, then the Primary and Secondary will have different programs.
- DriveLogix Back-up on DeviceNet is not Hot Back-up. Hot Back-up implies complete synchronization of program, program variables, and I/O. Also, I/O switchover is completely bumpless is Hot Back-up.



DriveLogix System Specifications

Using This Appendix

For information about:	See page
DriveLogix Controller	A-1
1756-BA1 Battery	A-3
DriveLogix Controller Serial Cables	A-4
DriveLogix Controller LEDs	A-6

DriveLogix Controller

Category:	DriveLogix 5720	DriveLogix 5720 with Memory Expansion
user memory	256k bytes	
FLEXBUS Local Rail current output	640 mA maximum @ 5.1V dc	
thermal dissipation	87 BTU/hour	
storage temperature	-40 to 70 degrees C (-40 to 158 degrees F)	
battery	1756-BA1 (Allen-Bradley PN 94194801)	
	0.59g lithium	
serial cable	1756-CP3 directly to controller	
	1747-CP3 directly to controller	
FLEXBUS Local I/O cable	4100-CCF3	

Category:	DriveLogix 5720	DriveLogix 5720 with Memory Expansion
Certifications: (when product is marked)	The drive is designed to meet the following specifications: NFPA 70 - US National Electric Code NEMA ICS 3.1 - Safety standards for Construction and Guide for Selection, Installation and Operation of Adjustable Speed Drive Systems. NEMA 250 - Enclosures for Electrical Equipment IEC 146 - International Electrical Code.	
		UL and cUL Listed to UL508C and CAN/CSA-C2.2 No. 14-M91
		Marked for all applicable European Directives (1) EMC Directive (89/336/EEC) Emissions EN 61800-3 Adjustable Speed electrical power drive systems Part 3 Immunity EN 61800-3 Second Environment, Restricted Distribution Low Voltage Directive (73/23/EEC) EN 60204-1 Safety of Machinery - Electrical Equipment of Machines EN 50178 Electronic Equipment for use in Power Installations

TIP

Refer to publication 20D-UM001 for PowerFlex 700S drive specifications. Environmental specifications for the host drive apply to the DriveLogix controller.

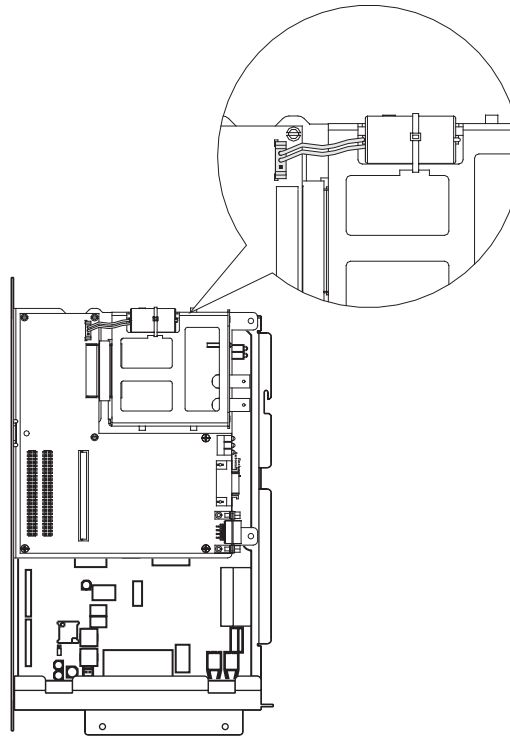
IMPORTANT

The amount of memory that the software displays includes both the user available memory and the memory reserved for overhead. See the specifications for your controller to determine how much memory you have available for programming. The software might display a higher number, but the additional memory is required by system overhead and may not be available for programming.

1756-BA1 Battery

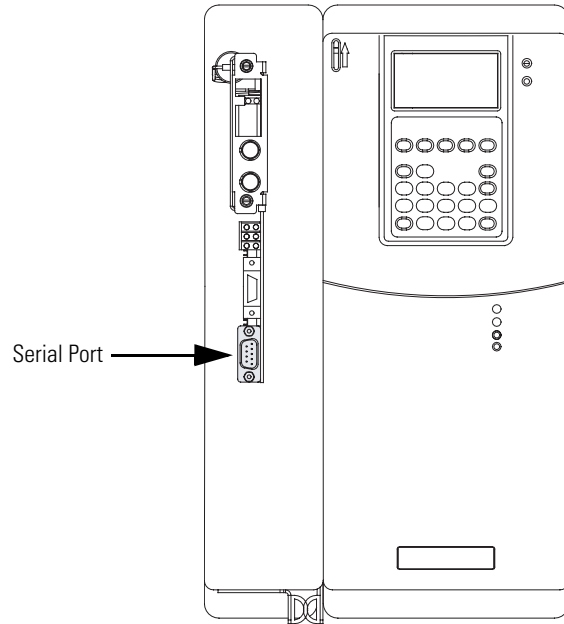
The DriveLogix controller uses the 1756-BA1 battery:

Battery	1756-BA1
	0.59g lithium



DriveLogix Controller Serial Cables

The RS-232 port is a non-isolated serial port built-in to the front of the controller.



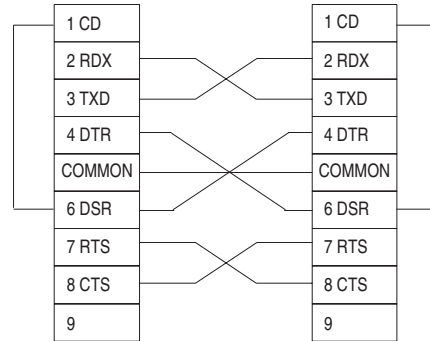
To connect to the serial port, determine whether you need an optical isolator. If you connect the controller to a modem or an ASCII device, consider installing an isolator between the controller and modem or ASCII device. An isolator is also recommended when connecting the controller directly to a programming workstation.

Are you using an isolator?

Use this cable:

No

The 1756-CP3 cable attaches the controller directly to the controller.

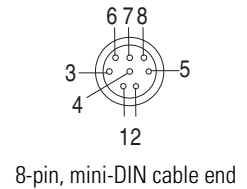
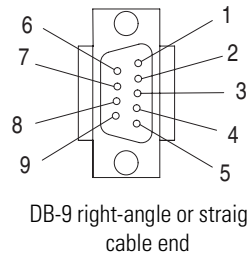


If you make your own cable, it must be shielded and the shields must be tied to the metal shell (that surrounds the pins) on both ends of the cable.

You can also use a 1747-CP3 cable (from the SLC product family). This cable has a taller right-angle connector housing than the 1756-CP3 cable.

Yes

The 1761-CBL-AP00 cable (right-angle connector to controller) or the 1761-CBL-PM02 cable (straight connector to the controller) attaches the controller to port 2 on the 1761-NET-AIC isolator. The mini-DIN connector is not commercially available, so you cannot make this cable.



Pin:	DB-9 end:	Mini-DIN end:
1	DCD	DCD
2	RxD	RxD
3	TxD	TxD
4	DTR	DTR
5	ground	ground
6	DSR	DSR
7	RTS	RTS
8	CTS	CTS
9	N/A	N/A

DriveLogix Controller LEDs

Type	Name	Color	State	Description	
Daughtercard LEDs	PORT	Green	–	Status of DPI port internal communications (if present).	
	MOD	Yellow	–	Status of communications module (when installed).	
	NET A	Red	–	Status of network (if connected).	
	NET B	Red	–	Status of secondary network (if connected).	
	NET A		OFF		No power, Host is faulted, Host is holding daughtercard in reset.
			Red	Steady	Major Fault
			Red	Flashing	Minor Fault
			Green	Flashing	No connections established
			Green	Steady	Connections established
	NET B		OFF		No power, Host is faulted, Host is holding daughtercard in reset.
			Red	Steady	Major Fault
			Red	Flashing	Minor Fault
			Green	Flashing	No connections established
			Green	Steady	Connections established
	Controller LEDs	RUN	Green	Off	No task(s) running. Controller in Program mode.
			Green	Steady	One or more tasks are running. Controller is in Run mode.
FORCE		Amber	Off	No forces present.	
		Amber	Steady	Forces present and enabled.	
		Amber	Flashing	Forces present but not enabled.	
BATT		Red	Off	Battery will support memory.	
		Red	Steady	Battery may not support memory - replace battery.	
I/O			Green	Off	Controller project not downloaded (the condition after power up). No I/O or communications configured.
			Green	Steady	Communicating to all devices.
			Green	Flashing	One or more devices are not responding.
			Red	Flashing	No required I/O connections can be made, controller is in Run mode.
RS 232		Green	Off	No activity.	
		Green	Flashing	Data being received or transmitted.	
OK		Off		No power applied.	
		Red	Flashing	Recoverable fault or flash programming.	
		Red	Steady	Controller faulted. Clear faults, clear memory, or replace the controller.	
		Green	Steady	Controller OK.	

Installing and Maintaining the Battery

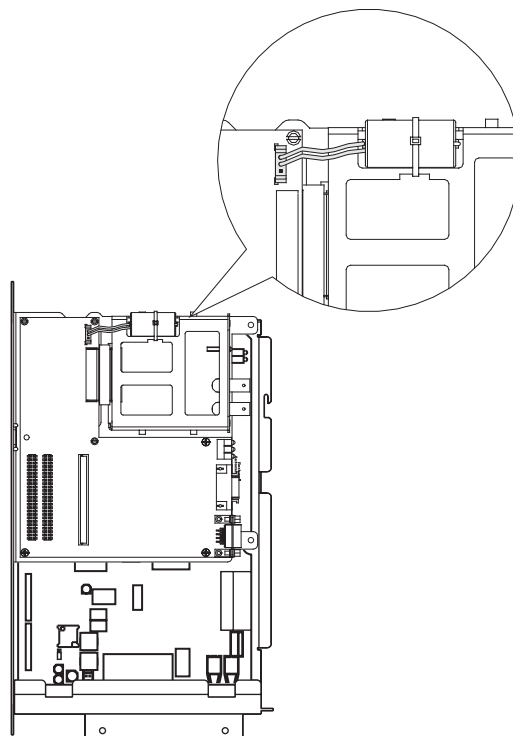
Using this Appendix

For information about:	See page
Installing and Maintaining the Battery	B-1
Storing replacement batteries	B-2
Estimating battery life	B-2
Replacing batteries	B-3

Connecting the Battery

Allen-Bradley ships the DriveLogix controller with the battery installed, but disconnected. You must connect the battery while installing the drive:

1. Referring to Access Procedures on page C-1, remove the cover(s) to gain access to the DriveLogix controller.
2. Connect the plug on the battery lead into the socket on the DriveLogix controller.



3. Referring to Access Procedures on page C-1, replace the cover(s)

Storing Replacement Batteries

Because a battery may leak potentially dangerous chemicals if stored improperly, store batteries as follows:

ATTENTION



Store batteries in a cool, dry environment. We recommend 25° C with 40% to 60% relative humidity. You may store batteries for up to 30 days between -45° to 85° C, such as during transportation. To avoid possible leakage, *do not* store batteries above 60° C for more than 30 days.

Estimating Battery Life

When the battery is about 95 percent discharged, the controller provides the following warnings:

- On the front of the controller, the BATTERY LED turns on (solid red).
- A minor fault occurs (type 10, code 10).

To prevent the battery from leaking potentially dangerous chemicals, replace the battery at least as often as:

ATTENTION



To prevent possible battery leakage, even if the BATTERY LED is off, replace the battery according to this schedule:

If the temperature 1 in. below the controller is:	Replace the battery within:
0° to 35° C	No required replacement
36° to 40° C	3 years
41° to 45° C	2 years
46° to 50° C	16 months
51° to 55° C	11 months
56° to 60° C	8 months

To estimate how long the battery will support the memory of the controller:

1. Determine the temperature (° C) 1 in. below the DriveLogix controller.

- Determine the percentage of time that the controller is powered off per week.

EXAMPLE

If a controller is off:

- 8 hr/day during a 5-day work week
- all day Saturday and Sunday

Then the controller is off 52% of the time:

- total hours per week = $7 \times 24 = 168$ hours
- total off hours per week = $(5 \text{ days} \times 8 \text{ hrs/day}) + \text{Saturday} + \text{Sunday} = 88$ hours
- percentage off time = $88/168 = 52\%$

Use the off-time percentage you calculated with the following table to determine battery life:

Temperature:	Worst-case battery life estimate:		
	Power off 100%:	Power off 50%:	Battery duration after the LED turns on: ⁽¹⁾
60° C	2 months	3.5 months	3 days
25° C	2 months	4 months	3 days

⁽¹⁾ The battery indicators (BATTERY) warns you when the battery is low. These durations are the amounts of time the battery will retain controller memory from the time the controller is powered down after the LED first turns on.

IMPORTANT

If the BATTERY LED turns on when you apply power to the controller, the battery life may be less than the table above indicates. Some of the warning time may have been used while the controller was off and unable to turn on the BATTERY LED.

Replacing a Battery

Because the controller uses a lithium battery, you must follow specific precautions when handling or disposing a battery.

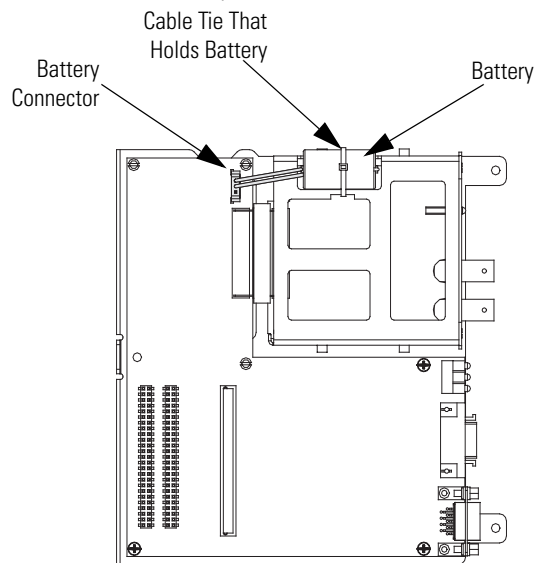
ATTENTION

The controller uses a lithium battery, which contains potentially dangerous chemicals. Before handling or disposing a battery, review *Guidelines for Handling Lithium Batteries*, publication AG-5.4.

1. Remove the front cover of the drive's control assembly.
2. Upload the controller's memory and program to a computer with RSLogix 5000 programming software.
3. Turn off power to the DriveLogix controller.
4. Remove the side cover of the drive's control assembly (not necessary for high power drives).
5. Remove drive's control assembly, if necessary.
6. Does the existing battery show signs of leakage or damage?

If:	Then:
Yes	Before handling the battery, review <i>Guidelines for Handling Lithium Batteries</i> , publication AG-5.4.
No	Go to the next step.

7. Remove the old battery, by unplugging the battery and cutting the cable tie that holds the battery.



8. Install a new 1756-BA1 battery.

ATTENTION

Only install a 1756-BA1 battery. If you install a different battery, you may damage the controller.



9. Attach the battery label. Write on the battery label the date you install the battery.

10. Secure the new battery by installing a new cable tie.
11. Re-install the control assembly, if removed.
12. Turn on power to the DriveLogix controller.
13. On the front of the controller, is the BATTERY LED off?

If:	Then:
Yes	Go to the next step.
No	A. Check that the battery is correctly connected to the controller. B. If the BATTERY LED remains on, install another 1756-BA1 battery. C. If the BATTERY LED remains on after you complete Step B., contact your Rockwell Automation representative or local distributor.

14. Download the controller's memory and program from the computer with RSLogix 5000 programming software.
15. Turn off power to the DriveLogix controller.
16. Re-install the cover(s).
17. Dispose the old battery according to state and local regulations.

ATTENTION

Do not incinerate or dispose lithium batteries in general trash collection. They may explode or rupture violently. Follow state and local regulations for disposal of these materials. You are legally responsible for hazards created while your battery is being disposed.

Notes:

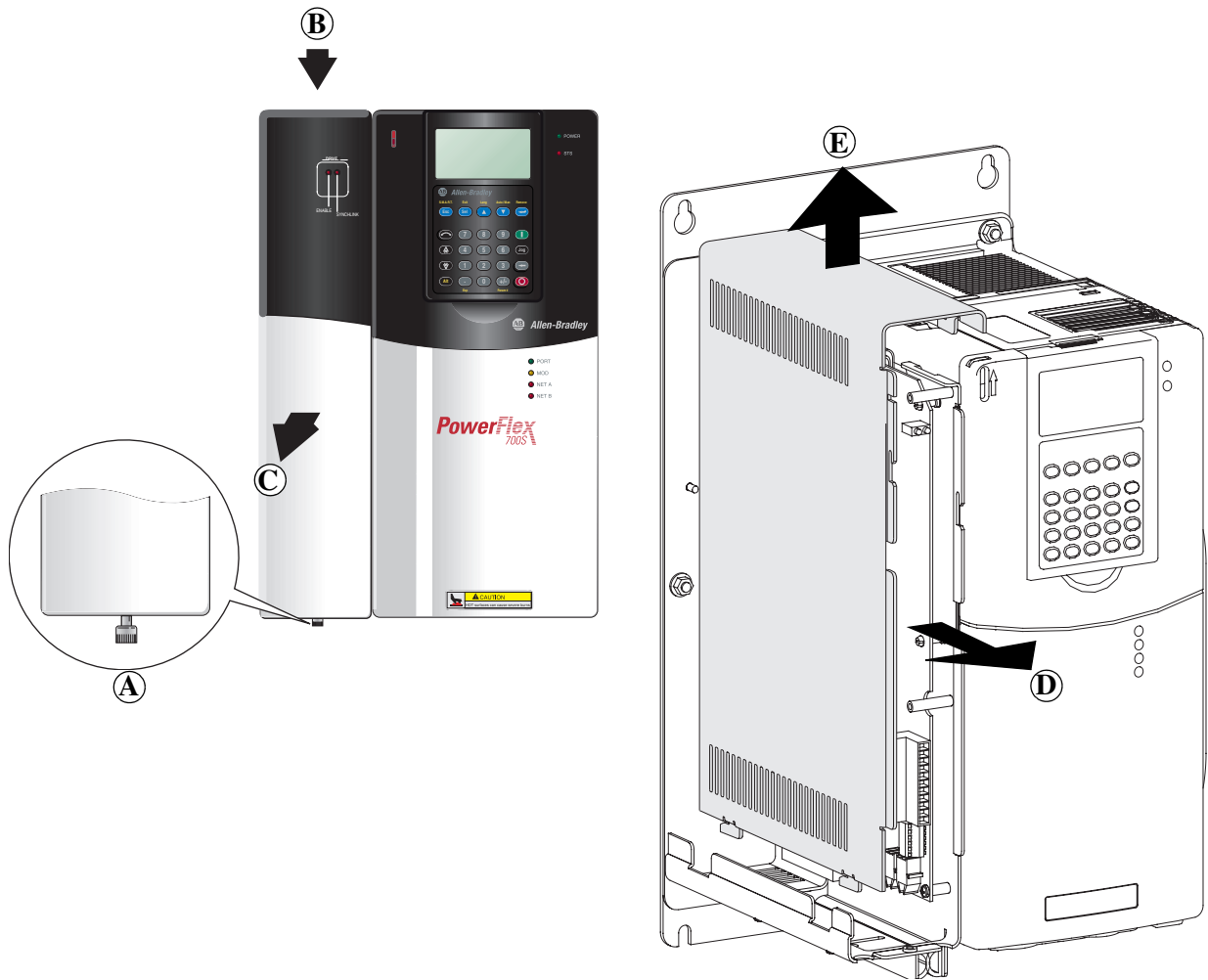
Access Procedures

Using this Appendix

For information about:	See page
Removing Cover(s)	C-2
Removing Cover (For High Power Drives)	C-3
Replacing Cover(s)	C-4
Replacing Cover (For High Power Drives)	C-6

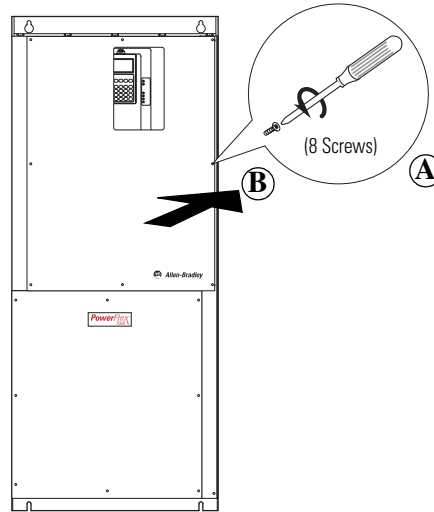
Removing Cover(s)

Task	Description
(A)	Loosen captive screw
(B)	Push down on front cover
(C)	Pull front cover away from assembly
(D)	Pull side cover forward
(E)	Lift side cover off of control assembly



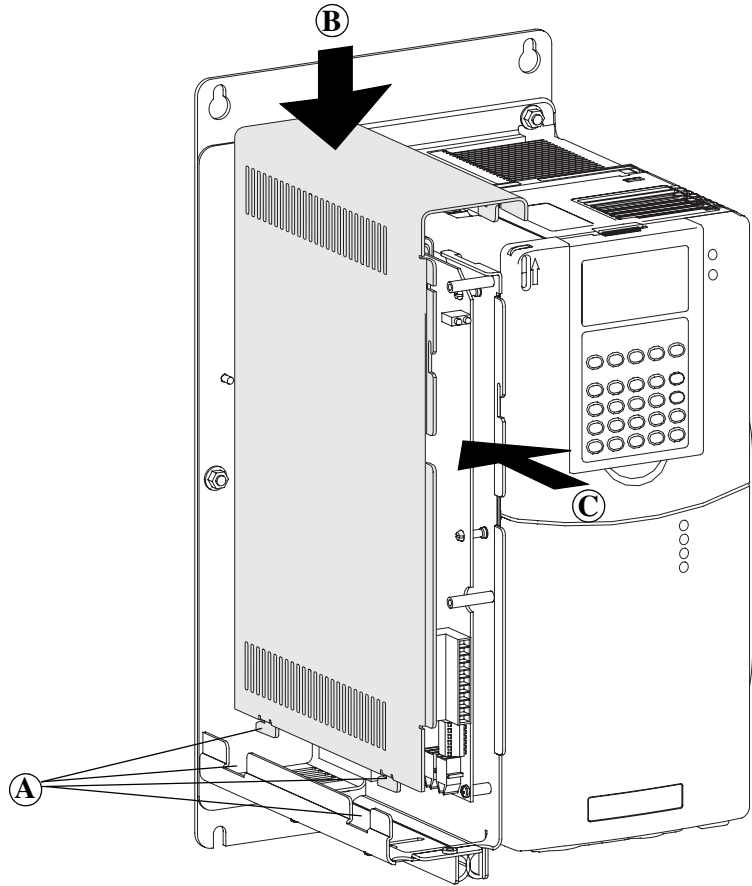
Removing Cover (For High Power Drives)

Task	Description
Ⓐ	Remove eight (8) screws
Ⓑ	Remove power cover (top cover)



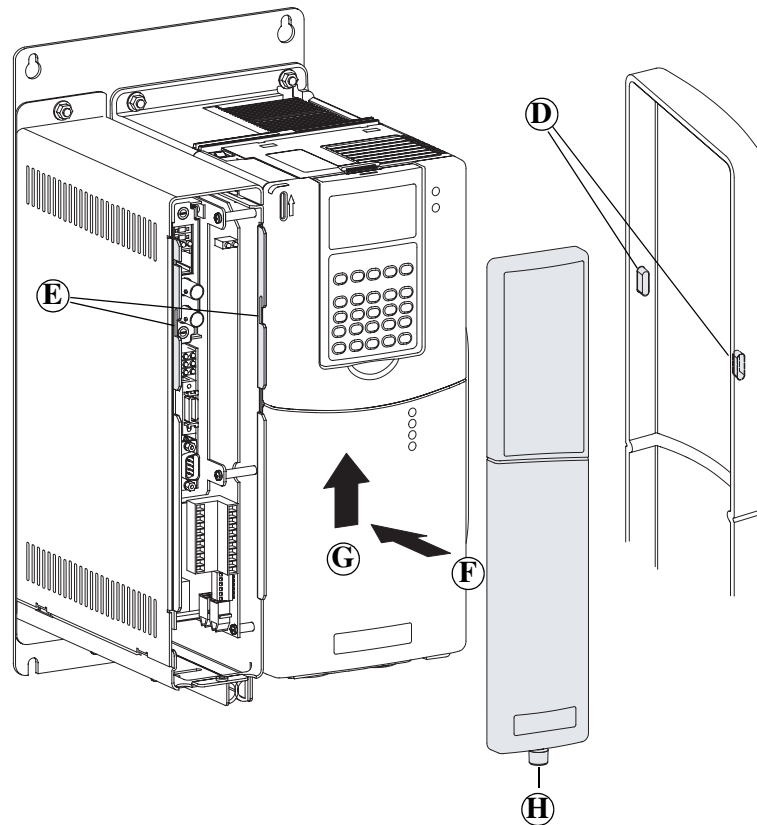
Replacing Cover(s)

Task	Description
(A)	Align tabs on side cover with slots on drive
(B)	Push side cover down onto control assembly
(C)	Push side cover back onto control assembly



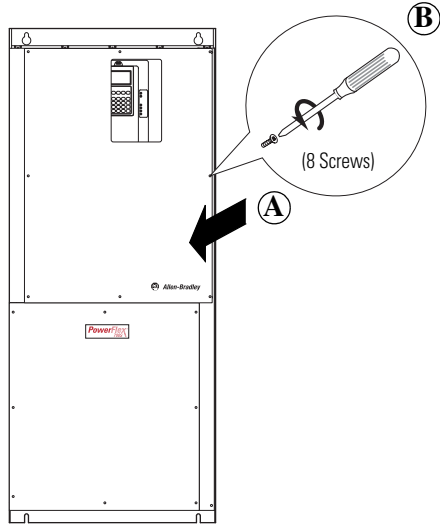
Replacing Cover(s) Continued

Task	Description
Ⓓ	Locate tabs on inside of front cover
Ⓔ	Align tabs on front cover with slots on flanges
Ⓕ	Push front cover onto drive
Ⓖ	Push front cover up into slots
Ⓗ	Tighten captive screw



Replacing Cover (For High Power Drives)

Task	Description
(A)	Install power cover (top cover)
(B)	Install eight (8) screws



Numerics

1756-BA1 B-1
1788-CN2DN 8-11
1788-CNCx 7-1
1788-DNBO 8-1
1788-ENBT 6-1
1788-MODULE 11-1

A

adding

local analog module 1-12
 local output module 1-10

alias

defining 4-16
 getting started 1-17

analog module

adding 1-12

ASCII protocol 9-15

B

battery B-1

how to replace B-3
 life B-2
 storage B-2
 when to replace B-2

C

changing

module properties 1-14
 project properties 1-5

commands

motion 5-12
 motion event 5-12
 motion group 5-12
 motion move 5-12
 motion state 5-12

communicating

ControlNet 7-1
 DeviceNet 8-1
 DH-485 10-1
 EtherNet/IP 6-1
 mapping address 6-18, 7-15
 serial 9-1
 with other controllers 6-15, 7-13
 with other Logix-based controller 6-14,
 7-12

communication card

ControlNet 7-4
 DeviceNet 8-3
 EtherNet/IP 6-7

installing (ControlNet) 7-2
 installing (DeviceNet) 8-2
 installing (EtherNet/IP) 6-2
 installing (Third Party Link) 11-2
 Third-Party Link 11-2

communication driver

ControlNet 7-3
 serial 9-7

communication format 3-1, 3-6, 4-8

Custom User-Defined Control 3-1, 3-11
 mapping 3-2
 Motion Control 3-1, 3-10
 Position Control 3-1, 3-8
 User-Defined 3-1
 User-Defined Control 3-9
 Velocity Control 3-1, 3-7

configuring

alias 4-16
 ASCII protocol 9-15
 communication format 4-8
 ControlNet system 7-1
 DeviceNet system 8-1
 DF1 master 9-11
 DF1 point-to-point 9-8
 DF1 slave 9-11
 DH-485 system 10-1
 DIN rail 4-5
 drive for motion 5-2
 electronic keying 4-7
 EtherNet/IP system 6-1
 host PowerFlex 700S 1-6, 3-4
 inhibit I/O module 4-10
 local I/O 4-6
 motion 5-1
 remote devices 6-8, 7-5
 response to connection failure 3-24,
 4-16
 serial system 9-1
 third party link 11-1

configuring Ethernet communication

drivers ??-6-6

AB_ETH driver 6-3-6-6

connection

ControlNet guidelines 7-21
 EtherNet/IP guidelines 6-23
 I/O module 2-12
 monitoring rack-optimized 4-19
 requirements 2-17
 response to failure 4-16

connection failure 3-24

controller ownership 4-8

controller updates 3-3

ControlNet

- accessing remote devices 7-7
- communication card 7-4
- communication driver 7-3
- configuring the system 7-1
- connection guidelines 7-21
- consuming a tag 7-20
- example DriveLogix Controller and Remote I/O 7-22
- example DriveLogix Controller to DriveLogix Controller 7-24
- example DriveLogix Controller to Other Devices 7-28
- hardware 7-2
- installing communication card 7-2
- mapping address 7-15
- message to other controller 7-13
- message to other Logix-based controller 7-12
- overview 7-1
- produced/consumed tag 7-17
- producing a tag 7-19
- remote devices 7-5
- schedule network 7-10
- sending messages 7-11

cover

- removing C-2
- removing (high power drives) C-3
- replacing C-4

cover (high power drives)

- replacing C-6

creating

- project 1-3, 1-4
- tags 1-16

D**data** 4-13**DeviceNet**

- scan list 8-4

developing

- programs 2-2

DeviceNet

- accessing remote devices 8-6
- communication card 8-3
- configuring the system 8-1
- DriveLogix back-up on the network 12-1–12-14
- example using a 1788-CN2DN linking device 8-11
- hardware 8-2
- installing communication card 8-2
- overview 8-1

DF1 protocol

- master 9-5, 9-11
- master/slave methods 9-10
- point-to-point 9-5, 9-8
- slave 9-5, 9-11

DH-485

- configuring the port 10-3
- configuring the system 10-1
- example network configuration 10-9
- grounding 10-9
- hardware 10-1
- installing 10-6
- network initialization 10-5
- nodes 10-5
- overview 10-1
- terminating 10-9
- token rotation 10-4

DIN rail

- configuring 4-5

documenting I/O 1-17**downloading**

- project 1-3, 1-20, 2-20

drive

- revision 3-5

DriveExecutive 3-15, 3-21**E****electronic keying** 3-5, 4-7**entering**

- logic 1-18

EtherNet/IP

- accessing remote devices 6-10
- communication card 6-7
- configuring the system 6-1
- connection guidelines 6-23
- consuming a tag 6-22
- example DriveLogix controller and remote devices 6-23
- example DriveLogix controller to other devices 6-29
- hardware 6-2
- installing communication card 6-2
- mapping address 6-18
- message to other controller 6-15
- message to other Logix-based controller 6-14
- overview 6-1
- produced/consumed tag 6-20
- producing a tag 6-21
- remote devices 6-8
- sending messages 6-13

event tasks 2-6–2-8**example**

DH-485 configuration 10-9
 DriveLogix controller and remote devices
 over EtherNet/IP 6-23
 DriveLogix Controller and Remote I/O on
 ControlNet 7-22
 DriveLogix Controller to DriveLogix
 Controller on ControlNet 7-24
 DriveLogix Controller to Other Devices on
 ControlNet 7-28
 DriveLogix controller to other devices
 over EtherNet/IP 6-29
 monitoring I/O module 4-18
 monitoring rack-optimized connection
 4-19
 using a 1788-CN2DN linking device 8-11

F

fault bit 4-17

G

getting started

adding a local analog module 1-12
 adding a local output module 1-10
 changing module properties 1-14
 changing project properties 1-5
 configuring the host PowerFlex 700S 1-6,
 3-4
 creating a project 1-4
 creating tags 1-16
 documenting I/O with alias tags 1-17
 downloading a project 1-20
 entering logic 1-18
 overview 1-1
 steps 1-3
 viewing controller memory usage 1-22
 viewing I/O tags 1-15
 viewing scan time 1-21

H

hardware

ControlNet 7-2
 DeviceNet 8-2
 DH-485 10-1
 EtherNet/IP 6-2
 serial 9-2
 Third-Party Link 11-2

host PowerFlex 700S

configuring 1-6, 3-4
 placing 3-4

I

I/O configuration

FLEX I/O adapter 6-9-??
 local ENBT module 6-8-??

I/O module

alias 4-16
 communication format 4-8
 configuring local 4-6
 connection 2-12
 DIN rail 4-5
 electronic keying 4-7
 example logic for monitoring a
 rack-optimized connection 4-19
 example logic for monitoring the module
 4-18
 fault bits 4-17
 local overview 4-1
 monitoring 4-16

inhibit operation

Drive Connection 3-13
 I/O module 4-10

installing

communication card (ControlNet) 7-2
 communication card (DeviceNet) 8-2
 communication card (EtherNet/IP) 6-2
 communication card (Third Party Link)
 11-2
 cover C-4
 cover (high power drives) C-6

K

keying

drive 3-5

keying, electronic 4-7

3-5

L

local I/O

configuring 4-6
 DIN rail 4-5
 overview 4-1

logic

entering 1-18

Logix environment 2-1

low battery B-2

M

mapping address 6-18, 7-15

master/slave communication 9-10

memory

controller usage 1-22
nonvolatile 2-20

message

sending over ControlNet 7-11
sending over EtherNet/IP 6-13
to other controller 6-15, 7-13
to other Logix-based controller 6-14,
7-12

monitoring

example logic for I/O module 4-18
example logic for rack-optimized
connection 4-19
I/O module 4-16
rack-optimized connection 4-19

motion

configuring 5-1
configuring host drive 5-2
motion event 5-12
motion group 5-12
motion move 5-12
motion state 5-12
programming 5-5
supported commands 5-12
system requirements 5-1

motion event

supported commands 5-12

motion group

supported commands 5-12

motion move

supported commands 5-12

motion state

supported commands 5-12

N

nonvolatile memory 2-20

O

output module

adding 1-10

P

placing

host PowerFlex 700S 3-4

priority 2-3

produced/consumed tag

overview 6-20, 7-17

program

defining 2-5
developing 2-2

programming

for motion 5-5

project

creating 1-3, 1-4
developing 2-2
downloading 1-3, 1-20
program 2-5
properties 1-5
routine 2-5
task 2-3

R

rack optimized I/O

add FLEX I/O adapter to I/O configuration
6-9-??
add local ENBT module to I/O
configuration 6-8-??

remote devices

accessing over ControlNet 7-7
accessing over DeviceNet 8-6
accessing over EtherNet/IP 6-10
configuring over ControlNet 7-5
configuring over EtherNet/IP 6-8

removing

cover C-2
cover (high power drives) C-3

replace the battery

how B-3
when B-2

replacing

cover C-4
cover (high power drives) C-6

Requested Packet Interval 3-4

revision

drive 3-5

routine

defining 2-5

RPI 3-4

RSLinx communication drivers ??-6-6

AB_ETH driver 6-3-6-6

S

scan list 8-4

scan time 1-21

schedule network 7-10

serial

ASCII protocol 9-15
communication driver 9-7
configuring the port 9-4
configuring the system 9-1
hardware 9-2
master 9-11

overview 9-1
point-to-point 9-8
slave 9-11

slave/master communication 9-10

store batteries B-2

supported motion commands 5-12

system requirements

motion 5-1

T

tag

alias 4-16
consuming 6-22, 7-20
creating 1-16
names 4-13
produced/consumed overview 6-20,
7-17
producing 6-21, 7-19
sample alias 1-17
viewing 1-15

task

defining 2-3

priority 2-3

Third Party Link

installing communication card 11-2

third party link

configuring the system 11-1

Third-Party Link

communication card 11-2

third-party link

hardware 11-2

V

viewing

Communication Interface to the
Controller 3-18

controller memory usage 1-22

I/O tags 1-15

scan time 1-21

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444

Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640

Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 20D-UM002C-EN-P - November 2003

Supersedes 20D-UM002B-EN-P March 2003

Copyright © 2003 Rockwell Automation. Printed in the U.S.A.



Allen-Bradley

DriveLogix System

User Manual