EtherNet/IP™ Programming

#33 Programming with RSLogix 5000

7/3/2014
Aerotech Inc.
www.aerotech.com

CONTENTS

ETHERNET/IP™

EtherNet/IP™ (Ethernet Industrial Protocol) is a communications protocol developed by Rockwell Automation, and is currently managed by the Open DeviceNet Vendors Association (ODVA). It is designed for use in industrial automation applications.

The Aerotech Ensemble and Soloist controllers have an EtherNet/IP™ plugin that can be installed. The controllers have 4 tasks. This plugin runs on Task 3 of the controller. It also takes up the Task 4 slot for communications. This is not inherent in the interface. Therefore, user programs can run in Task 1 and 2.

ODVA requires that users be registered. By doing this a vendor ID is created that can be read over an EtherNet/IP™ program to verify that the component is licensed. The Aerotech Soloist and Ensemble are ODVA certified. The Aerotech EtherNet/IP™ Vendor ID issued by ODVA is 935 for these controllers.

ETHERNET/IP™ PROGRAMMING MODES

Aerotech supports two main programming modes, Class 1 IO and Class 3 Messaging.

A Class 1 connection establishes a periodic exchange of data between the RSLogix module and the Ensemble. The Ensemble lets you access the global integer registers and global double registers through this Class 1 connection. This interface uses the registers as I/O data. The registers are transmitted and received at a fixed time interval that is set by the scanner device. Use this interface to send and receive data that requires a fast response time.

This connection request also establishes the number of global registers being written to, and the size of these registers. For compatibility with third-party PLC programming software, values that are read from and written to the global double registers are limited to 32-bit single-precision floating point values. Global double registers are 64-bit double-precision floating point values. These values are rounded to 32-bit single-precision floating point values in the Class 1 I/O Interface.

A Class 3 connection is used for individual request/response transactions. A request from an RSLogix module always results in a response from the Ensemble indicating the success or failure of the request. The response may also include status information. The destination of the service request is defined by a portion of the service request packet called the path. The path is either a literal ASCII character string or an object description. The Ensemble receiving the service request can distinguish between an ASCII character string path and an object description path by header bytes inside the path. A request to an object is identified inside the path by its class number, instance number, and attribute number. Class identifies which type of object is being referenced, and instance defines the particular object of that type.

The Ensemble has two modes for this Class 3 connection – ASCII Command Interface Object and Register Interface Object. The ASCII Command Interface Object is used to send AeroBasic commands directly to the controller. The Register Interface Object is used to manually set individual IGLOBAL and DGLOBAL registers in the Ensemble.

ETHERNET/IP™ PROGRAMMING RSLOGIX 5000

Often machine or line control requires both a separate PLC and motion controller. The Ensemble, a multi-axis coordinated motion controller, can be used to interface with an Allen-Bradley L32E PLC using standard EtherNet/IP™. This application note demonstrates how to set up the L32E PLC and the Ensemble in a Master/Slave configuration. The Ensemble is EtherNet/IP™ certified. If you are using an Allen-Bradley Logix (and RS Logix 500), refer to the application note "Using Ethernet/IP™ with RS Logix 500".

When the PLC module is in PROG mode the PLC is no longer running. Ensure that conditions are safe before you put your module into this mode.

In this demonstration we will go through the configuration procedure to send commands to the Ensemble over EtherNet/IP™ through the RSLOGIX 5000 software.

This example uses a CompactLogix L32E PLC and an Ensemble controller with the EtherNet/IP™ plugin installed and running.

The CompactLogix L32E IP address must be configured to be within the first 3 octets of the IP address on the Ensemble, and should have the same subnet mask of the Ensemble.

Example:

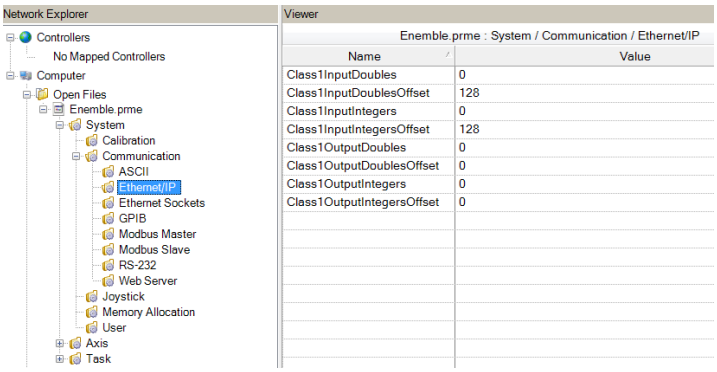
CompactLogix L32E	IP address: 192.168.254.200	Subnet mask: 255.255.255.0
Ensemble	IP address: 192.168.254.100	Subnet mask: 255.255.255.0

CLASS 1 I/O

There are two Class 1 connections that are available, one for global integer registers, IGLOBAL(*n*), and one for global double registers, DGLOBAL(*n*).

Connection	Input Instance	Output Instance	Data Type	Despcription
IGLOBAL	100	150	DINT	Reads and writes arrays of global integer registers.
DGLOBAL	101	151	REAL	Reads and writes arrays of global double registers.

The size of the Class 1 data packet is based on the number of registers to be transmitted. Use the EtherNet/IP™ parameters to specify the starting register offsets and the number of registers to transmit.



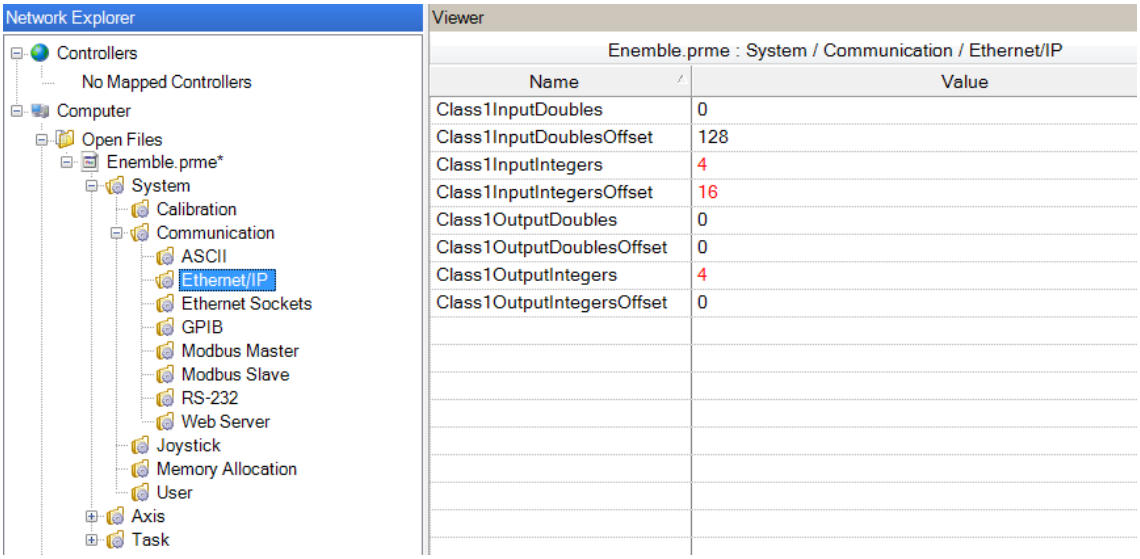
The Class1Input parameters are used to specify the number of registers that you want to receive from the remote scanner of the Class 1 I/O register interface.

The Class1Output parameters are used to specify the number of registers that you want to send to the remote scanner of the Class 1 I/O register interface.

For compatibility with third-party PLC programming software, values that are read from and written to the global double registers are limited to 32-bit single-precision floating point values. Global double registers are 64-bit double-precision floating point values. These values are rounded to 32-bit single-precision floating point values in the Class 1 I/O interface.

The following example transmits global integer registers 0 through 3 (IGLOBAL(0) through IGLOBAL(3)) to the remote scanner, and receives global integer registers 16 through 19 (IGLOBAL(16) through IGLOBAL(19)) from the PLC.

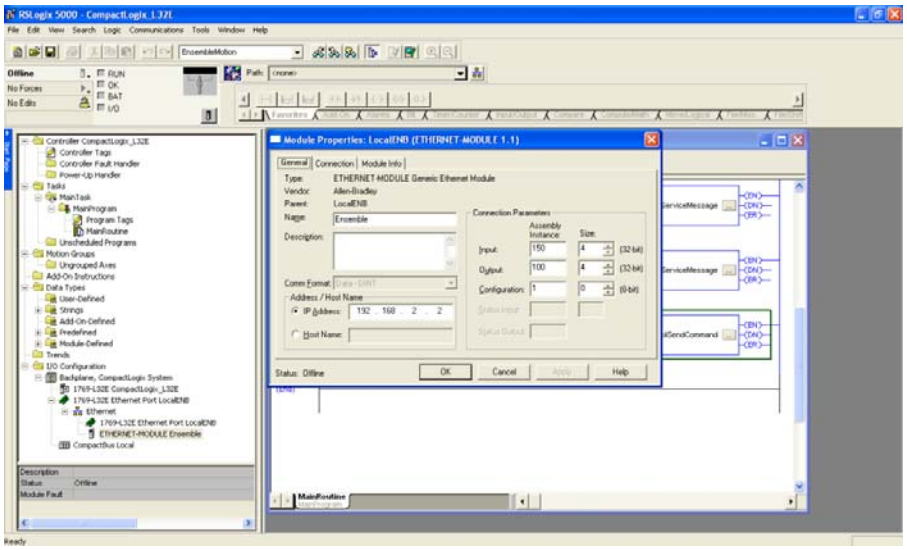
1. In the Configuration Manager, configure the following EtherNet/IP™ parameters to the following settings.
    A. Class1InputIntegers = 4
    B. Class1InputIntegersOffset = 16
    C. Class1OutputIntegers = 4
    D. Class1OutputIntegersOffset = 0



2. Send and commit the parameters to the controller, then reset the controller for these changes to take effect.
3. In the RSLogix software, add a new ethernet module.

    A. In the I/O Configuration folder of the left-hand pane, right-click Ethernet and select New Module. The Select Module dialog comes into view.

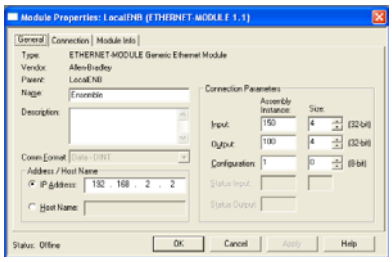
    B. In the Communications drop-down of the Select Module dialog, select ETHERNET-MODULE Generic Ethernet Module as the module type. The Module Properties dialog comes into view.



    C. In the Module Properties dialog, enter the following information:

In the Name section, change the name of the module to a name that you want to use.

In the Description section, type a description for the module.



In the Comm Format drop-down, select the communication format data as Data - DINT.

In the Address/Host Name section, select IP Address and type the IP address of the controller.

In the Connection Parameters section, change the Assembly Instance to 150 for the Input, 100 for the Output, and 1 for the Configuration.

In the Connection Parameters section, change the Size to 4 for the Input size, 4 for the Output size, and 0 for the Configuration size.

Save the module settings.

To test this global integer register configuration, do the following.

1. In Motion Composer, select the Register Manager tab. Here, you can change global integer registers 0 through 3 and read global integer registers 16 through 19.
2. In the Integer Register Reader section of the Register Manager tab, click the Set button. The Register Range Selector dialog comes into view.
3. In the Register Range Selector dialog, set the range of the global integer registers to a Minimum range of 0 and a Maximum range of 19.
4. To close this dialog, click OK. In the Integer Register Reader section of the Register Manager tab, you should now see the range that you set.
5. For global integer registers 0 through 3, change one of these values.
    A. In the Values field, double-click the value that you want to change.
    B. Type a new value.
    C. Press Enter.
6. In the Controller Tags in RSLogix, view the array named NameOfModule:I.Data. This array should hold the values that you changed.
7. In the Controller Tags in RSLogix, view the array named NameOfModule:O.Data.
8. In the NameOfModule:O.Data array, change the values.
9. In the Register Manager tab of Motion Composer, click the Refresh button to load the values of the registers. You should see that global integer registers 16 through 19 now show the values from RSLogix.

CLASS 3 MESSAGING

Aerotech supports the ASCII Command Interface Object and the Register Interface Object of the Class 3 Messaging.

CLASS 3 ASCII

Use the ASCII Command Interface to send ASCII text strings to the controller and to perform a set of actions such as commanding motion or retrieving diagnostic information. The ASCII Command Interface object extends this functionality to EtherNet/IP™.

The String format used by the ASCII Command Interface object consists of a 4-byte DINT (specifying the string length) followed by the number of characters specified in the DINT. A String is not to be null-terminated and cannot be more than 82 characters long.

The ASCII Command service parameter is set to an AeroBasic command, followed by an End-Of-String (EOS), 0xA, character.
The ASCII Response data begins with a single-character response code that shows the status of the command:

An ACK character, %, is sent to indicate success.
A NAK character, !, is sent if there is a command error.
A FAULT character, #, is sent if there is a task error.

For commands that expect return data, the response character is followed by the return data, which is terminated by the EOS character.
NOTE: The EOS, ACK, NAK, and FAULT characters are configurable using drive parameters. The default values are referred to above.

There are no Class Attributes to the ASCII Interface. There is one Instance Attribute, 0x1.

Number	Access Rule	Name	Data Type	Description
0x64 (100)	Get	Command String	String	Returns the most recently received ASCII Command.
0x65 (101)	Get	Response String	String	Returns the most recently produced ASCII Response.

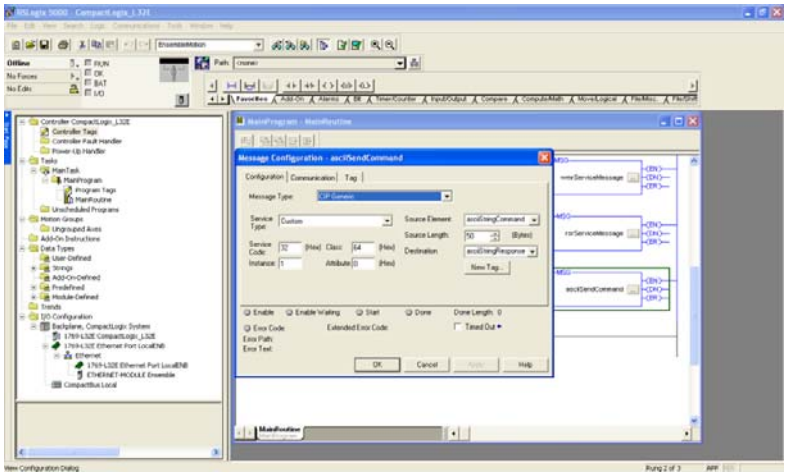
There is one Common Service.

Service Code	Name	Description
0xE (14)	Get_Attribute_Single	Returns the contents of the specified attribute.

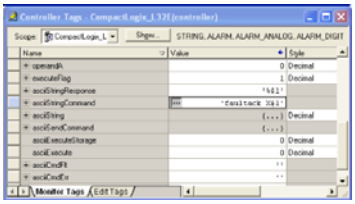
There is one Object Specific Service.

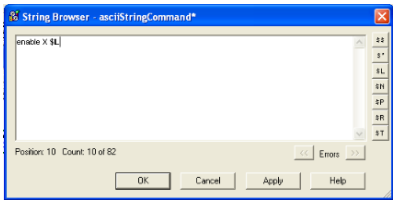
Service Code	Name	Description
0x32 (50)	Execute_Command	Executes a command through the ASCII Command Interface.

Instance Service	Name	Type	Description
Service Parameter	ASCII Command	String	ASCII command to be executed.
Service Response Data	ASCII Response	String	Response from the ASCII Command Interface.

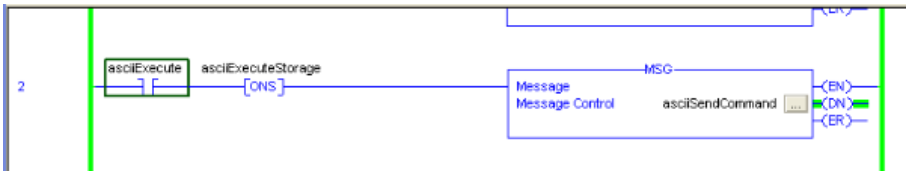


Highlight the asciiStringCommand controller tag that was created and click on the button to the left of the value. This will bring up the String Browser. From here you can enter your AeroBasic command. Ensure that you add the line feed character at the end of your string, $L. This is entered by clicking on the $L button to the right of the editor. Then press OK.

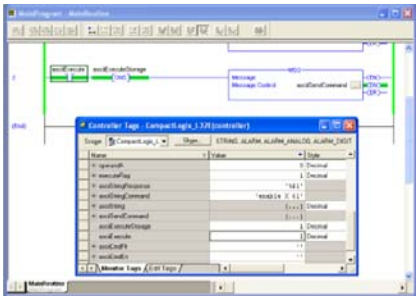




When asciiExecute is set to a 1 this will execute the message command.



The asciiStringResponse will then return the appropriate character from the controller.



CLASS 3 REGISTER

Use the register interface to access the Integer and Double Registers of the controller through EtherNet/IP™. This provides a flexible, general-purpose interface between the controller and EtherNet/IP™ that can be adapted to many different applications.

The register interface instance services allow the reading and writing of the Integer and Double Registers of the controller over EtherNet/IP™. Data consistency is guaranteed internally, so you can access these registers at the same time through EtherNet/IP™ and task programs on the controller.

For compatibility with third-party PLC programming software, values read from and written to the Double Registers are limited to 32-bit single-precision floating point values. When reading a Double Register, the 64-bit double-precision floating point value is rounded to a 32-bit single-precision floating point value when it is returned through the Register Interface Object.

The Register Interface object does not have class attributes.

There are two instances of the register interface object.

- Instance 0x01 is mapped to the Integer Registers of the controller.
- Instance 0x02 is mapped to the Double Registers of the controller.

Instance Attributes:

Number	Access Rule	Name	Data Type	Description
0x64	Get	Number of Registers	DINT	Returns the number of registers in this set.
0x65	Get	Register Size	DINT	Returns the size of each register, in bytes.
0x66	Get	Register Set Base Address	DINT	Returns the 32-bit base memory address of the register set.
0x67	Get	Register Set Mutex	DINT	Returns the mutex number associated with the register set.

There is one common service:

Service Code	Name	Description
0x0E	Get_Attribute_Single	Returns the contents of the specified attribute.

There are four object specific services:

Service Code	Name	Description
0x32	Read_Single_Register	Reads the contents of a single register.
0x33	Write_Single_Register	Writes the contents of a single register.
0x34	Read_Multiple_Registers	Reads the contents of a series of registers.
0x35	Write_Multiple_Registers	Writes the contents of a series of registers.

Read_Single_Register Instance Service Parameters:

Name	Type	Description
Register Number	DINT	The number of the register to be read.

Read_Single_Register Instance Service Response Data:

Name	Type	Description	
Status Code	DINT	0	Operation completed successfully.
		1	Operation could not be done because the register numbers are invalid.
		2	The service request format is invalid.
Register Contents	Variable	The value of the register.(1)(2)	

(1) The data is not returned unless the status code is 0.
(2) The values that are read/written to registers have the size specified by the Register Size attribute for that instance. For instance 0x1 (Integer Registers) and 0x2 (Double Registers) the values are 32 bits wide.

Write_Single_Register Instance Service Parameters:

Name	Type	Description
Register Number	DINT	The number of the register to be written.
New Register Value	*Variable*	The new register value to be written.

Write_Single_Register Instance Service Response Data:

Name	Type	Description	
Status Code	DINT	0	Operation completed successfully.
		1	Operation could not be done because the register numbers are invalid.
		2	The service request format is invalid.

Read_Multiple_Register Instance Service Parameters:

Name	Type	Description
Start Register Number	DINT	The number of the first register to be read. Must be less than the End Number Register.
End Register Number	DINT	The number of the last register to be read. Must be greater than the Start Register Number.

Read_Multiple_Register Instance Service Response Data:

Name	Type	Description	
Status Code	DINT	0	Operation completed successfully.
		1	Operation could not be done because the register numbers are invalid.
		2	The service request format is not valid.
Register Contents	Variable	The value of the register.(1)(2)	

(1) The data is not returned unless the status code is zero.
(2) When reading and writing a series of registers, the values are specified in order, end-to-end.

Write_Multiple_Register Instance Service Parameters:

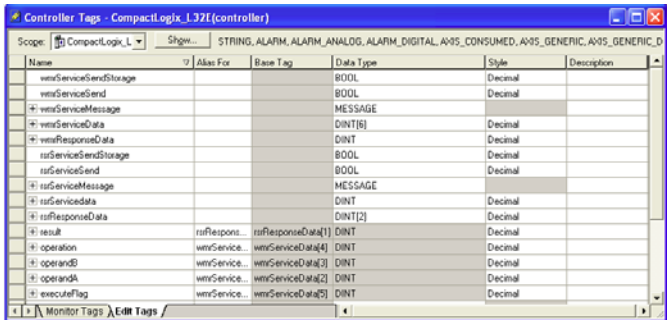
Name	Type	Description
Start Register Number	DINT	Number of the first register to be written. Must be less than End Register Number.
End Register Number	DINT	Number of the last register to be written. Must be greater than Start Register Number.
New Register Values	Variable	The new register values to be written.

Write_Multiple_Register Response Data:

Name	Type	Description	
Status Code	DINT	0	Operation completed successfully.
		1	Operation could not be done because the register numbers are invalid.
		2	The service request format is invalid.

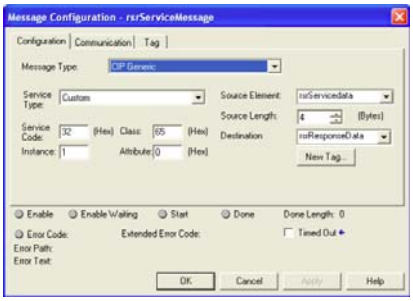
The register instances are used to pass information back and forth between the RSLogix program and the Ensemble or Soloist. The Ensemble or Soloist would have an AeroBasic program running on them that would execute some code based on the values of these registers.

Tags:



Sample Code:

Read Single Register message setup:



Write Multiple Register message setup: