



# EtherNet/IP<sup>®</sup>

**Ethernet-IP Communication  
with I-Mark**

---

**Technical Brochure**



## Contents

Contents .....	2
EtherNet/IP™ .....	3
Register Interface Object .....	4
Class Code: 0x65 (101) .....	4
Services .....	5
Behavior .....	10
EXAMPLES .....	10
Setup I-Mark for EtherNet/IP Communication .....	12
Step 1: Controller Setup.....	12
Step 2: Create a Layout with EtherNet/IP placeholder.....	15
Step 4: Set Layout as active layout for marking.....	17
Step 4: Save and download the configuration to the controller. ....	18
I-Mark Register Mapping .....	19
Output Control Register:.....	20
Writing Marking Data to the I-Mark Controller:.....	21
Establishing Communication with RSLogix 5000 .....	23
Step 1: Create a new RSLogix 5000 project .....	23
Step 2: Connect to the PLC using RSLogix 5000.....	23
Step 3: Assign IP addresses .....	24
Step 4: Verify EtherNet/IP communication .....	27
Register Interface Example with RSLogix 5000.....	31
Overview .....	31
Step 1: Programming the PLC side with RSLogix.....	32
Step 2: Demonstration.....	36

# Marking Machines



## **EtherNet/IP™**

CMT's EtherNet/IP™ option interfaces with I-Mark controller and the various ODVA Common Industrial Protocol (CIP™) objects that are used. It is intended only for users who have a working knowledge of the ODVA CIP and EtherNet/IP specifications. It is not intended to be used to learn about EtherNet/IP networks or protocol.

This option is available when purchased with an I-Mark Machine

The I-Mark Controller supports the following EtherNet/IP Interface Object:

[LINK TO SAMPLE CODE](#)



## Register Interface Object

### Class Code: 0x65 (101)

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

#### Attributes

##### ▣ Class Attributes

##### ▣ Instance Attributes

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

Number	Access Rule	Name	Data Type	Description
0x64 (100)	Get	Number of Registers	DINT	Returns the number of registers in this set.
0x65 (101)	Get	Register Size	DINT	Returns the size of each register, in bytes.
0x66 (102)	Get	Register Set Base Address	DINT	Returns the 32-bit base memory address of the register set.
0x67 (103)	Get	Register Set Mutex	DINT	Returns the mutex number associated with the register set.

# Marking Machines



## Services

### Common Services

Service Code	Name	Description
0x0E (14)	Get_Attribute_Single	Returns the contents of the specified attribute.

### Object-Specific Services

Service Code	Name	Description
0x32 (50)	Read_Single_Register	Read the contents of a single register.
0x33 (51)	Write_Single_Register	Write the contents of a single register.
0x34 (52)	Read_Multiple_Registers	Read the contents of a series of registers.
0x35 (53)	Write_Multiple_Registers	Write the contents of a series of registers.



Table: Service Parameters

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

Table: Service Response Data

Name	Type	Description
Status Code	DINT	0 The operation completed successfully
		1 The operation could not be performed because of invalid register numbers.
		2 Invalid service request format.
Register Contents	Variable	The value of the register. <sup>(1)</sup> <sup>(3)</sup>

# Marking Machines



Write\_Single\_Register Instance Service

Table: 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.

Table: Service Response Data

Name	Type	Description
Status Code	DINT	0 The operation completed successfully
Name	Type	Description
		1 The operation could not be performed because of invalid register numbers.
		2 Invalid service request format.



Table: Service Parameters

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

Table: Service Response Data

Name	Type	Description
Status Code	DINT	0 The operation completed successfully
		1 The operation could not be performed because of invalid register numbers.
		2 Invalid service request format.
Register Contents	Variable	The value of the register. <sup>(2)</sup> <sup>(3)</sup>

# Marking Machines



## Write\_Multiple\_Register Instance Service

Table: Service Parameters

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

Table: Service Response Data

Name	Type	Description
Status Code	DINT	0 The operation completed successfully
		1 The operation could not be performed because of invalid register numbers.
		2 Invalid service request format.

(1) The values that are read and written to registers have the size specified by the Register Size attribute for that instance. For instance 0x01 (Integer Registers) and instance 0x02 (Double Registers) the values are 32 bits wide.

(2) When reading and writing a series of registers, the values are specified in order, end-to-end.

(3) This data is not returned unless the status code is zero (the operation completed successfully).



## Behavior

The register interface instance services allow the reading and writing of the controller's Integer and Double Registers over EtherNet/IP. Data consistency is guaranteed internally, so you can access these registers simultaneously through EtherNet/IP with Multiple connections.

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 I-Mark controller utilizes only the 32-bit Integer Registers for control via EtherNet/IP

### EXAMPLES

1. EtherNet/IP request to write the value 1 (Start Marking) to Integer Register (298), which is the Input Control Register:
  1. Service: 0x33 (Write\_Single\_Register)
  2. Class: 0x65 (Register Interface Object)
  3. Instance: 0x01 (Instance 1 maps to Integer Registers)
  4. Attribute: 0x01 (Normal Writing Method)
  5. Request Data (Register Number, New Register Value): 0x00, 0x01, 0x00, 0x00, 0x0A, 0x00, 0x00, 0x00
  6. Response Data: (Status Code: Invalid Register): 0x01, 0x00, 0x00, 0x00

Note: Based on this Example, you can see we are using 2 words only for this packet instruction.

2. EtherNet/IP request to read the value of a Single Integer Register (299); which is the output Control Register:
  1. Service: 0x33 (Read\_Single\_Register)
  2. Class: 0x65 (Register Interface Object)
  3. Instance: 0x01 (Instance 1 maps to Integer Registers)

# Marking Machines



4. Attribute: 0x01 (Normal Writing Method)
5. Request Data (Integer Register Number): 0x12, 0x0B, 0x00, 0x00
6. Response Data (Status Code, Register Values): 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00

Note: Based on this Example the Controller is reporting back that it is currently Marking (Value of 2)

For sending data to the controller to be marked in a placeholder, the following would be the example

2. EtherNet/IP request to write the values 512 and 1024 to IntegerRegisters (300) and (301):
  1. Service: 0x35 (Write\_Multiple\_Registers)
  2. Class: 0x65 (Register Interface Object)
  3. Instance: 0x01 (Instance 1 maps to Integer Registers)
  4. Attribute: 0x01 (Normal Writing Method)
  5. Request Data (Start Register Number, End Register Number, Register Values): 0xFF, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00
  6. Response Data: (Status Code): 0x00, 0x00, 0x00, 0x00
  7. Note that you need a single 32-bit word for each of the registers you are sending data to within the I-Mark Controller. Each Integer register is capable of 4 characters of information. So if you wanted the marking to print "ABCD" you would only need to send to 1 register (300) with 1 word of data after converting the ABCD ASCII into integer on the PLC side first.



## Setup I-Mark for EtherNet/IP Communication

### Step 1: Controller Setup

The I-Mark controller will monitor the register **298** as our Input Control Register which allows you to remotely control the marking machine. Likewise for our Output status, the I-Mark controller (when configured for Ethernet/IP) will write an integer value to the register **299** as our current status. What we have in our software is a screen which allows you to map what functions you want to the different I/O bits within the program.

After you connect up to the controller (Home Tab  Connect to Marking Machines), the controller will show up under the Navigation pane on the lower left side of the software. You can double click on controller listed in here or if you select it, the Ribbon menu across the top will change to this controller specifically and you can select "Open Marking Machine". With this screen you can modify the parameters of the machine as well as get the full status and live feed of what the machine is doing.

1. With the ribbon still highlighted on your controller page, there is an icon which looks like a Green and Red arrow

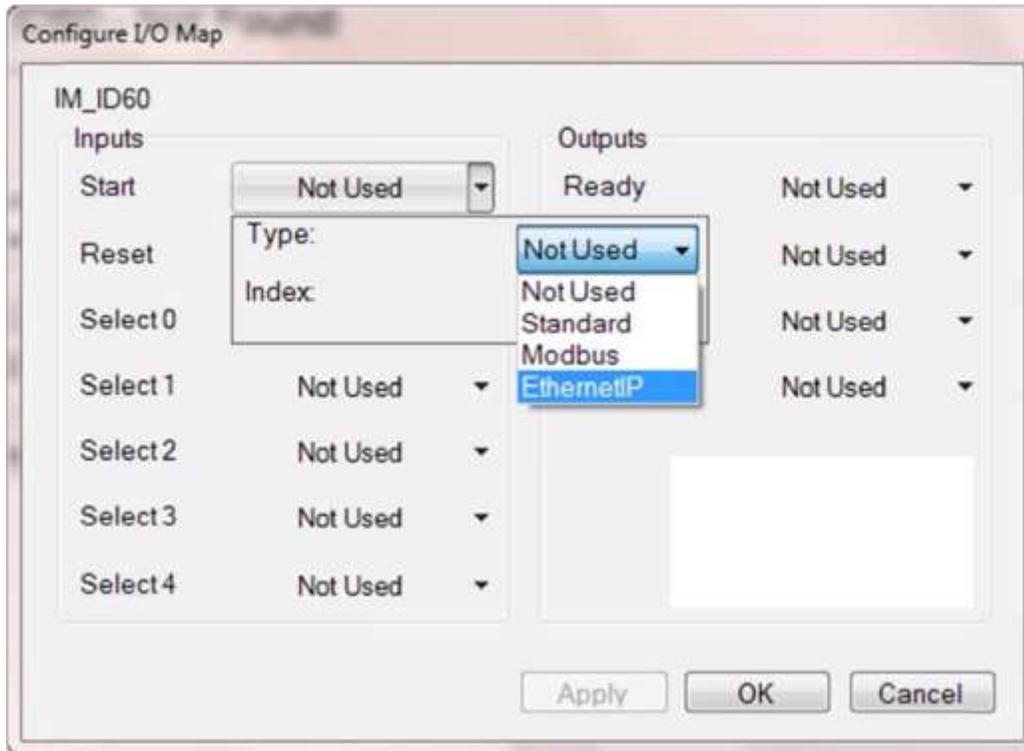
opposing each other....



# Marking Machines



This button will open up a dialog to allow you to configure the I/O's of this controller.



This dialog will allow you to map the function listed to the left with the desired I/O index number and medium you would like to control it with. So for your configuration, you are going to go through each of these items and change their "Type" to Ethernet/IP and then make sure the index number below it is unique.



A basic setup should look like this chart here.

Inputs			Outputs	
Start	EtherNetIP: 0	Binary 1	Ready	EtherNetIP: 0
Reset	EtherNetIP: 1	Binary 2	Marking	EtherNetIP: 1
Select 1	EtherNetIP: 2	Binary 4	Completed	EtherNetIP: 2
Select 2	EtherNetIP: 3	Binary 8	Fault	EtherNetIP: 3
Select 3	EtherNetIP: 4	Binary 16		
Select 4	EtherNetIP: 5	Binary 32		
Select 5	EtherNetIP: 6	Binary 64		

3. Once you have configured it to be this way, press OK to apply and save this configuration to the workspace.

# Marking Machines

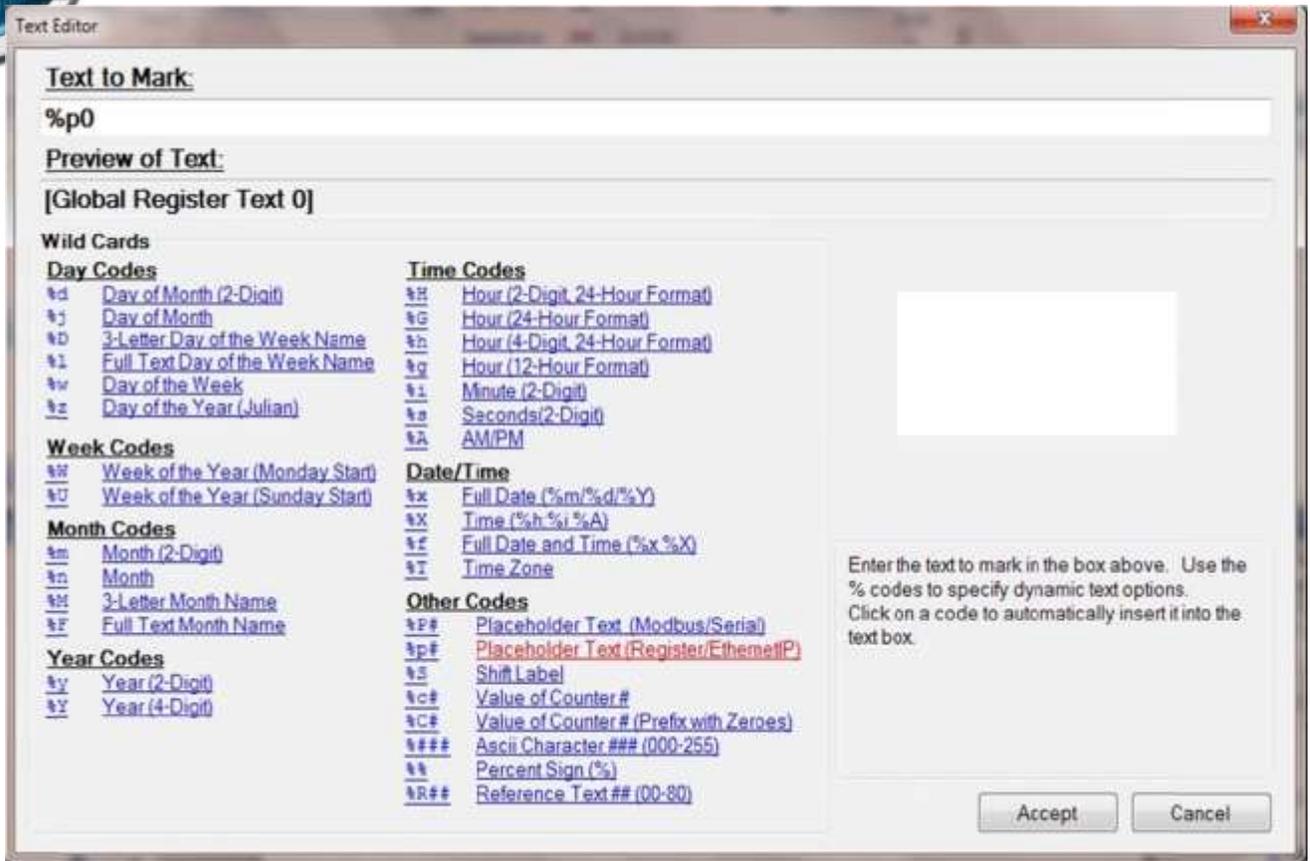


## Step 2: Create a Layout with EtherNet/IP placeholder

1. Go back to the Home tab and click “Create New Layout”. It will ask you what type of machine you want to create a layout for, if you’re directly connected to the machine there will only be 1 listed there (IM\_ID60) just select this and click OK.
2. Now you can see the Layout which appears to look like a grid, this is the physical marking window for the machine. To add a new text object which will print data received from the PLC over Ethernet/IP, click on the “Layout” tab at the top and single click on the button which says “Text”.



This button will drop a text entity into the center of the layout which reads simply “Text”. You modify this entity by double clicking on it which opens up a new dialog for editing the text entity. Delete the sample “Text” in the top field and then look down at the different codes available for dynamic data. One of the codes is labeled “Placeholder Text (Register/Ethernet/IP)”. You can click directly on this text or type in %p0 to the Text to Mark field as shown here.



Now press Accept to apply the change to the Text entity. The text within the layout will say "Global Register Text 0" as this is just a simulation to illustrate that we are waiting for data to appear in order to mark it here because the register at this time is empty.

Step 3: Assign Layout to the controller.

1. Click directly on the layout to activate the layout tab on the ribbon,
2. Click on the button labeled "Assign to Machine"



# Marking Machines



Click on the machine listed below this button which corresponds to the name of the machine you're currently working with. This will assign the currently layout to this machine so when a download occurs it will synchronize.

3. Save the Layout by going to the "Home" tab and select "Save Layout"

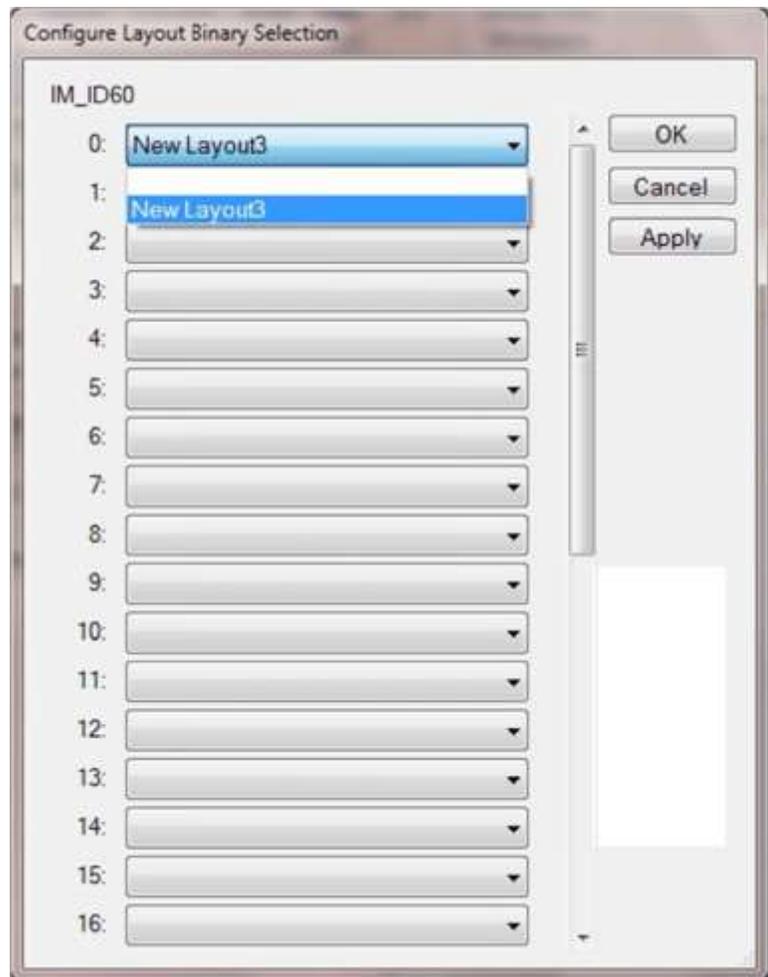


## Step 4: Set Layout as active layout for marking.

1. Click the tab on the toolbar named after the controller you are working on. On this ribbon will be a button called "Layouts" click it



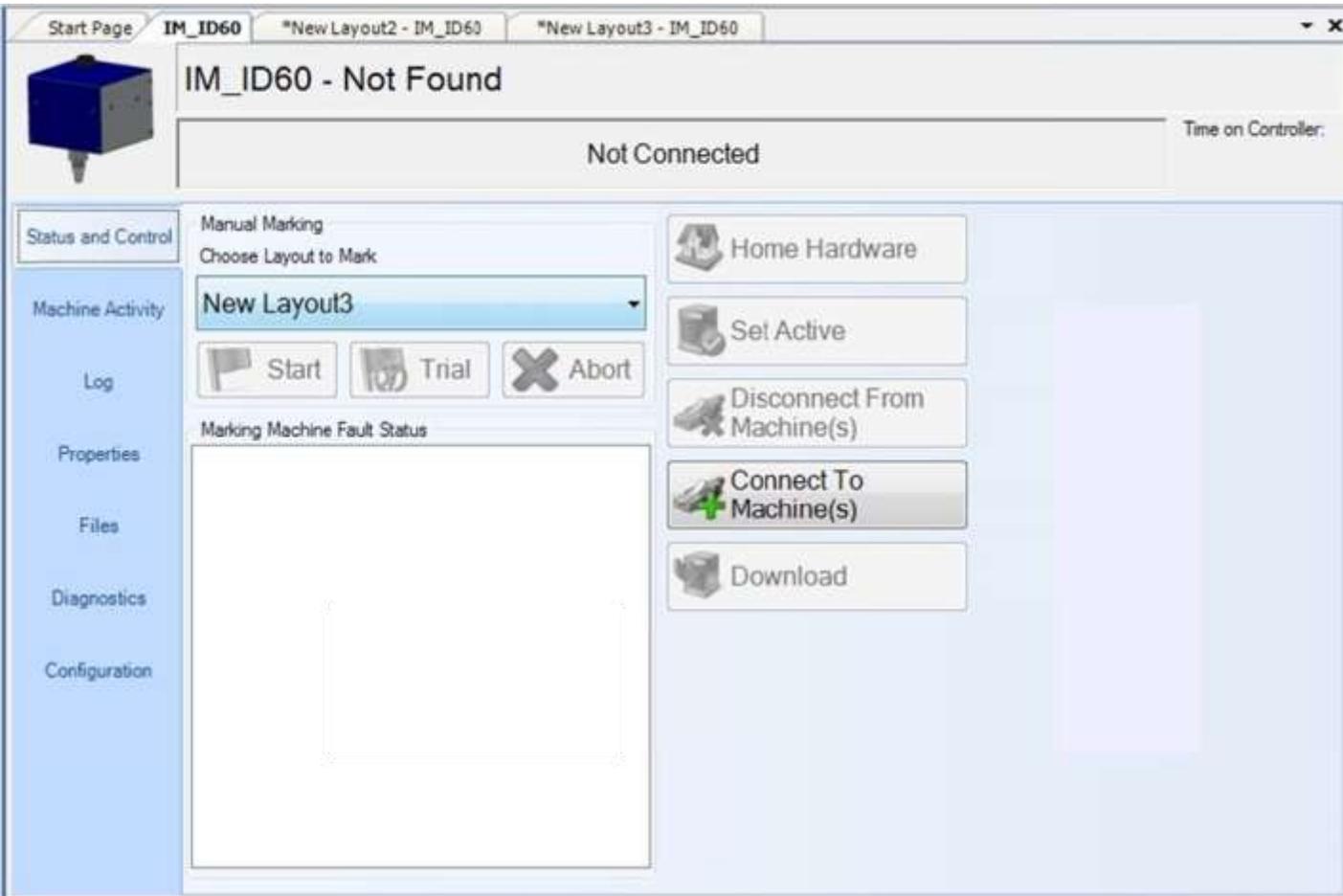
Clicking this button will open up a dialog for you to assign layouts to a specific binary assignment. Set the layout you just created as the





## Step 4: Save and download the configuration to the controller.

1. Navigate to the Setup and Control tab within the Controller Page.



2. Click on the Download button to save all of your Layouts and configuration to the controllers memory.
3. After the progress completes, you may now disconnect from I-Mark.



## I-Mark Register Mapping

Input Control Register:

The I-Mark controller uses the Integer Register number **298** as the dedicated Input Control Register for the marking machine. This register is checked once every 1ms and executes the command mapped to the binary value(s) found in it. In the section "[I-Mark Register Mapping](#)", we explained how to setup your input and output functions for EtherNet/IP and their Binary Index.

Taking that example below is a chart illustrating a standard mapping of I/O for EtherNet/IP.

Inputs			Outputs	
Start	EtherNetIP: 0	Binary 1	Ready	EtherNetIP: 0
Reset	EtherNetIP: 1	Binary 2	Marking	EtherNetIP: 1
Select 1	EtherNetIP: 2	Binary 4	Completed	EtherNetIP: 2
Select 2	EtherNetIP: 3	Binary 8	Fault	EtherNetIP: 3
Select 3	EtherNetIP: 4	Binary 16		
Select 4	EtherNetIP: 5	Binary 32		
Select 5	EtherNetIP: 6	Binary 64		

To execute the function assigned to the binary index, perform a bitwise calculation and enter the Sum of the binary values into the Input Control Register. The next time this



# I-Mark

register is polled by the controller for its value, the controller will execute the command(s) requested.

Example: To tell the controller to "Start" marking, write the value of 1 to the Input Control Register. Then when the register is polled for its value, the controller will initiate the marking sequence for the layout assigned to "0" in the previous section.

Example 2: If you have assigned multiple layouts to Select Number using the Layouts dialog, put the value of 1 for "Start" command plus the value of 4 for the Select 1 command. So the Input Control Register will have the integer number 5 commanding the marker to Start marking using Select 1.

When a new packet of data is sent to this register it will clear the previous out first before writing the new value into it. It is a good practice to write the value of 0 into the register after the marker has started/finished the command. If the register contains the "Start" command, the marker will not return to home until the register is brought back to 0 as it is trying to execute a "Start" each time the register is scanned. After you have sent the packet of data to the Marker, it will provide a response back (See Register Interface Object section), using this Response packet in the PLC as a trigger to send a new MSG instruction containing the value of 0 to the I-Mark controllers Input Control Register is recommended.

## **Output Control Register:**

The Output Control Register will function in a similar manner to the Input Control Register. The marking controller will provide a status in the form of a Bitwise calculation which can be read by the PLC for current marker status. Like the Input Control Register, the output will likewise be updated at a frequency of 1ms depending on the current process being executed on the I-Mark Controller.

# Marking Machines



Example: Using the chart above, when the marking controller is in a Ready state, the register 299 will contain the value of

1. Similarly if there is a fault within the marking controller, the value of 8 will be present in the Output Control register.

## Writing Marking Data to the I-Mark Controller:

The I-Mark controller will accept ASCII integer values in specified placeholder registers to be used as marking data for your layout. These registers are global which allows them to be used in multiple layouts without outside intervention. The I-Mark memory utilized 32-bit integer registers to hold this marking data similar to the Input and Output Control

Registers. Within the I-Mark software itself you can create an entity known as a "Placeholder" which will utilize this

local data to be marked within the layout. The place holder is capable of a maximum 80 characters of information with each character using 8 bits or 1 byte of data.

Remembering that our registers can hold 32-bits of information, each placeholder would occupy a total of 20 Integer Registers for holding that data. The I-Mark software allocates these Integer Registers for the assigned placeholders in the follow locations.

Placeholder		
%p0	Register. 300-319	Placeholder 0
%p1	Register. 320-339	Placeholder 1
%p2	Register. 340-359	Placeholder 2
%p3	Register. 360-379	Placeholder 3
%p4	Register. 380-399	Placeholder 4
%p5	Register. 400-419	Placeholder 5
%p6	Register. 420-439	Placeholder 6
%p7	Register. 440-459	Placeholder 7
%p8	Register. 460-479	Placeholder 8
%p9	Register. 480-499	Placeholder 9



# I-Mark

Once the layout which utilizes any of these registers is executed, the I-Mark controller will look at the value of the corresponding register and print what was found within it. If a register is programmed within a layout but does not contain any data, there will be a corresponding error message within the controller producing a Fault (Output Control Register Bit 8).



## Establishing Communication with RSLogix 5000

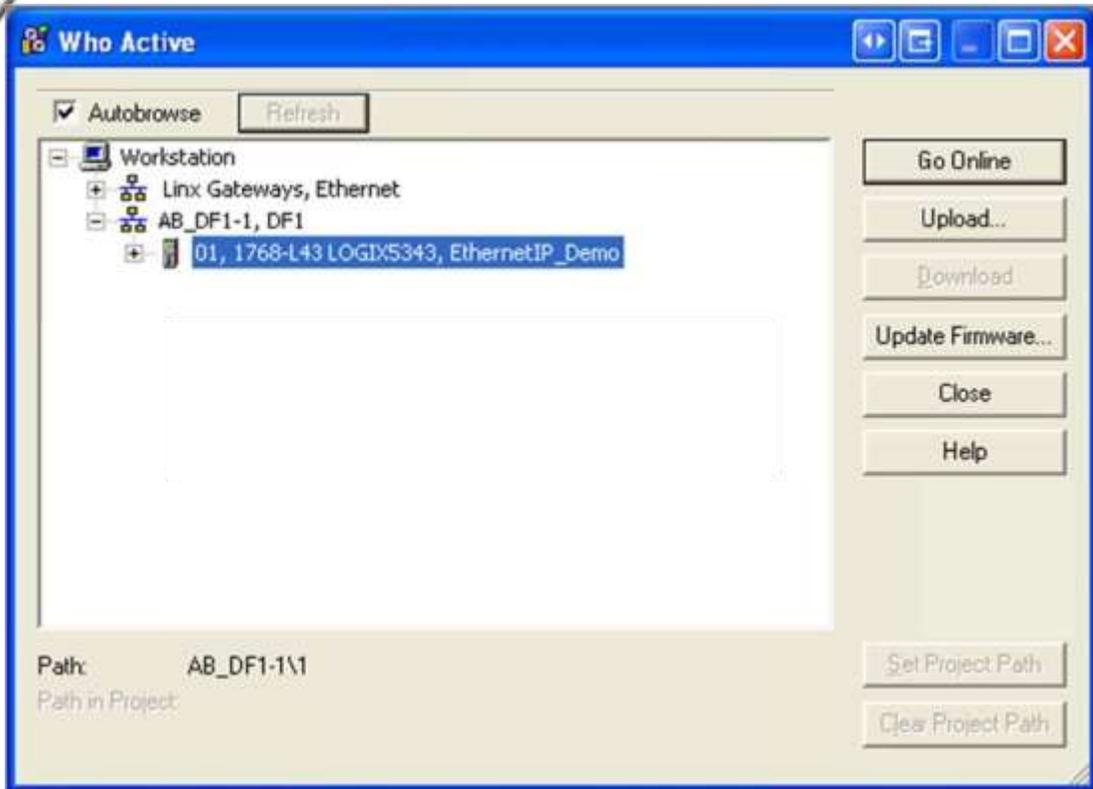
This procedure provides an example of how to establish EtherNet/IP communication between a PLC (programmed via RSLogix 5000) and an I-Mark controller. Note that your setup procedure may differ depending on the particular PLC and version of RSLogix that you are using. As always, the user must be certain that conditions are safe before downloading changes to the PLC, and that any existing logic on the PLC is backed up before it is overwritten.

### Step 1: Create a new RSLogix 5000 project

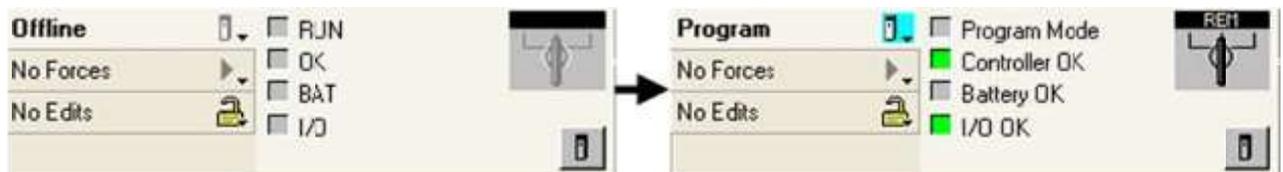
1. Open RSLogix 5000.
2. Create a new project by going to File → New.
  - a. Select the appropriate controller type and revision.
  - b. Specify a name for the controller and click OK.

### Step 2: Connect to the PLC using RSLogix 5000

1. After verifying that it is safe to do so, switch the PLC into programming mode.
2. In RSLogix 5000, select the communications path to the PLC.
  - a. Go to Communications → Who Active.
  - b. Highlight the PLC in the communications tree and click Go Online.



- c. You may be prompted that the open project (the one we just created) does not match the project in the controller. We are going to download the new project to the controller. This will erase the existing logic on the PLC. If this is OK, click Download.
- d. The online toolbar in RSLogix 5000 will indicate that the PLC is now online.



- e. Use the online toolbar to go offline.

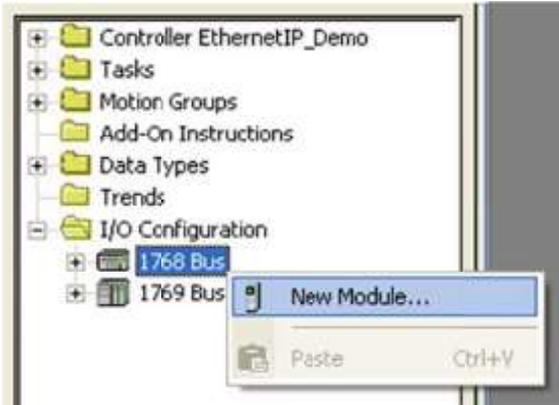
### Step 3: Assign IP addresses

Assign an IP address to the EtherNet/IP module on the PLC. The following example is for a CompactLogix L43 controller with a 1768-ENBT/A EtherNet/IP module. A static IP address is used for this example.

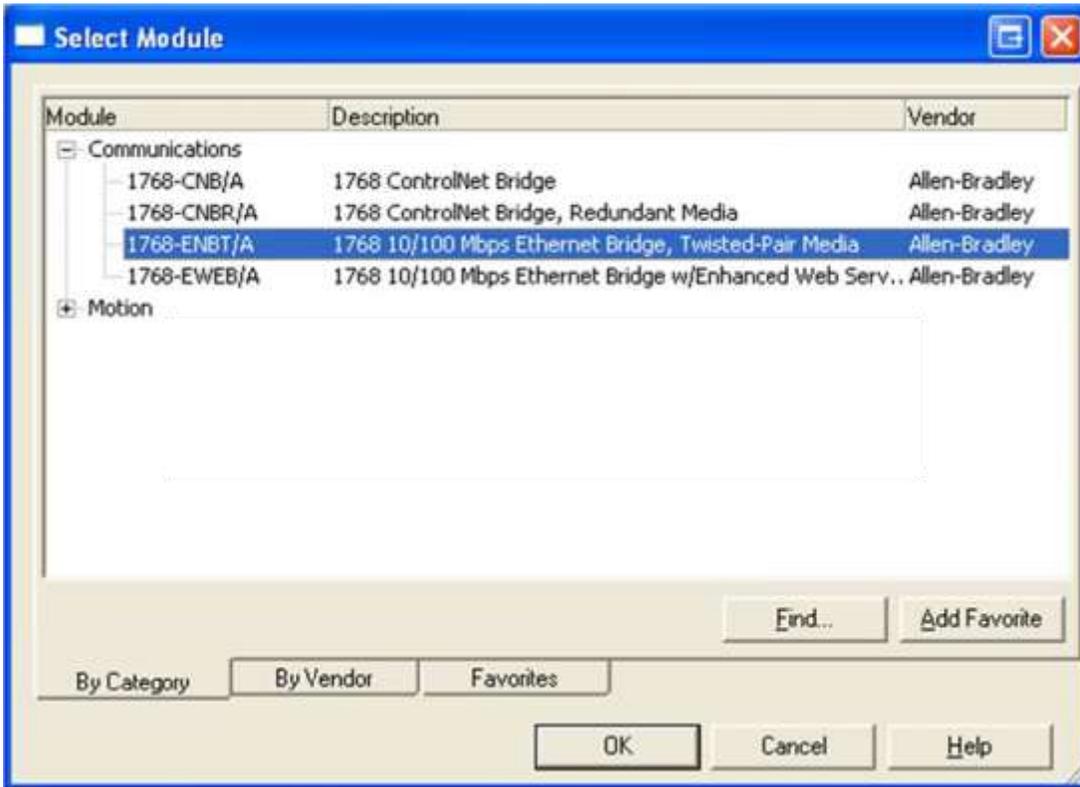
# Marking Machines



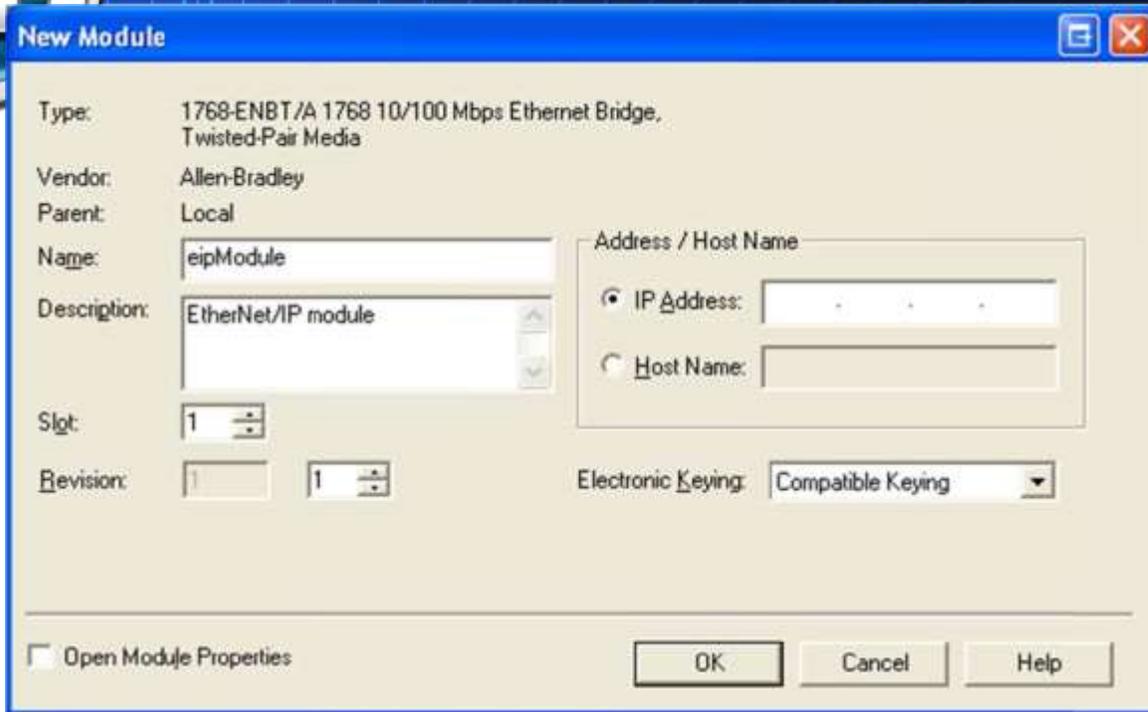
In the controller organizer, right click on the bus that carries the EtherNet/IP module and select New Module.



Choose the appropriate module and click OK.



Enter a name and description for the module, uncheck Open Module Properties and click OK.



The image shows a 'New Module' dialog box with the following fields and options:

- Type: 1768-ENBT/A 1768 10/100 Mbps Ethernet Bridge, Twisted-Pair Media
- Vendor: Allen-Bradley
- Parent: Local
- Name: eipModule
- Description: EtherNet/IP module
- Slot: 1
- Revision: 1
- Electronic Keying: Compatible Keying
- Address / Host Name section with radio buttons for:
  - IP Address: [ ]
  - Host Name: [ ]
- Open Module Properties
- Buttons: OK, Cancel, Help

Go online, downloading offline changes as necessary.

In the controller organizer, right click on the EtherNet/IP module and select Properties.

Click on the Port Configuration tab in the module properties. Configure the IP address and other network settings, and then click Set. Click OK to exit the module properties window.

# Marking Machines



The screenshot shows the 'Module Properties: Local:1 (1768-ENBT/A 1.1)' dialog box with the 'Port Configuration' tab selected. The 'IP Address' is set to 192.168.1.1, and the 'Subnet Mask' is 255.255.255.0. The 'Gateway Address' is 0.0.0.0, and both 'Primary DNS Server Address' and 'Secondary DNS Server Address' are 0.0.0.0. The 'Domain Name' and 'Host Name' fields are empty. The 'Select Port Speed' dropdown is set to 10 Mbps, and the 'Current Port Speed' is 10 Mbps. The 'Select Duplex' dropdown is set to Indeterminate, and the 'Current Duplex' is Indeterminate. The 'Enable DNS' and 'Auto-Negotiate Port Speed and Duplex' checkboxes are checked. The 'Refresh' and 'Set' buttons are visible. The status at the bottom is 'Running'.

1. Assign an IP address to the I-Mark controller. A static IP address is used for this example.
  - a. Controllers Properties Page.
  - b. Locate the Data section within the properties.
  - c. Double click to specify the network settings and press ENTER.

## Step 4: Verify EtherNet/IP communication

This step will guide you through the creation of a basic PLC program to test the EtherNet/IP communication with the IMark controller.

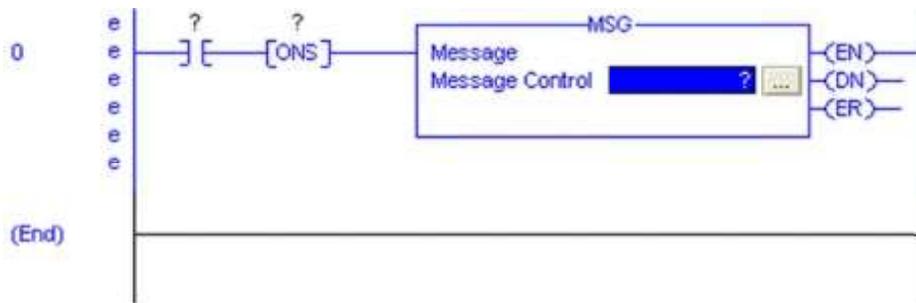
1. Go offline in RSLogix 5000.
2. In the controller organizer, right click on Controller <name> → Controller Tags and select Edit Tags.
  - a. Create the tags shown:



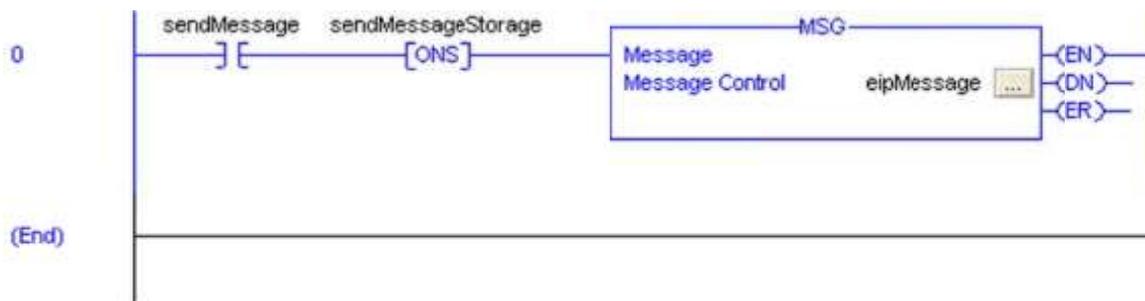
# I-Mark

Name	△	Data Type	Style
+ eipMessage		MESSAGE	
+ vendorID		INT	Decimal
sendMessage		BOOL	Decimal
sendMessageStorage		BOOL	Decimal

3. In the controller organizer, right click on Tasks → MainTask → MainProgram → MainRoutine and select Open. An empty ladder diagram will be displayed.
4. Add the following elements to the rung: Examine If Closed (XIC), One Shot (ONS) and Message (MSG).



5. Set the operands as follows:
  - a. Examine If Closed (XIC): sendMessage
  - b. One Shot (ONS): sendMessageStorage
  - c. Message (MSG): eipMessage

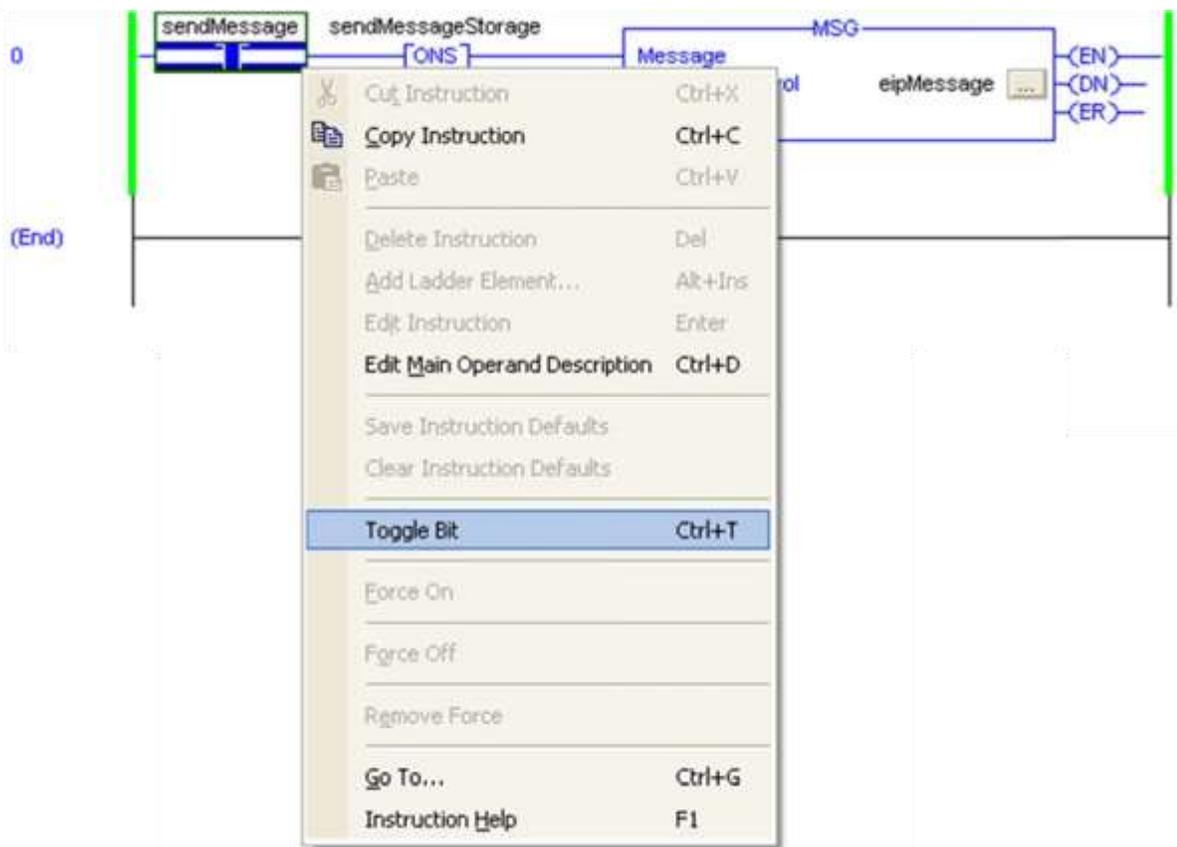




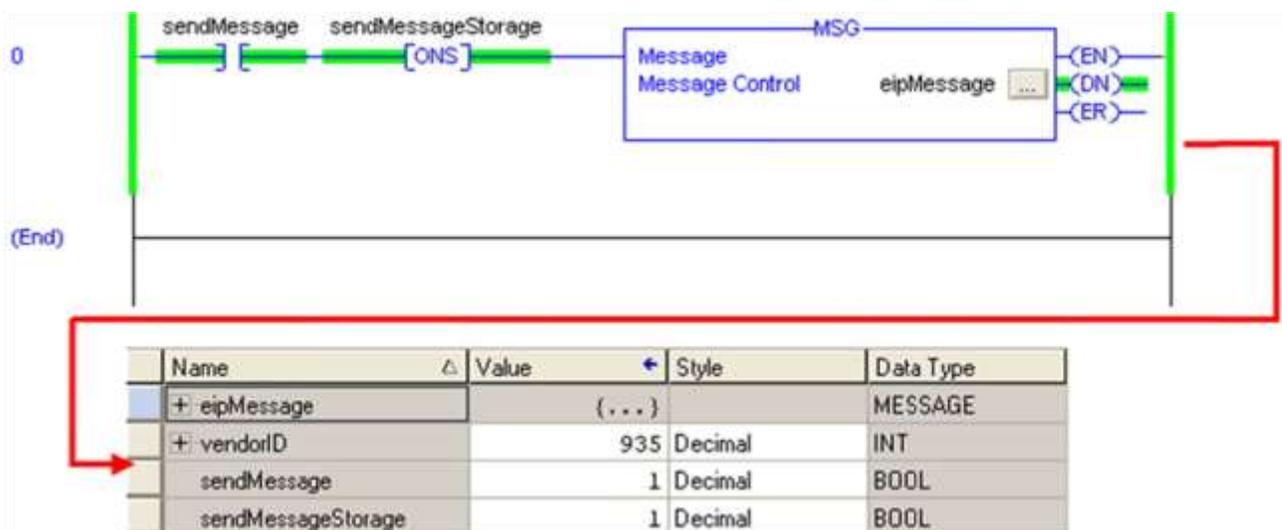
6. Click the ellipses button on the MSG instruction to configure the EtherNet/IP message. Configure the message as follows:
  - a. Configuration Tab
    - 1 Service Code: E
    - 2 Class: 1
    - 3 Instance:1
    - 4 Attribute: 1
    - 5 Source Length: 0
    - 6 Destination: vendorID
  - b. Communication Tab
    - 1 Path: eipModule, 2,  
192.168.1.2
  - c. Click OK.
7. Go online, downloading offline changes as necessary.
8. Once the project has been downloaded, switch the PLC into run or remote run mode.
9. Right click on the XIC element in the ladder diagram and select Toggle Bit. This will activate the rung and trigger the MSG block to send the EtherNet/IP message.



# I-Mark



10. Once the DN output from the MSG block is active, return to the controller organizer, right click on Controller <name> → Controller Tags and select Monitor Tags. The vendorID tag should now have the value 935 (the EtherNet/IP vendor ID for CMT, Inc). You have now verified the EtherNet/IP communication between the PLC and controller.





## Register Interface Example with RSLogix 5000

This example demonstrates how the EtherNet/IP™ Register Interface object can be used to exchange arbitrary data between the PLC and I-Mark controller. Please read the [Establishing Communication with RSLogix 5000](#) document and verify the EtherNet/IP™ communication with the controller before proceeding. It would also be very beneficial to read the [Register Interface Object](#) topic in the controller help file before continuing. You may create a new RSLogix 5000 project for this example application or continue to build on an existing project. As always, the user must be certain that conditions are safe before downloading changes to the PLC, and that any existing logic on the PLC is backed up before it is overwritten.

### Overview

This example application demonstrates how the Register Interface object can be used to exchange data and "trigger" the execution of some process on the controller. It involves some basic logic on the PLC (programmed via RSLogix 5000). Although this example is fairly simplistic, the programming concepts employed are extensible to much more complex applications.

For this example, the I-Mark controller will execute a programmed Layout when commanded by the PLC. The operands are specified on the PLC and passed to the controller. The controller stores the result in its register space, and the PLC reads the value back through the Register Interface.

Two EtherNet/IP messages are required for this application: one message to send the operands and trigger the behavior on the controller, and a second message to read the result back into the PLC. Since simple integers are needed to be passed to the controller, we want to use the `Write_Single_Registers` service of the Register Interface object for the first message. A single result value is read back into the PLC, so a `Read_Single_Register` service may be used for the second message.



1. Configure the necessary controller tags for the Write\_Single\_Register message.
  - a. Create controller tags named wsrServiceSend and wsrServiceSendStorage of type BOOL; these will be used in the ladder diagram to control when the Write\_Single\_Register message is sent.
  - b. We need tags for the operands, a code to specify the requested operation, and a flag to instruct the controller to perform the marking.
    - i. Create a controller tag named operandA of type DINT.
    - ii. Create a controller tag named operandB of type DINT.
    - iii. Create a controller tag named operation of type DINT.
    - iv. Create a controller tag named executeFlag of type DINT.
  - c. We need a tag to hold the service data for the Write\_Single\_Register service. The service data for the Write\_Single\_Register service are 1) the starting register number (32 bits), 2) the new register values to write (32 bits). For our message, this is a total of two 32-bit words. Create a controller tag named wsrServiceData of type DINT[4].
  - d. For convenience, we can alias the other controller tags so they map directly into the service data tag.
    - i. Set controller tag operandA as an alias for wsrServiceData[2].
    - ii. Set controller tag operandB as an alias for wsrServiceData[3].
    - iii. Set controller tag operation as an alias for wsrServiceData[4].



- iv. Set controller tag executeFlag as an alias for wsrServiceData[5].
  - e. The Write\_Single\_Register service returns a 32-bit status code. Create a controller tag named wmrReponseData of type DINT to hold this value.
  - f. Finally, a MESSAGE tag must be created; create a controller tag named wsrServiceMessage of type MESSAGE.
2. Configure the necessary controller tags for the Read\_Single\_Register message.
    - a. Create controller tags named rsrServiceSend and rsrServiceSendStorage of type BOOL; these will be used in the ladder diagram to control when the Read\_Single\_Register message is sent.
    - b. We need a tag for the result value that will be read back from the controller. Create a controller tag named result of type DINT.
    - c. We need a tag to hold the service data for the Read\_Single\_Register service. The service data for the Read\_Single\_Register service consists only of the register number to be read (32 bits). Create a controller tag named rsrServiceData of type DINT.
    - d. The Read\_Single\_Register service returns a 32-bit status code followed by the contents of the register in another 32-bit register. Create a controller tag named rsrReponseData of type DINT[2] to hold these values.
    - e. For convenience, we can alias the result controller tag so it maps directly into the service response data tag. Set controller tag result as an alias for rsrResponseData[1].
    - f. Finally, a MESSAGE tag must be created; create a controller tag named rsrServiceMessage of type MESSAGE.

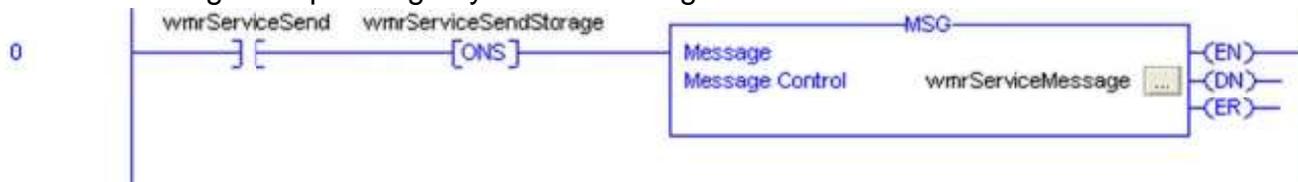
In summary:



Name	Alias For	Data Type	Style
wmrServiceSend		BOOL	Decimal
wmrServiceSendStorage		BOOL	Decimal
+ operandA	wmrServiceData[2]	DINT	Decimal
+ operandB	wmrServiceData[3]	DINT	Decimal
+ operation	wmrServiceData[4]	DINT	Decimal
+ executeFlag	wmrServiceData[5]	DINT	Decimal
+ wmrServiceData		DINT[6]	Decimal
+ wmrResponseData		DINT	Decimal
rsrServiceSend		BOOL	Decimal
rsrServiceSendStorage		BOOL	Decimal
+ result	rsrResponseData[1]	DINT	Decimal
+ rsrServiceData		DINT	Decimal
+ rsrResponseData		DINT[2]	Decimal
+ wmrServiceMessage		MESSAGE	
+ rsrServiceMessage		MESSAGE	

3. Create the logic to send the Write\_Single\_Register service to the controller:

a. Add the following example rung to your ladder diagram:



2. When wsrServiceSend is activated, the specified message is sent. We want this to be our Write\_Single\_Register message, so configure the MSG block as follows:

1. Configuration Tab

1. Service Code: 33 (Write\_Single\_Register)
2. Class: 65 (Register Interface Object)
3. Instance: 1 (IntegerRegisters)
4. Attribute: 1 (Normal Mode)
5. Source Element: wsrServiceData
6. Source Length: 16

# Marking Machines



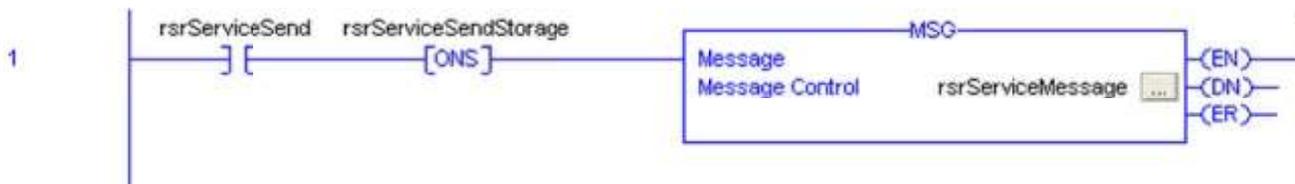
7. Destination: wsrResponseData

## 2. Communication Tab

1. Path: eipModule, 2, 192.168.1.2 (or the appropriate path to your EtherNet/IP module)

4. Create the logic to send the Read\_Single\_Register service to the controller:

1. Add the following rung to your ladder diagram:



2. When rsrServiceSend is activated, the specified message is sent. We want this to be our Read\_Single\_Register message, so configure the MSG block as follows:

## 1. Configuration Tab

1. Service Code: 32 (Read\_Single\_Register)
2. Class: 65 (Register Interface Object)
3. Instance: 1 (IntegerRegisters)
4. Attribute: 1 (Normal Write Mode)
5. Source Element: rsrServiceData
6. Source Length: 4
7. Destination: rsrResponseData

## 2. Communication Tab

1. Path: eipModule, 2, 192.168.1.2 (or the appropriate path to your EtherNet/IP module)



## Step 2: Demonstration

- a. Set a Marking Layout as assigned to 0 on the I-Mark controller.
- b. Download the RSLogix 5000 project to the PLC. Switch the PLC into a run mode.
- c. Set the register number values in the service data tags for the Write\_Single\_Register and Read\_Single\_Register messages.
  - a. Set wsrServiceData[0] (starting register number for Write\_Single\_Register) to 0.
  - b. Set rsrServiceData (register number for Read\_Single\_Register) to 4.
- d. Set values for the operandA and operandB tags. Set the value of the operation tag to 0 for addition or any other value for multiplication. Set the value of the executeFlag tag to 1.
- e. Change the value of the wsrServiceSend tag from 0 to 1. This will energize its rung in the ladder diagram and cause the Write\_Single\_Register message to be sent to the controller.
- f. Check the value of the wsrResponseData tag. It should be zero, indicating that the operation completed successfully.
- g. Change the value of the rsrServiceSend tag from 0 to 1. This will energize its rung in the ladder diagram and cause the Read\_Single\_Register message to be sent to the controller.
- h. Check the value of the rsrResponseData tag. rsrResponseData[0] should be zero, indicating that the operation completed successfully. result will contain the result value obtained from the controller.
- i. To send additional messages, change the value of the corresponding "ServiceSend" tag to zero, wait for the "ServiceSendStorage" tag to become zero, and then return the ServiceSend tag to one. A message is only sent on a rising edge of the corresponding "ServiceSend" signal.