

COGNEX®

DataMan®

Communications and Programming Guide

12/2/2011
Version 4.2

Distribué par :



Contact :
hvssystem@hvssystem.com

Tél : 0326824929
Fax : 0326851908

Siège social :
2 rue René Laennec
51500 Taissy
France

www.hvssystem.com

The software described in this document is furnished under license, and may be used or copied only in accordance with the terms of such license and with the inclusion of the copyright notice shown on this page. Neither the software, this document, nor any copies thereof may be provided to, or otherwise made available to, anyone other than the licensee. Title to, and ownership of, this software remains with Cognex Corporation or its licensor. Cognex Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Cognex Corporation. Cognex Corporation makes no warranties, either express or implied, regarding the described software, its merchantability, non-infringement or its fitness for any particular purpose.

The information in this document is subject to change without notice and should not be construed as a commitment by Cognex Corporation. Cognex Corporation is not responsible for any errors that may be present in either this document or the associated software.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, nor transferred to any other media or language without the written permission of Cognex Corporation.

Copyright © 2011 Cognex Corporation. All Rights Reserved.

Portions of the hardware and software provided by Cognex may be covered by one or more of the U.S. and foreign patents listed below as well as pending U.S. and foreign patents. Such pending U.S. and foreign patents issued after the date of this document are listed on Cognex web site at <http://www.cognex.com/patents>.

VisionPro

5481712, 5495537, 5548326, 5583954, 5602937, 5640200, 5751853, 5768443, 5825913, 5850466, 5872870, 5901241, 5943441, 5978080, 5978521, 5987172, 6005978, 6039254, 6064388, 6075881, 6137893, 6141033, 6167150, 6215915, 6240208, 6324299, 6381366, 6381375, 6411734, 6421458, 6459820, 6490375, 6516092, 6563324, 6658145, 6687402, 6690842, 6695753, 6718074, 6748110, 6771808, 6804416, 6836567, 6850646, 6856698, 6920241, 6959112, 6963338, 6973207, 6975764, 6985625, 6993177, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7190834, 7242801, 7251366, 7313761, EP0713593, JP3522280, JP3927239

DataMan

5742037, 5943441, 6215915, 6236769, 6282328, 6381375, 6408109, 6457032, 6690842, 6941026, 7175090, 7181066, 7412106, 7427028, 7549582, 7604174, 7614563, 7617984, US-2005-0087601-A1, US-2006-0131418-A1, US-2006-0131419-A1, US-2006-0133757-A1, US-2007-0090193-A1, US-2007-0091332-A1, US-2007-0152064-A1, US-2007-0170259-A1, US-2008-0004822-A1, US-2008-0011855-A1, US-2008-0142604-A1, US-2008-0143838-A1, US-2008-0158365-A1, US-2009-0090781-A1, US-2009-0108073, US-2009-0121027-A1, US-2009-0166424-A1, US-2009-0294541-A1, WO06065619A1, EP1687752

CVL

5495537, 5548326, 5583954, 5602937, 5640200, 5717785, 5751853, 5768443, 5825483, 5825913, 5850466, 5859923, 5872870, 5901241, 5943441, 5949905, 5978080, 5987172, 5995648, 6002793, 6005978, 6064388, 6067379, 6075881, 6137893, 6141033, 6157732, 6167150, 6215915, 6240208, 6240218, 6324299, 6381366, 6381375, 6408109, 6411734, 6421458, 6457032, 6459820, 6490375, 6516092, 6563324, 6658145, 6687402, 6690842, 6718074, 6748110, 6751361, 6771808, 6798925, 6804416, 6836567, 6850646, 6856698, 6920241, 6959112, 6975764, 6985625, 6993177, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7190834, 7242801, 7251366, EP0713593, JP3522280, JP3927239

VGR

5495537, 5602937, 5640200, 5768443, 5825483, 5850466, 5859923, 5949905, 5978080, 5995648, 6002793, 6005978, 6075881, 6137893, 6141033, 6157732, 6167150, 6215915, 6240208, 6748110, 6751361, 6771808, 6804416, 6836567, 6850646, 6856698, 6959112, 6975764, 6985625, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7190834, 7242801, 7251366

OMNIVIEW

6215915, 6381375, 6408109, 6421458, 6457032, 6459820, 6594623, 6804416, 6959112, 7383536

CVL Vision Library

5495537, 5548326, 5583954, 5602937, 5640200, 5717785, 5751853, 5768443, 5825483, 5825913, 5850466, 5859923, 5872870, 5901241, 5943441, 5949905, 5978080, 5987172, 5995648, 6002793, 6005978, 6064388, 6067379, 6075881, 6137893, 6141033, 6157732, 6167150, 6215915, 6240208, 6240218, 6324299, 6381366, 6381375, 6408109, 6411734, 6421458, 6457032, 6459820, 6490375, 6516092, 6563324, 6658145, 6687402, 6690842, 6718074, 6748110, 6751361, 6771808, 6798925, 6804416, 6836567, 6850646, 6856698, 6920241, 6959112, 6975764, 6985625, 6993177, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7190834, 7242801, 7251366, EP0713593, JP3522280, JP3927239

SMD 4

5995648, 5850466, 6751361, 6690842, 6563324, 6490375, 5949905, 5978080, 6137893, 6167150, 6075881, 6748110, 5859923, 6411734, 6324299, 6516092, 7190834, 6658145, 6836567, 6850646, 6975764, 6985625, 6993192, 7006712, 7043081, 7058225, 7065262, 7088862, 7164796, 7251366, 6856698, 6002793, 6005978, 6771808, 6804416, 7016539, 6959112, 5602937, 7242801, 5640200, 5495537, 5768443, 5825483, 6421458, 6459820,

6215915, 6381375, 6457032, 6157732, 6408109, 6141033, 6026176, 6442291, 6151406, 6396942, 6614926, 5371690, 5845007, 5943441, 6963338, 5805722, 5909504, 5933523, 5964844, 5974169, 5987172, 6078700, 6252986, 6278796, 6307210, 6408429, 6424734, 6526165, 6571006, 6639624, 6681039, 6748104, 6813377, 6853751, 6898333, 6950548, 6993177, 7139421, 5757956

BGA II and BGA III

5495537, 5602937, 5640200, 5768443, 5801966, 5825483, 5850466, 5859923, 5949905, 5978080, 5995648, 6002793, 6005978, 6026176, 6055328, 6075881, 6115042, 6118893, 6130959, 6137893, 6141009, 6141033, 6151406, 6157732, 6167150, 6215915, 6289117, 6324299, 6353676, 6381375, 6396942, 6408109, 6411734, 6421458, 6442291, 6457032, 6459820, 6490375, 6516092, 6563324, 6577775, 6614926, 6658145, 6690842, 6748110, 6751361, 6771808, 6804416, 6836567, 6850646, 6856698, 6959112, 6975764, 6985625, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7190834, 7242801, 7251366

Wire Bonder

5495537, 5532739, 5581632, 5602937, 5640199, 5640200, 5642158, 5676302, 5754679, 5757956, 5768443, 5825483, 5835622, 5850466, 5859923, 5861909, 5949905, 5978080, 5991436, 5995648, 6002793, 6005978, 6035066, 6061467, 6075881, 6137893, 6141033, 6157732, 6167150, 6215915, 6289492, 6324299, 6381375, 6408109, 6411734, 6421458, 6457032, 6459820, 6490375, 6516092, 6563324, 6658145, 6690842, 6748110, 6751361, 6771808, 6804416, 6836567, 6850646, 6856698, 6959112, 6975764, 6985625, 6993192, 7006712, 7016539, 7043081, 7058225, 7065262, 7088862, 7164796, 7171036, 7190834, 7242801, 7251366

The following are registered trademarks of Cognex Corporation:

acuReader® BGAI® Check it with Checker® Checker® Cognex Vision for Industry CVC-1000® CVL® DataMan® DisplayInspect® DVT® EasyBuilder® IDMax® In-SightIn-Sight 2000® In-Sight® (insignia with cross-hairs) MVS-8000® OmniView® PatFind® PatFlex® PatInspect® PatMax® PatQuick® SensorView® SmartLearn® SmartView® SMD4® UltraLight® Vision Solutions® VisionPro® VisionView®

The following are trademarks of Cognex Corporation:

3D-Locate™ 3DMax™ CheckPoint™ Cognex VSoC™ FFD™ iLearn™ InspectEdge™ Legend™ LineMax™ NotchMax™ ProofRead™ SmartAdvisor™ SmartSync™ SmartSystem™

Other product and company names mentioned herein are the trademarks, or registered trademarks, of their respective owners.



About this Manual	9
Networking	10
Connecting your DataMan to the Network.....	10
Connecting your DataMan Reader to the Network.....	10
Connecting your DataMan Base Station to the Network.....	10
Direct Connection to Your Computer	11
Configuring the DataMan to Reside on the Same Subnet as the PC.....	11
Configuring the PC to Reside on the Same Subnet as the DataMan.....	13
Connecting Your Reader across Subnets	16
Troubleshooting an Ethernet Connection.....	17
Industrial Network Protocols	19
EtherNet/IP	20
DMCC	20
Reader Configuration Code	20
Setup Tool.....	20
Getting Started	21
Object Model.....	24
Attributes.....	25
SoftEvents	27
General Fault Indicator	27
Services.....	27
Acquire Service.....	28
SendDMCC Service.....	28
GetDecodeResults Service.....	29
GetDecodeResults Request Data Format	29
Acquisition Sequence.....	29
Decode / Result Sequence.....	30
Behavior of DecodeStatusRegister.....	30
Results Buffering.....	31
Assembly Object	32
Input Assembly.....	32
Output Assembly	33
PCCC Object.....	33
Rockwell ControlLogix Examples.....	36
Implicit Messaging	36
Establishing an Implicit Messaging Connection.....	36

Accessing Implicit Messaging Connection Data	42
Verifying Implicit Messaging Connection Operation	45
Explicit Messaging	47
Issuing DMCC Commands	47
Rockwell CompactLogix Examples	51
Rockwell SLC 5/05 Examples	52
Setting up the PLC for Ethernet Communication	52
Message Instruction (MSG)	53
Sending DMCC Commands from an SLC 5/05	55
Message Instruction Results	58
Using the Generic EtherNet/IP Profile.....	58
Establishing a Generic Implicit Messaging Connection	58
Accessing Generic Implicit Messaging Connection Data	61
Examples.....	61
MC Protocol.....	63
DMCC	63
Reader Configuration Code	63
Setup Tool.....	63
MC Protocol Scanner	64
Getting Started	64
Network Configuration	65
Data Block Configuration.....	66
Interface	67
Control Block.....	67
Control Block Field Descriptions	68
Status Block	69
Status Block Field Descriptions	69
Input Data Block	70
Input Data Block Field Descriptions	70
Output Data Block.....	71
Output Data Block Field Descriptions	71
String Command Block	71
String Command Block Field Descriptions.....	72
String Command Result Block.....	72
String Command Result Block Field Descriptions	72
Operation	72

Scanning.....	72
Handshaking	73
Acquisition Sequence.....	74
Decode / Result Sequence.....	75
Results Buffering.....	75
SoftEvents	76
String Commands	76
General Fault Indicator	77
Examples	77
Function.....	77
Triggering a Read	78
Getting Read Results.....	78
Execute String Commands (DMCC)	79
Execute Soft Events	79
Modbus TCP	81
DMCC	81
Reader Configuration Code	81
Setup Tool.....	81
Modbus TCP Handler	82
Getting Started	82
Network Configuration	83
Data Block Configuration.....	83
Interface	84
Control Block.....	84
Control Block Field Descriptions	85
Status Block	86
Status Block Field Descriptions	86
Input Data Block	87
Input Data Block Field Descriptions	88
Output Data Block.....	88
Output Data Block Field Descriptions	88
String Command Block	88
String Command Block Field Descriptions.....	89
String Command Result Block.....	89
String Command Result Block Field Descriptions	89
Operation	89

Requests.....	89
Typical Sequence Diagram	90
Handshaking	90
Acquisition Sequence.....	90
Decode / Result Sequence.....	91
Results Buffering.....	92
SoftEvents	92
String Commands	93
General Fault Indicator	93
Examples	94
ApplicationLayer Example	94
Function.....	94
Transferring "Control" Register Data	94
Transferring Status Register Data	95
Transferring Output Data.....	95
DataManControl Example.....	95
Function.....	95
Triggering a Read	95
Getting Read Results.....	96
Execute String Commands (DMCC)	96
Execute Soft Events	96
PROFINET	97
DMCC	97
Reader Configuration Code	97
Setup Tool.....	98
Getting Started	98
Modules	103
Acquisition Control Module.....	104
Acquisition Status Module	105
Results Control Module.....	105
Results Status Module	106
Soft Event Control Module	106
User Data Module	107
Result Data Module.....	108
Operation	109
SoftEvents	109

General Fault Indicator	109
Acquisition Sequence.....	110
Decode / Result Sequence.....	111
Behavior of DecodeStatusRegister.....	111
Results Buffering.....	112
Siemens Examples.....	112
Symbol Table	112
Trigger and Get Results	114
Using Soft Events.....	118
Executing DMCC Commands.....	120
DataMan Application Development	122
DMCC Overview	122
Command Syntax.....	122
Command Header Syntax.....	122
Header Examples	122
Command	122
Commands	123
Parameters	123
Arguments.....	123
Footer	123
Reader Response	123
Examples	124
DMCC Application Development.....	124
DMCC .NET Contents	124
Building the DMCC .NET Sample Code	124
Using DMCC .NET.....	124
Enumerating DataMan Devices	125
Connecting to a DataMan Device	125
DataMan Device Settings.....	126
Displaying Static and Live Images from a DataMan Device.....	126

About this Manual

The *DataMan Communications and Programming Guide* provides information about how to integrate a DataMan device into your particular environment, including:

- Network configuration
- Industrial network protocols
- Integration with PLCs
- DataMan Control Commands (DMCC) API

Accordingly, the DataMan connected to a network can be triggered to acquire images by several methods. It can be done by the Setup Tool, it can be triggered by trigger bits or manipulating objects (industrial protocols), by external hard wired input or through DMCC command. This document provides a detailed description on how to do each.

Networking

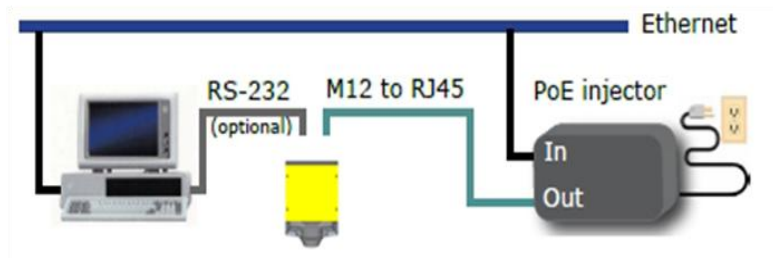
You can connect your DataMan via a simple Ethernet connection. You can either set the IP address and subnet mask of your DataMan manually or let them be configured automatically using DHCP.

Connecting your DataMan to the Network

Connecting your DataMan Reader to the Network

Supply power to the reader using a Power over Ethernet (PoE) injector. Cognex recommends the following connection sequence:

1. Connect the PoE injector to the Ethernet network (both ends of the patch cable).
2. Connect the power cord (AC 230V/110V) to the PoE injector.
3. Connect the reader to the PoE injector.

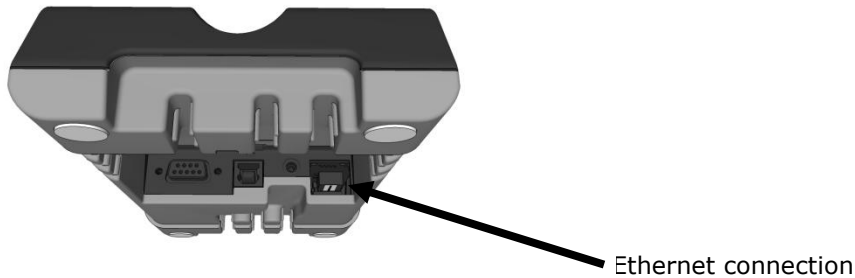


To disconnect the reader:

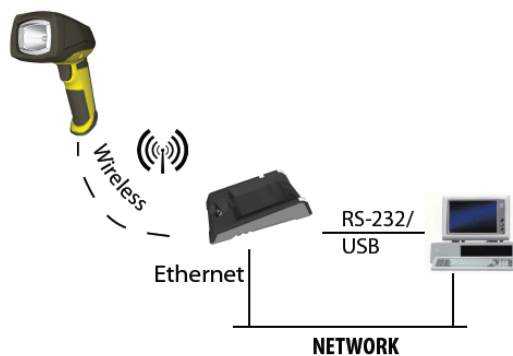
4. Disconnect the reader from the PoE injector.
5. Disconnect the power cord from the PoE injector.
6. Disconnect the PoE injector from the Ethernet network.

Connecting your DataMan Base Station to the Network

1. Power up your base station using one of these two options:
 - If you want to connect the Ethernet cable directly to the network or your PC, power up the base station using a 24V power supply.
 - If you want to use a Power Over Ethernet adapter, that will power up your base station.
2. Connect your base station to your PC with an Ethernet cable.



3. The base station becomes visible as connected through Ethernet, and it routes data through the wireless interface to the reader.



Direct Connection to Your Computer

When connecting a DataMan directly to an Ethernet port on a PC, both the PC and the DataMan must be configured for the same subnet. This can be done automatically through Link Local Addressing or you can manually configure your reader and your PC.

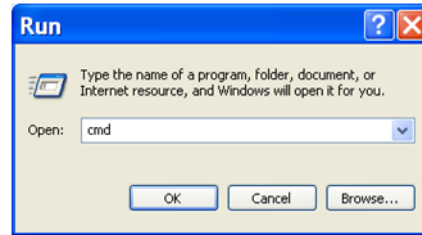
Link Local Addressing automatically requests and assigns an IP address. In the Setup Tool, this corresponds to the DHCP Server communication option. This is the default, you do not have to make any changes.

You can also manually configure your DataMan to reside on the same subnet as the PC or the other way round: configure your PC to reside on the same subnet as your DataMan. These options are detailed in the following sections.

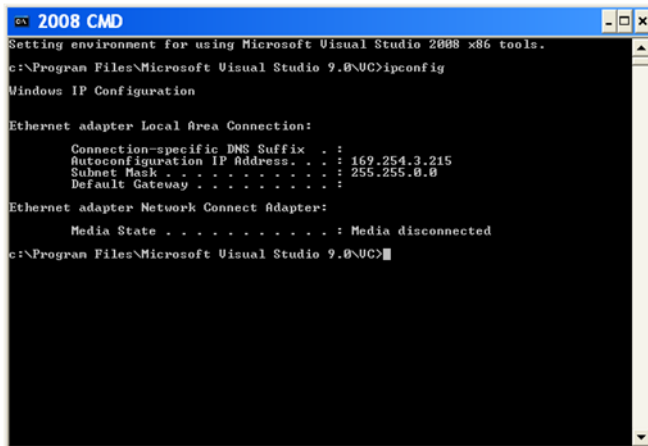
Configuring the DataMan to Reside on the Same Subnet as the PC

Perform the following steps to configure your DataMan device:

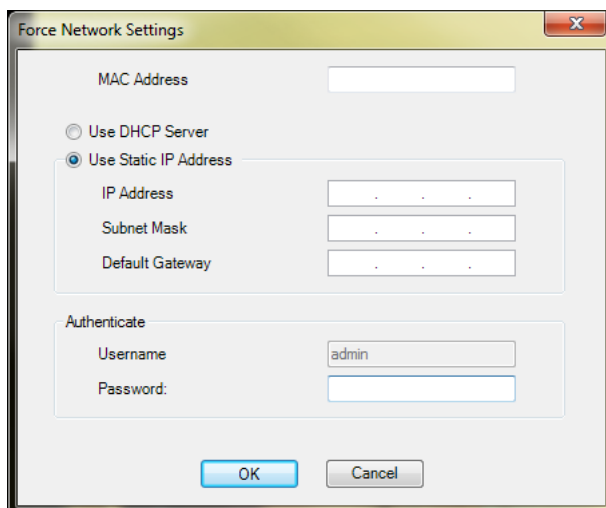
1. Use the **ipconfig** utility to determine the IP Address and subnet mask of your PC. In the Start menu, click Run...



2. In the Open field, type "cmd" and click OK.
3. In the command prompt window, type "ipconfig" and press Enter. A listing of all network adaptors on the PC is shown.



4. Record your PC's IP Address and Subnet Mask. In this example,
 - IP Address is 169.254.135.189
 - Subnet Mask is 255.255.0.0.
5. Go to the Setup Tool's **Connect to Reader** pane, and use the **Force Network Settings** dialog to manually configure the network settings on the target DataMan.
6. Click the **Force Network Settings** button. The **Force Network Settings** dialog opens.

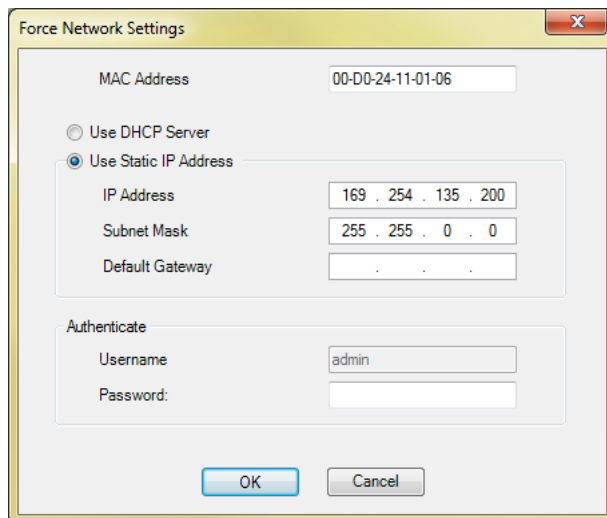


7. To force the network settings on your DataMan:
 - a. Enter the MAC address of the DataMan in the MAC Address field. The MAC Address of the DataMan can be found on the label of the reader or the base station.
 - b. Select **Use Static IP Address**.
 - c. Enter an IP Address and Subnet Mask that will be on the same subnet as the PC. Make sure this IP address is not yet in use (for example, test by pinging it).
 - Example IP Address: 169.254.135.200
 - Subnet Mask: 255.255.0.0

NOTE

The default Subnet Mask is 255.255.255.0. You can set it back to default by scanning the Reset Scanner to Factory Defaults Configuration Code.

Authentication should be left blank unless Authentication has been enabled on the DataMan. Authentication is disabled by default.



Click OK. Your DataMan is configured to the network settings specified, and it reboots automatically.

Your DataMan appears under the **Network devices** node after the address has been resolved. This can take up to 60 seconds.

8. If the device does not appear after 1 or 2 minutes, push the **Refresh** button on the Setup Tool's **Connect to Reader** pane. This will force the Setup Tool to scan for DataMan devices connected to the PC or connected to the same network.

Configuring the PC to Reside on the Same Subnet as the DataMan

If it is preferred that the DataMan network settings remain unchanged, you must already know the IP Address and Subnet Mask of the DataMan or you must connect to the DataMan via RS-232 to find them out. The DataMan IP Address and Subnet Mask can be found under **Communication Settings**.

The screenshot shows the 'Communication Settings' dialog box with the 'Network' tab selected. It features two radio buttons: 'Use DHCP Server' (unselected) and 'Use Static IP Address' (selected). Below the radio buttons, there are three input fields for static IP configuration: 'IP Address' (10 . 86 . 80 . 119), 'Subnet Mask' (255 . 255 . 255 . 0), and 'Default Gateway' (10 . 86 . 80 . 205).

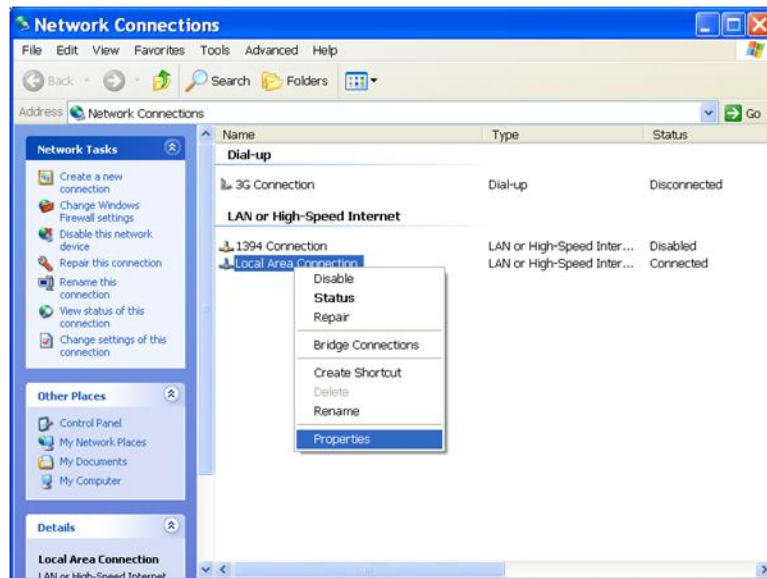
You can check (and configure) the Static IP Address of both the wireless reader and the base station under **Communication Settings** when you connect to the base station in the Setup Tool.

The screenshot shows the 'Communication Settings' dialog box with the 'Wireless' tab selected. It features two radio buttons: 'Use DHCP Server' (unselected) and 'Use Static IP Address' (selected). Below the radio buttons, there are two sections for static IP configuration: 'Base Station' and 'Wireless Reader'. Each section has 'IP Address' and 'Subnet Mask' input fields. The 'Base Station' IP is 10 . 86 . 80 . 183 and the 'Wireless Reader' IP is 10 . 86 . 80 . 184. Both have a Subnet Mask of 255 . 255 . 255 . 0. At the bottom, there is a 'Default Gateway' input field with the value 10 . 86 . 80 . 205.

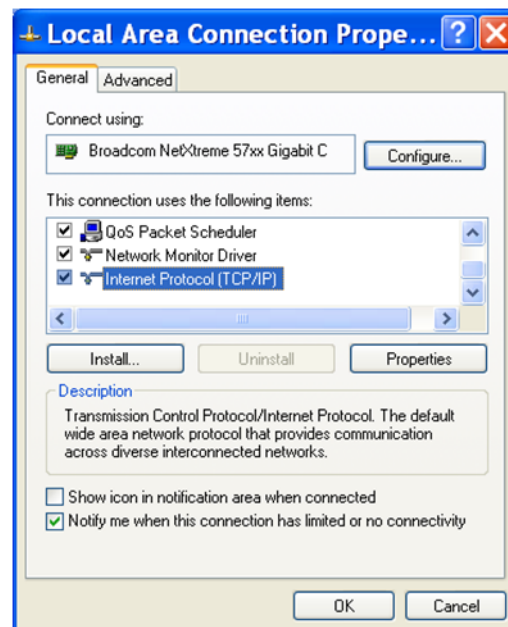
Once the IP Address and Subnet Mask of the DataMan are known, the PC's network settings can be changed.

Perform the following steps to configure your PC (examples here are of Windows XP):

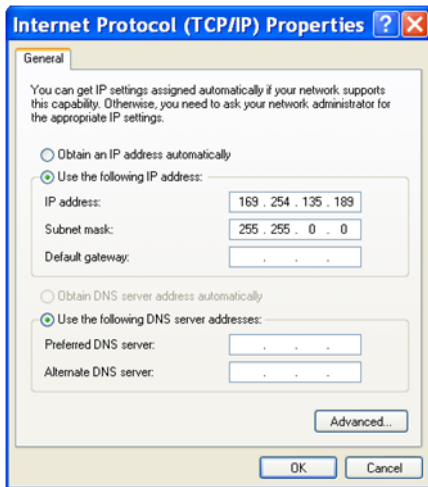
9. In the Start Menu, right click **My Network Places**, click the Properties menu option to launch **Network Connections**.
10. Right click on the network adaptor connected to the DataMan and select the **Properties** menu option.



11. Under the **General** tab, scroll down and select **Internet Protocol (TCP/IP)**, and click **Properties**.



12. Under the **General** tab, select the **Use the following IP address** radio option and enter an IP Address and Subnet Mask that are on the same subnet as your DataMan. Click OK.

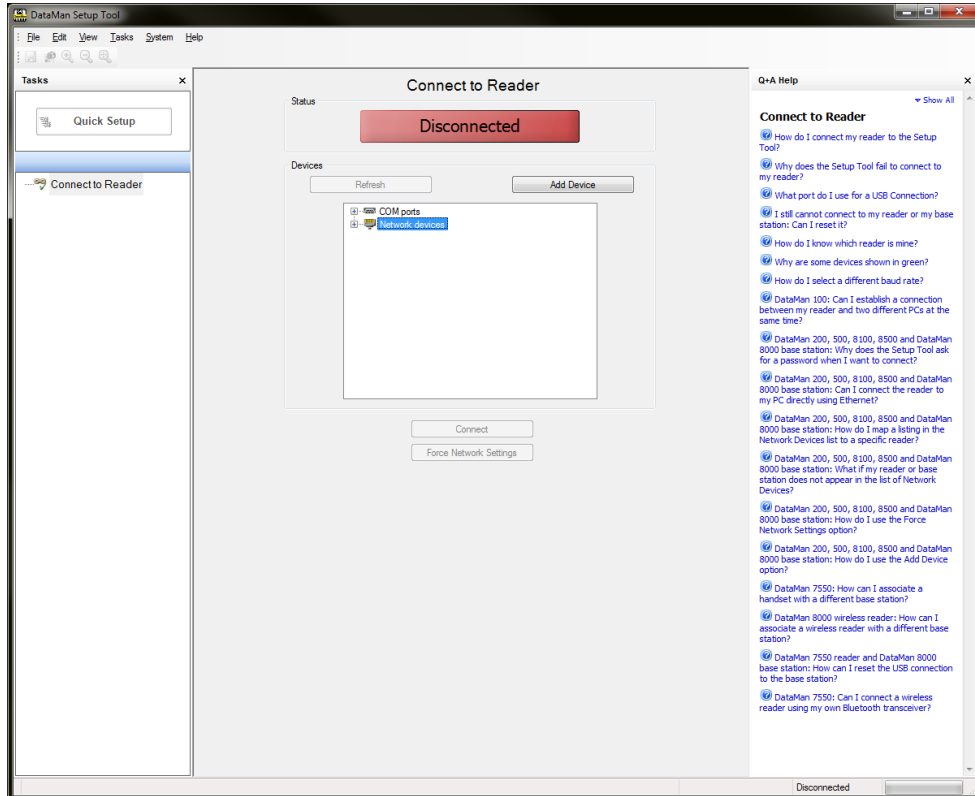


13. Click Close. The network settings of your PC will change to the new specified values.
14. Reboot the DataMan. It appears under the **Network devices** node on the **Connect to Reader** pane after the network address has been resolved.
15. If the device does not appear after 1 or 2 minutes, click the **Refresh** button on the Setup Tool's **Connect to Reader** pane. The Setup Tool scans for DataMan devices connected to the PC or connected to the same network.

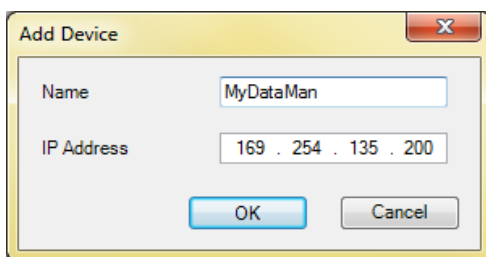
Connecting Your Reader across Subnets

The following options can be used to connect to the DataMan with the Setup Tool across subnets if you already know the IP Address of the DataMan.

1. In the Setup Tool's **Connect to Reader** pane, click **Add Device**.



2. Enter a name and the actual IP Address of the target DataMan. The name has no effect upon the DataMan. It is only used as an identifier to list the target DataMan under the **Network devices** node.



3. Click OK. The name appears under the **Network devices** node. Double click the new node or highlight it and click the **Connect** button. If the device is available, you will be connected (a DataMan 200 is connected in this example).

Troubleshooting an Ethernet Connection

Based on your network configuration, the Setup Tool may not be able to communicate with the DataMan device and it will not appear in the list of **Network devices**.

First check your Ethernet connection with the reader and click **Refresh** in the Setup Tool.

If you are using a tethered device (not a base station or a wireless reader), scan the **Enable DHCP** code in the *DataMan Configuration Codes* document available from the Start menu. This might allow the reader to acquire a suitable IP address from a DHCP server on your subnet.

If the device still does not appear, you can use either the **Add Device** or **Force Network Settings** options in the Setup Tool.

If you know the IP address of the device, use the **Add Device** option. If you do not know the IP address, use the **Force Network Settings** options. Either method should allow your DataMan to appear in the list of **Network devices** so that you can connect to it through the Setup Tool and your Ethernet connection.

You can also use the RS-232 connection to configure the device with parameters that allow it to communicate over your Ethernet network.

Industrial Network Protocols

DataMan uses industrial network protocols that are based on standard Ethernet protocols. These protocols: EtherNet/IP, PROFINET, MC Protocol and Modbus/TCP are enhanced to provide more reliability than standard Ethernet.

EtherNet/IP

DataMan supports EtherNet/IP™, an application level protocol based on the Common Industrial Protocol (CIP). EtherNet/IP provides an extensive range of messaging options and services for the transfer of data and I/O over Ethernet. All devices on an EtherNet/IP network present their data to the network as a series of data values called attributes. Attributes can be grouped with other related data values into sets, these are called Assemblies.

By default the DataMan has the EtherNet/IP protocol disabled. The protocol can be enabled via DMCC, scanning a reader programming code, or in the Setup Tool.

DMCC

The following commands can be used to enable/disable EtherNet/IP on the DataMan. The commands may be issued via RS-232 or Telnet connection.

NOTE

Because you have to make changes to the Telnet client provided by Windows to communicate with DataMan, it is recommended you use third party clients such as PuTTY.

Enable:

```
||>SET ETHERNET-IP.ENABLED ON  
||>CONFIG.SAVE  
||>REBOOT
```

Disable:

```
||>SET ETHERNET-IP.ENABLED OFF  
||>CONFIG.SAVE  
||>REBOOT
```

Reader Configuration Code

Scanning the following reader configuration codes will enable/disable EtherNet/IP.

NOTE

You must reboot the device for the change to take effect.



Setup Tool

EtherNet/IP can be enabled by checking **Enabled** on the Industrial Protocols pane's EtherNet/IP tab. Make sure you save the new selection by clicking Yes to the Reboot Required message window.

NOTE

The new settings take effect only after the reader is rebooted.

Getting Started

Preparing to use EtherNet/IP involves the following main steps:

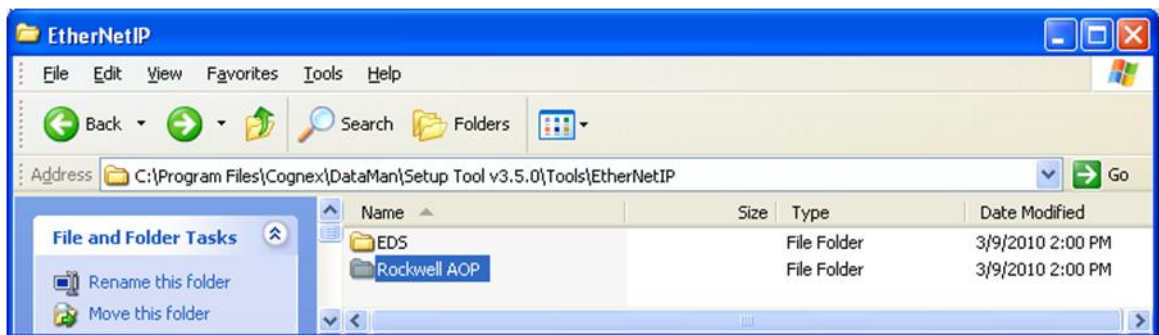
- Make sure you have the Rockwell Software tool on your machine.
- Set up the Rockwell Software tool so that it recognizes your DataMan device.
- Install the DataMan Electronic Data Sheet (EDS) for the DataMan reader.

Perform the following steps to set up EtherNet/IP:

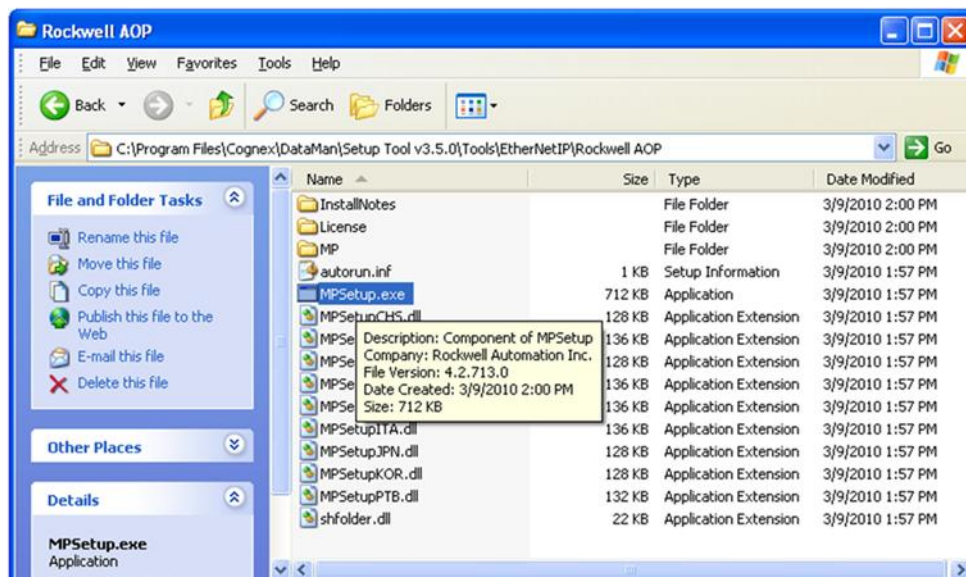
1. Verify that the Rockwell Software is on your PC.
2. Make sure you select the Add on Profile installation and the Samples installation. Add on Profile is only used with Rockwell ControlLogix or CompactLogix PLCs.
3. Install the Rockwell Add on Profiles by navigating to the following directory.

NOTE

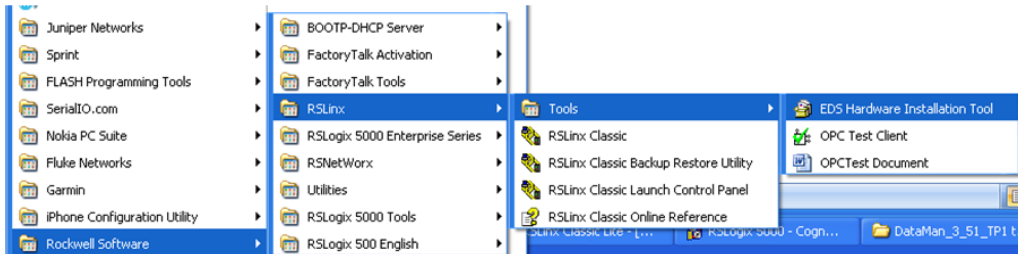
Adjust the path for the specific Setup Tool version that you are using.



4. Select the Rockwell AOP directory:



5. If you have not previously installed the Rockwell AOP, now run MPSetup.exe.
6. From the Start menu, go to Programs → Rockwell Software → RSLinx → Tools → EDS Hardware Install Tool.

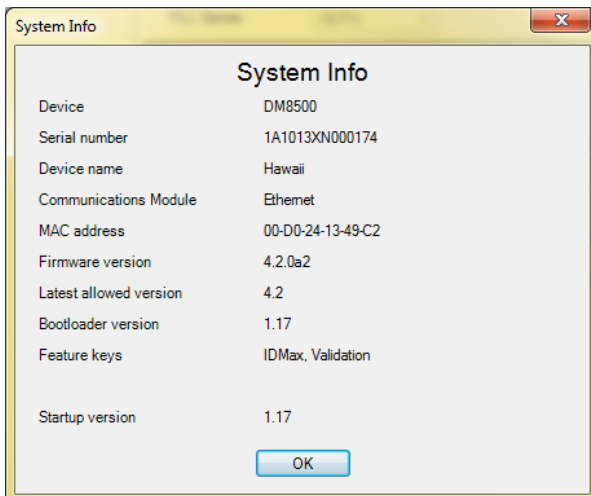


7. Run the ESD Install tool.

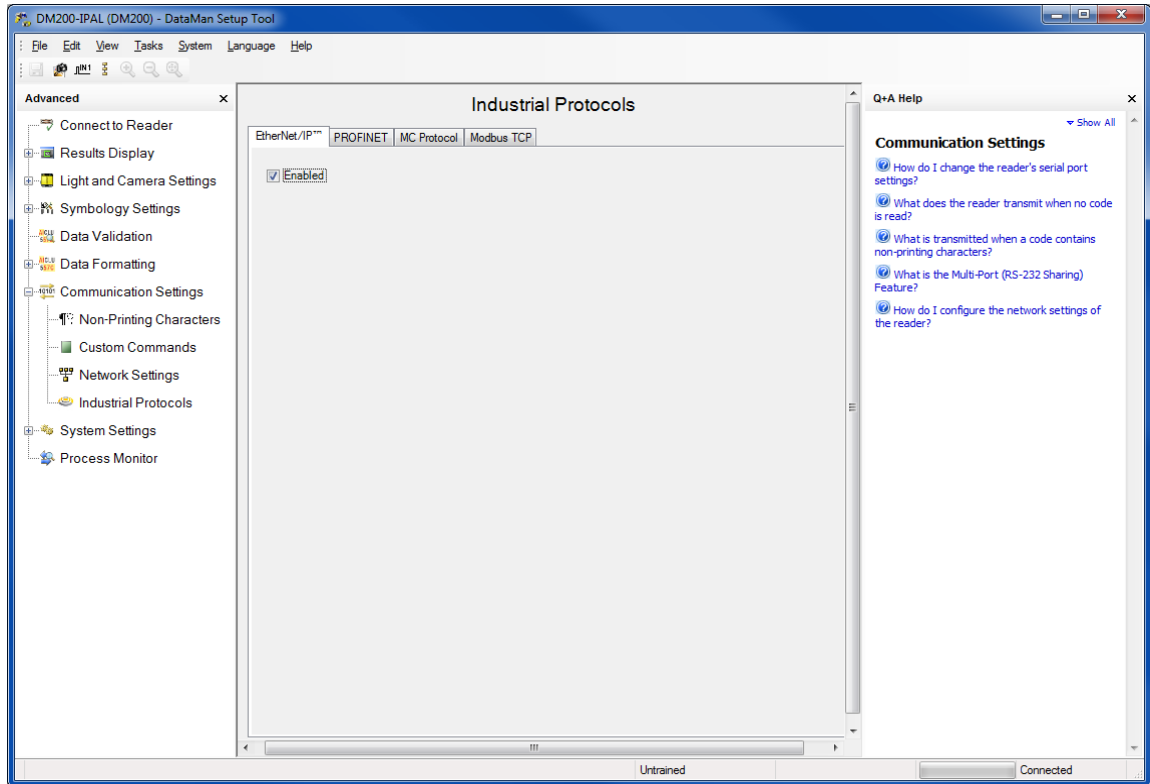
NOTE

If you have an existing EDS file, uninstall it first, then install the latest version of the EDS.

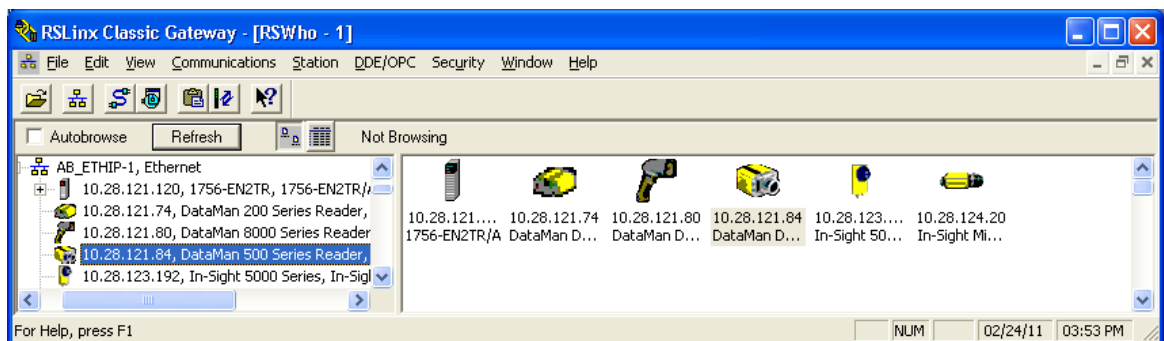
8. Run the Setup Tool and update the DataMan firmware.
9. Check if the firmware has been loaded into the unit by clicking in the Setup Tool View → System Info.



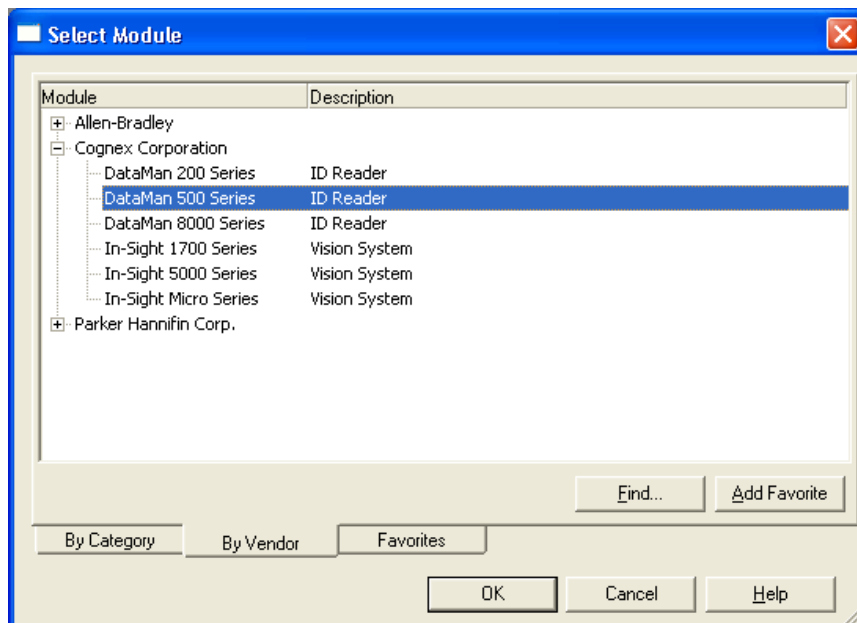
In the Setup Tool, go to the Industrial Protocols pane and check the **Enabled** checkbox on the **EtherNet/IP™** tab.



10. In order for the changes to take effect, you must save your settings and cycle power. In the Setup Tool menu, click System → Save Settings.
11. Reboot your reader.
12. Your DataMan is visible now in the RSWHO.



- If your DataMan is visible, but the icon is a question mark, repeat the EDS Installation.
13. Open one of the sample jobs and integrate your DataMan into your program using the Add on Profile.
 14. Alternatively, you can add the DataMan as a Module on your network.



Object Model

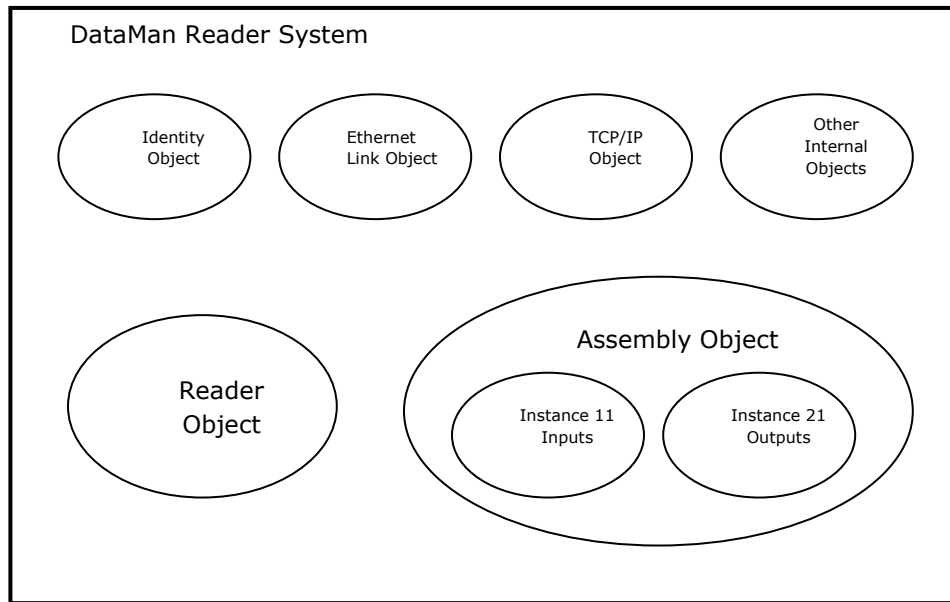
The ID Reader Object is a vendor specific object class. This means it is not part of the CIP common (public) architecture but rather an extension. It is a custom object that Cognex has added to the EtherNet/IP architecture on the DataMan device. This object models all data and functionality available in the DataMan reader. This includes triggering, status, events, errors and result data.

The ID Reader Object is identified by its vendor specific class code:

DataMan ID Reader Object Class Code: 0x79

Objects are made up of attributes (data) and services (functionality). These can be defined at the class level (common to all instances of the class) or the instance level (unique to an individual instance). There are common attributes and services defined by the CIP specification that apply to all objects (often these are optional). Vendors may also define their own attributes and services for their vendor specific classes.

The ID Reader Object attributes and services can be individually accessed via explicit messaging. Also a number of the ID Reader Object attributes are exposed in the DataMan assembly objects which allow them to be accessed as a group via implicit messaging.



Attributes

The DataMan ID Reader Object (Class Code: 0x79) has the following attributes.

Attribute ID	Access Rule	Name	Data Type	Description
0x9	Set	AcqTriggerEnable	BOOL	0 = EtherNet/IP triggering is disabled 1 = EtherNet/IP triggering is enabled
0xA	Set	AcqTrigger	BOOL	Acquire an image when this attribute changes from 0 to 1.
0xB	Get	AcqStatusRegister	BYTE	Bit0: Trigger Ready Bit1: Trigger Ack Bit2: Acquiring Bit3: Missed Acquisition Bit4-7: Reserved
0xC	Set	UserData	ARRAY of BYTE	User defined data that may be used as an input to the acquisition/decode.
0xD	Set	BufferResultsEnable	BOOL	When true, it enables buffering of the decode results.
0xE	Get	DecodeStatusRegister	BYTE	Bit0: Decoding Bit1: Decode completed (toggle) Bit2: Results buffer overrun Bit3: Results available Bit4: Reserved Bit5: Reserved Bit6: Reserved Bit7: General fault indicator
0xF	Set	DecodeResultsAck	BOOL	Acknowledges that the client received the decode results.

Attribute ID	Access Rule	Name	Data Type	Description
0x10	Get	DecodeResults	STRUCT of	The last decode results
		ResultsID	UINT	Decode results identifier. Corresponds to the TriggerID of the decoded image.
		ResultCode	UINT	Decode result summary code value Bit0: 1=Read, 0=No read Bit1: 1=Validated, 0=Not Validated Bit2: 1=Verified, 0=Not Verified Bit3: 1=Acquisition trigger overrun Bit4: 1=Acquisition buffer overrun Bit5-15: Reserved (future use)
		ResultExtended	UINT	Extended result information
		ResultLength	UINT	Current number of result data bytes.
		ResultData	ARRAY of BYTE	Result data from last decode
0x12	Set	SoftEvents	BYTE	SoftEvents act as virtual inputs (execute action on 0 to 1 transition) Bit0: Train code Bit1: Train match string Bit2: Train focus Bit3: Train brightness Bit4: Un-Train Bit5: Reserved (future use) Bit6: Execute DMCC command Bit7: Set match string
0x15	Get	TriggerID	UINT	Trigger identifier. ID of the next trigger to be issued.
0x16	Set	UserDataOption	UINT	Optional user data information
0x17	Set	UserDataLength	UINT	Current number of user data bytes.

Attribute ID	Access Rule	Name	Data Type	Description
0x18	Get	SoftEventAck	BYTE	Acknowledgment of SoftEvents. Bit0: Train code ack Bit1: Train match string ack Bit2: Train focus ack Bit3: Train brightness ack Bit4: Un-Train ack Bit5: Reserved (future use) Bit6: Execute DMCC command ack Bit7: Set match string ack

SoftEvents

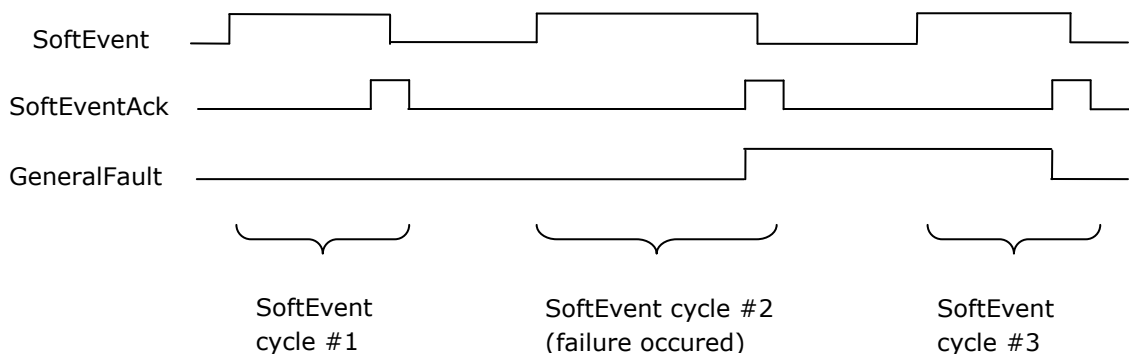
SoftEvents act as “virtual” inputs. When the value of a SoftEvent changes from 0 → 1 the action associated with the event will be executed. When the action completes the corresponding SoftEventAck bit will change from 1 → 0 to signal completion.

The SoftEvent and SoftEventAck form a logical handshake. After SoftEventAck changes to 1 the original SoftEvent should be set back to 0. When that occurs SoftEventAck will automatically be set back to 0.

The “ExecuteDMCC” and “SetMatchString” soft event actions require user supplied data. This data must be written to the UserData & UserDataLength area of the Input Assembly prior to invoking the soft event. Since both of these soft events depend on the UserData, only one may be invoked at a time.

General Fault Indicator

When a communication related fault occurs the “GeneralFault” bit will change from 0 → 1. Currently the only fault conditions supported are soft event operations. If a soft event operation fails, the fault bit will be set. The fault bit will remain set until the next successful soft event operation. Or, until TriggerEnable is set to 0 and then back to 1.



Services

The ID Reader Object supports the following Common CIP services.

Service Code	Service Name	Description
0x05	Reset	Resets the ID Reader object
0x0E	Get_Attribute_Single	Returns the contents of the specified attribute.
0x10	Set_Attribute_Single	Modifies the specified attribute

The ID Reader Object supports the following vendor specific services.

Service Code	Service Name	Description
0x32	Acquire	Triggers a single acquisition
0x34	SendDMCC	Sends a DMCC command to the device
0x35	GetDecodeResults	Gets the content of the DecodeResults attribute

Acquire Service

The Acquire Service will cause an acquisition to be triggered (if the acquisition system is ready to acquire an image). If the acquisition could not be triggered, then the Missed Acquisition bit of the AcqStatusRegister will be set until the next successful acquisition.

SendDMCC Service

The SendDMCC Service sends a DMCC command string to the device. The request data consists of the DMCC command string that is to be sent to the reader. The reply data will contain the string result of the DMCC command. Additionally the service provides a numeric result status for the call. Most of these result codes relate to the basic success/failure of the service execution. However, the service also maps the actual DMCC status codes. This allows the PLC to interpret the service request without having to parse the actual DMCC return string.

Service Return Code	Description	DMCC Return Code
0	Success – No error	0
1	Bad Command	-
4	No Answer – System too busy	-
100	Unidentified error	100
101	Command invalid	101
102	Parameter invalid	102
103	Checksum incorrect	103
104	Parameter rejected/alterd due to reader state	104
105	Assigned wireless reader is not available for the base station	105

NOTE

The string must be in the CIP STRING2 format (16-bit integer indicating the string length in characters followed by the actual string characters, no terminating null required).

GetDecodeResults Service

The GetDecodeResults service reads data from the DecodeResults attribute of the ID Reader Object. This service takes parameters indicating the "size" (number of bytes to read) and the "offset" (offset into the DecodeResults attribute to begin reading). This gives the service the flexibility to be used with PLC's that have different restrictions on the amount of data allowed in an explicit message. It also allows the user to access very large codes that cannot be completely transferred with implicit messaging (assembly object).

GetDecodeResults Request Data Format

Name	Type	Description
Size	UINT	The number of bytes of the DecodeResults attribute to read
Offset	UINT	The offset into the DecodeResults attribute. This specifies the first byte of the DecodeResults attribute to begin reading (0 based offset).

Acquisition Sequence

DataMan can be triggered to acquire images by several methods. It can be done implicitly via the Assembly object. Or done explicitly via the ID Reader object. When using explicit messaging it can be done in a single step by accessing the Acquire Service, it can also be done by directly manipulating the ID Reader object attributes (AcqTrigger and AcqStatusRegister) and finally it can be done via DMCC command. The ID Reader attributes will be discussed here but these same values can be accessed via the assembly objects.

On startup the AcqTriggerEnable attribute will be False. It must be set to True to enable triggering. When the device is ready to accept triggers, the Trigger Ready bit in the AcqStatusRegister will be set to True.

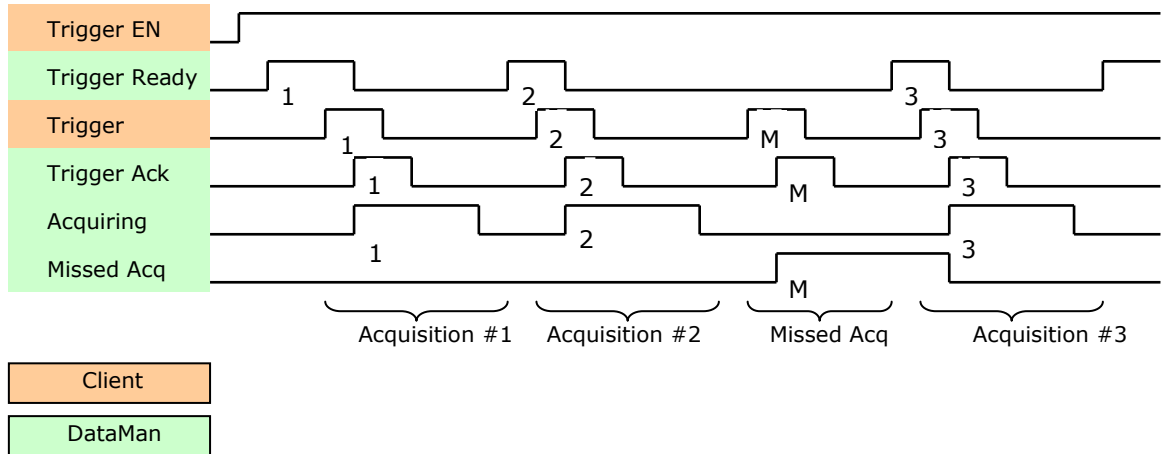
While the AcqStatusRegister "Trigger Ready" bit is True, each time the ID Reader object sees the AcqTrigger attribute change from 0 to 1, it will initiate an image acquisition. When setting this via the assembly objects, the attribute should be held in the new state until that same state value is seen in the Trigger Ack bit of the AcqStatusRegister (this is a necessary handshake to guarantee that the change is seen by the ID Reader object).

During an acquisition, the Trigger Ready bit in the AcqStatusRegister will be cleared and the Acquiring bit will be set to True. When the acquisition is completed, the Acquiring bit will be cleared. The Trigger Ready bit will again be set True once the device is ready to begin a new image acquisition.

If results buffering is enabled, the device will allow overlapped acquisition and decoding operations. Trigger Ready will be set high after acquisition is complete but while decoding is still in process. This can be used to achieve faster overall trigger rates. If result buffering is not enabled, the Trigger Ready bit will remain low until both the acquisition and decode operations have completed.

As a special case, an acquisition can be cancelled by clearing the Trigger signal before the read operation has completed. This allows for the cancellation of reads in Presentation and Manual mode if no code is in the field of view. To ensure that a read is not unintentionally

cancelled, it is advised that the PLC hold the Trigger signal True until both TriggerAck and ResultsAvailable are True (or DecodeComplete toggles state).



To force a reset of the trigger mechanism set the AcqTriggerEnable attribute to False, until the AcqStatusRegister is 0. Then, AcqTriggerEnable can be set to True to re-enable acquisition.

Decode / Result Sequence

After an image is acquired it is decoded. While being decoded, the Decoding bit of the DecodeStatusRegister is set. When the decode is complete, the Decoding bit is cleared and the Decode Completed bit is toggled.

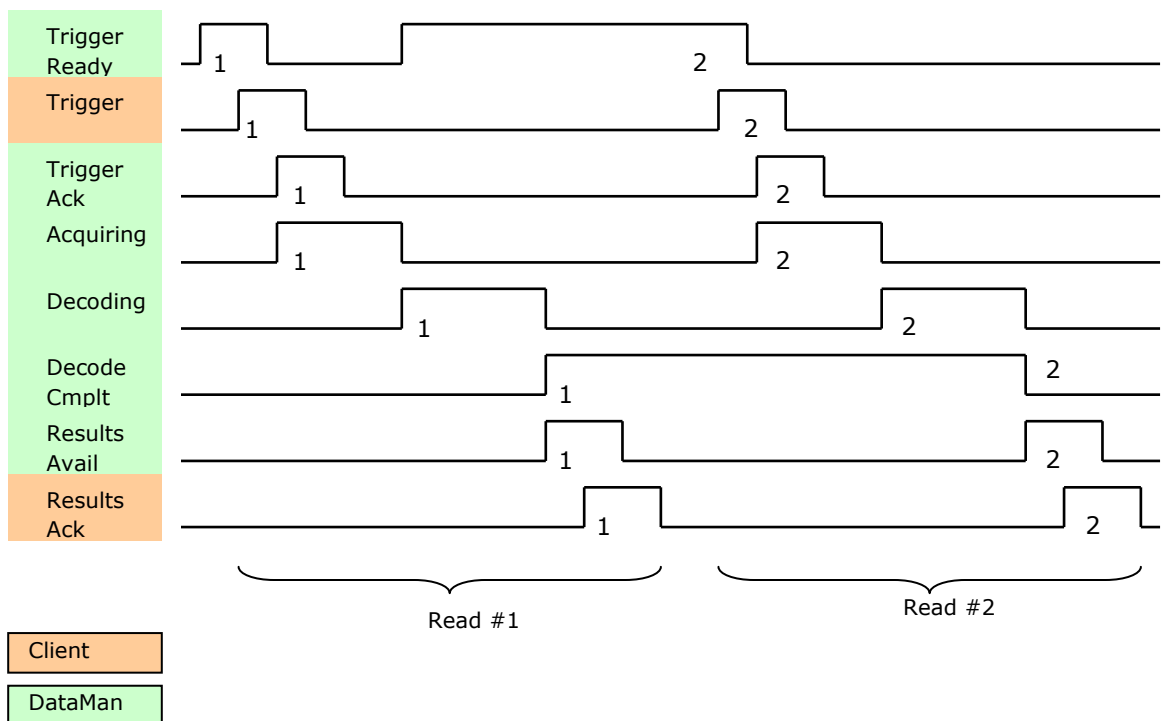
The BufferResultsEnable attribute determines how decode results are handled by the ID Reader Object. If the BufferResultsEnable attribute is set to False, then the decode results are immediately placed into the DecodeResults attribute and Results Available is set to True.

If the BufferResultsEnable attribute is set to True the new results are queued. The earlier decode results remain in the DecodeResults attribute until they are acknowledged by the client setting the DecodeResultsAck attribute to True. After the Results Available bit is cleared, the client should set the DecodeResultsAck attribute back to False to allow the next queued results to be placed in to the DecodeResults attribute. This is a necessary handshake to ensure the results are received by the DataMan reader's client (PLC).

Behavior of DecodeStatusRegister

Bit	Bit Name	Results if Buffering Disabled	Results if Buffering Enabled
1	Decoding	Set when decoding an image.	Set when decoding an image.
2	Decode Complete	Toggled on completion of an image decode.	Toggled on completion of an image decode.

Bit	Bit Name	Results if Buffering Disabled	Results if Buffering Enabled
3	Results Buffer Overflow	Remains set to zero.	Set when decode results could not be queued because the client failed to acknowledge a previous result. Cleared when the decode result is successfully queued.
4	Results Available	Set when new results are placed in the DecodeResults attribute. Stays set until the results are acknowledged by setting DecodeResultsAck to true.	Set when new results are placed in the DecodeResults attribute. Stays set until the results are acknowledged by setting DecodeResultsAck to true.



Results Buffering

There is an option to enable a queue for decode results. If enabled this allows a finite number of decode result data to queue up until the client (PLC) has time to read them. This is useful to smooth out data flow if the client (PLC) slows down for short periods of time.

Also, if result buffering is enabled the device will allow overlapped acquisition and decode operations. Depending on the application this can be used to achieve faster over all trigger rates. See Acquisition Sequence description above for further detail.

In general, if reads are occurring faster than results can be sent out the primary difference between buffering or not buffering is determining which results get discarded. If buffering

is not enabled the most recent results are kept and the earlier result (which was not read by the PLC fast enough) is lost. Essentially the more recent result will simply over write the earlier result. If buffering is enabled (and the queue becomes full) the most recent results are discarded until room becomes available in the results queue.

Assembly Object

Assemblies are combinations of selected attributes (data items) from CIP objects with in a device. The device vendor defines assemblies according to their needs. They combine data together in useful groupings according to the requirements of the application.

The designation of Input & Output assembly can be confusing. DataMan is an I/O adapter class device. The convention for adapters is that Input Assemblies produce (transmit) data for another device (i.e. DataMan → PLC) and Output Assemblies consume (receive) data from another device (i.e. PLC → DataMan). Essentially DataMan acts as an I/O module for another device such as a PLC.

Assembly objects use implicit messaging. In the abstract they are just blocks of data which are transmitted as the raw payload of implicit messaging packets. These implicit messaging packets are produced (transmitted) repeatedly at a predefined chosen rate (100ms, 200ms, etc).

DataMan readers have a single input assembly and single output assembly. These assemblies combine selected attributes (data) of the DataMan ID Reader Object into groupings that minimize network bandwidth and still allow for efficient control and processing. The data in these assemblies can also be accessed individually from the ID Reader Object. However, using the assembly objects is much more efficient. This is the reason that they are the primary means of runtime communication between a DataMan reader and a PLC.

Input Assembly

The Input assembly provides status information, process state, and decode results.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
11	0	Reserved				Missed Acq	Acquiring	Trigger Ack	Trigger Ready
	1	General Fault	Reserved			Results Available	Results Buffer Overrun	Decode Complete Toggle	Decoding
	2	Soft Event Ack 7	Soft Event Ack 6	Soft Event Ack 5	Soft Event Ack 4	Soft Event Ack 3	Soft Event Ack 2	Soft Event Ack 1	Soft Event Ack 0
	3 - 5	Reserved							
	6	Trigger ID (16-bit integer)							
	7								
	8	Result ID (16-bit integer)							
	9								
	10	Result Code (16-bit integer)							

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	11	Result Extended (16-bit integer)							
	12								
	13								
	14	Result Data Length (16-bit integer)							
	15								
	16	Result Data 0							
	...								
	499	Result Data 483							

Output Assembly

The Output assembly contains control signals, software event signals, and any user data required for the trigger & decode.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
21	0	Reserved				Results Ack	Buffer Results Enable	Trigger	Trigger Enable
	1	Soft Event 7	Soft Event 6	Soft Event 5	Soft Event 4	Soft Event 3	Soft Event 2	Soft Event 1	Soft Event 0
	2	Reserved							
	3								
	4	User Data Option (16-bit integer)							
	5								
	6	User Data Length (16-bit integer)							
	7								
	8	User Data 0							
	...								
	499	User Data 491							

PCCC Object

DataMan has limited support for the Rockwell PCCC object. This allows legacy PLC's (PLC-5, SLC, etc) to communicate with DataMan using their native PCCC command set and explicit messaging. The PCCC object allows DataMan to look like a Rockwell PLC-5 logic controller.

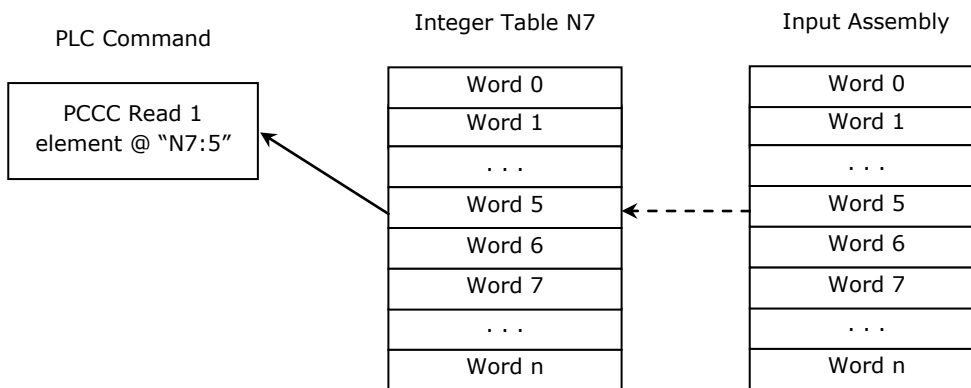
PCCC commands are organized to work with "data tables" that exist in legacy logic controllers. Each data table is an array of a give data type (BYTE, INT, FLOAT, etc). The commands are oriented to read/write one or more data items of a given data table. Items are addressed by specifying the data table and the index of the item in the table (indexes base from 0). For instance to read the 6th integer in PLC data table you would send the PCCC command to read N7:5. "N" specifies an integer table, "7" is the table number in the

PLC (each table has a unique numeric identifier – assigned when the user PLC program was created), and “5” is the index into the table (note indexes begin at 0).

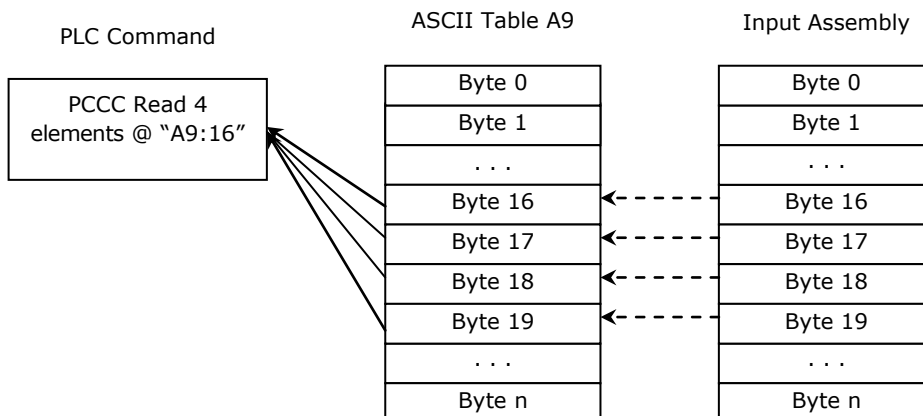
The PCCC object in DataMan maps the read and write requests to ID Reader assemblies (or in one special case to the DMCC service). Read commands return data from the Input assembly (instance 11). Write commands send data to the Output assembly (instance 21). In essence the PCCC Object gives the outward appearance of PLC-5 data tables but is actually accessing the assembly data. Currently the implementation only supports an Integer data table (N7) and an ASCII data table (A9). There is one special case of String data table (ST10:0) for DMCC.

Table	Data Type	Table Size
N7	Integer (16-bit)	250 elements
A9	ASCII (8-bit)	500 elements
ST10	String	1 element

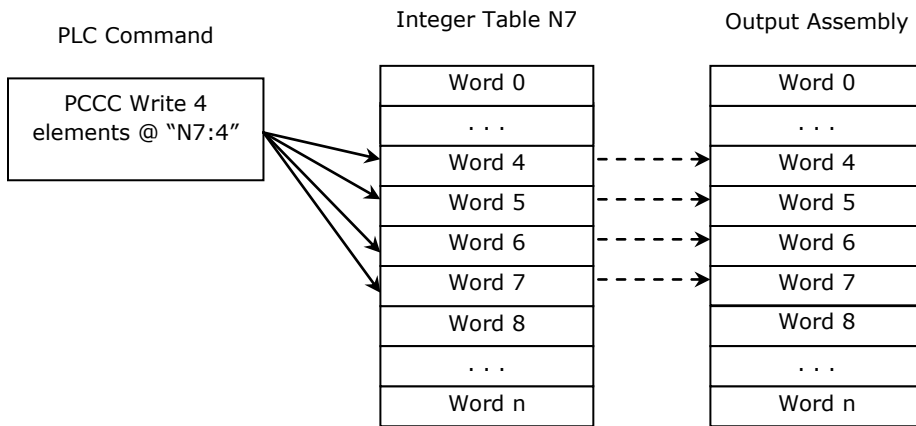
The ResultCode value is located at word offset 5 (counting from 0) of the Input Assembly. To access this value you would issue the following PLC command.



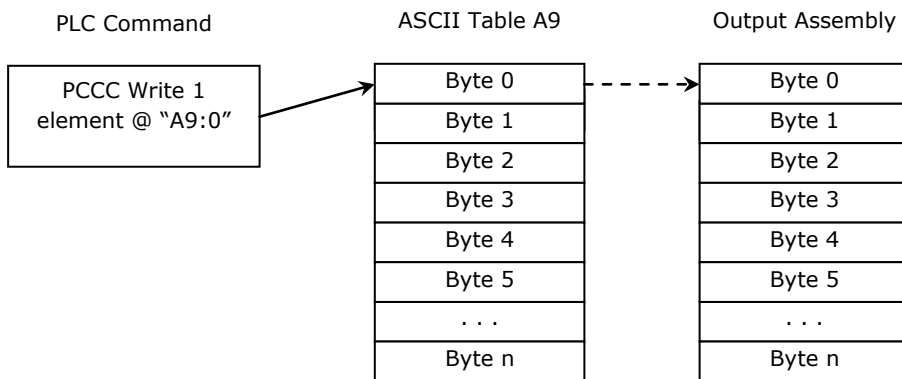
The decode ResultData begins at byte offset 16 (counting from 0) of the Input Assembly. To read the first 4 bytes of result data you would issue the following PLC command.



The UserData begins at word offset 4 (counting from 0) of the Output Assembly. To write 4 words of UserData you would issue the following PLC command.



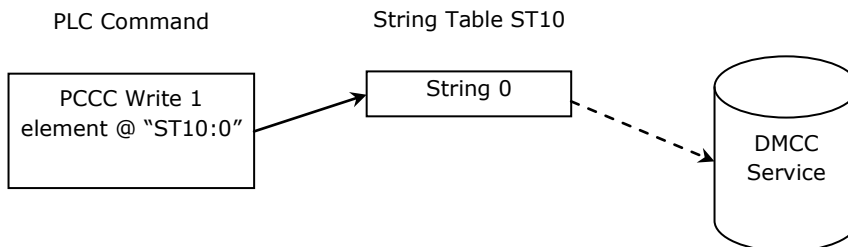
The bit to trigger an acquisition is in byte offset 0 of the Output Assembly. To write to this byte you would issue the following PLC command.



The PCCC Object supports a special case mapping of a string table element (ST10:0) to the DMCC service. Any string written to ST10:0 will be passed to the DMCC service for processing. This allows PCCC write string commands to be used to invoke DMCC commands.

NOTE

The string table is only one element in size. Writing to the other elements will return an error.



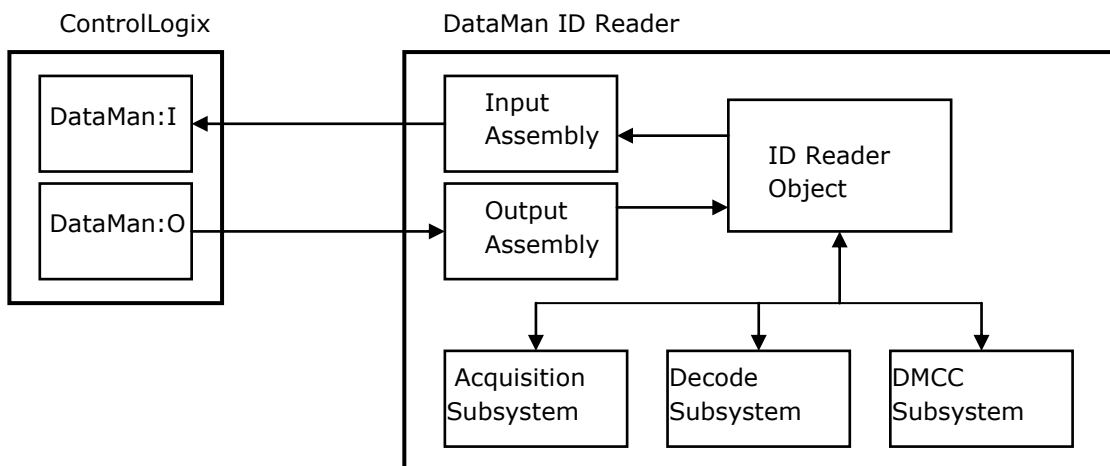
Rockwell ControlLogix Examples

Implicit Messages transmit time-critical application specific I/O data, and can be point-to-point or multicast. Explicit messages require a response from the receiving device. As a result, explicit messages are better suited for operations that occur less frequently. An instruction to send a DMCC command is an example of an explicit message.

Implicit Messaging

EtherNet/IP implicit messaging allows a DataMan reader's inputs and outputs to be mapped into tags in the ControlLogix PLC. Once these connections are established the data is transferred cyclically at a user defined interval (10ms, 50ms, 100ms, etc).

The figure below represents Ethernet-based I/O through EtherNet/IP:



The *Input Assembly* and *Output Assembly* map various attributes (data) from the ID Reader object: The *Input Assembly* is the collection of DataMan reader data values sent to the PLC (PLC inputs); and the *Output Assembly* is the collection of data values received by the DataMan reader from the PLC (PLC outputs).

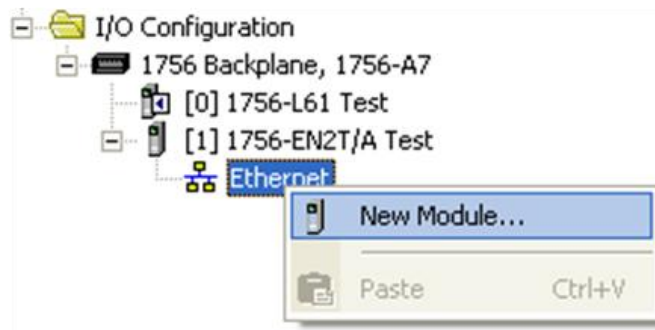
Establishing an Implicit Messaging Connection

To setup an EtherNet/IP implicit messaging connection between a DataMan and a ControlLogix controller, the DataMan reader must first be added to the ControlLogix I/O Configuration tree. The most efficient method is to use the Add-On-Profile. This example assumes that the Add-On-Profile has already been installed. If you do not have the Add-On-Profile, see Section [Using the Generic EtherNet/IP Profile](#).

To establish an implicit messaging connection with a ControlLogix PLC:

1. Open RSLogix5000 and load your project (or select "File->New..." to create a new one).

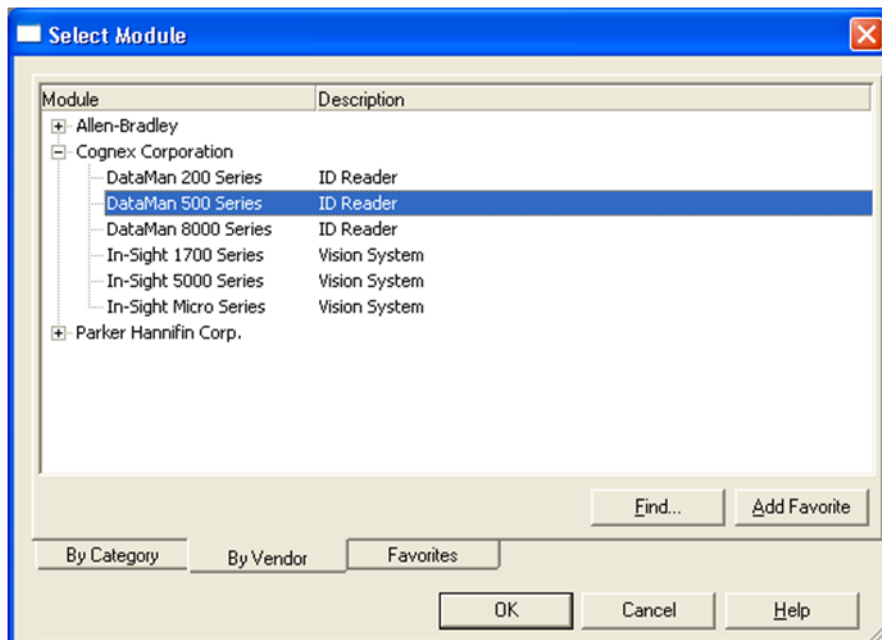
From the I/O Configuration node, select the *Ethernet* node under the project Ethernet Module, right-click on the icon and select New Module from the menu:



- From the Select Module dialog, choose your model of DataMan ID Reader from the list.

NOTE

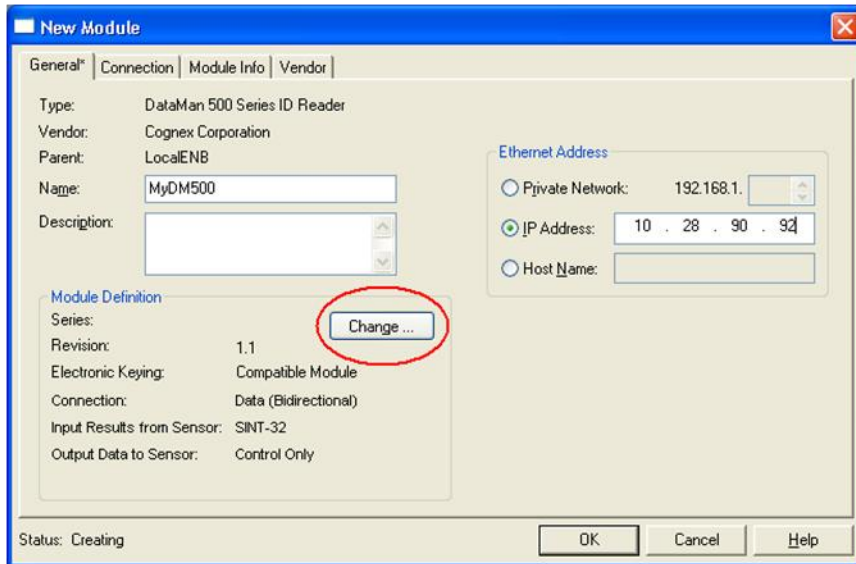
This option will only be available after the DataMan Add-On Profile has been installed.



NOTE

The remainder of the steps is identical regardless of which DataMan model is selected.

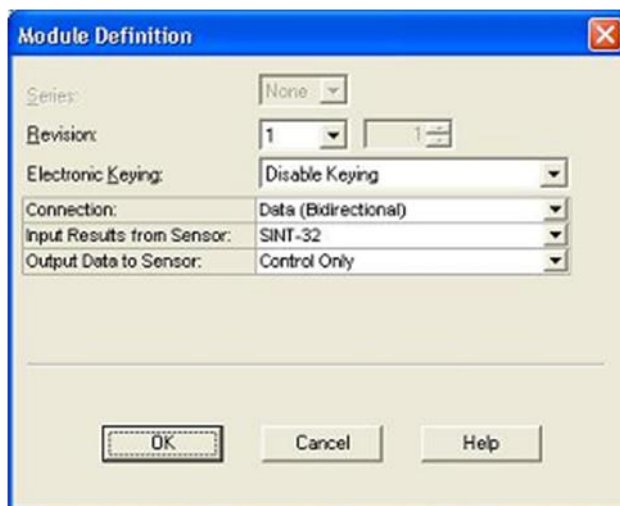
- After the selection is made, the configuration dialog for the DataMan ID Reader system will be displayed. Give the module a name and enter the DataMan's IP address. The default is a bidirectional (send/receive) connection consisting of control, status, and 32 bytes of result data with keying disabled. To change this default connection, select the "Change..." button. If no change is required skip over the next step.



4. Change the connection configuration.

Selecting the "Change..." button will bring up the Module Definition dialog. This dialog is used to alter the connection configuration. You can change:

- DataMan revision
- Electronic keying
- Connection type (bidirectional/receive-only)
- Amount of data received (from the DataMan)
- Amount of data sent (to the DataMan)



Electronic Keying: Defines the level of module type checking that is performed by the PLC before a connection will be established.

Exact Match – All of the parameters must match or the connection will be rejected.

- Vendor

- Product Type
- Catalog Number
- Major Revision
- Minor Revision

Compatible Module – The following criteria must be met, or else the inserted module will reject the connection:

- The Module Types must match
- Catalog Number must match
- Major Revision must match
- The Minor Revision of the module must be equal to or greater than the one specified in the software.

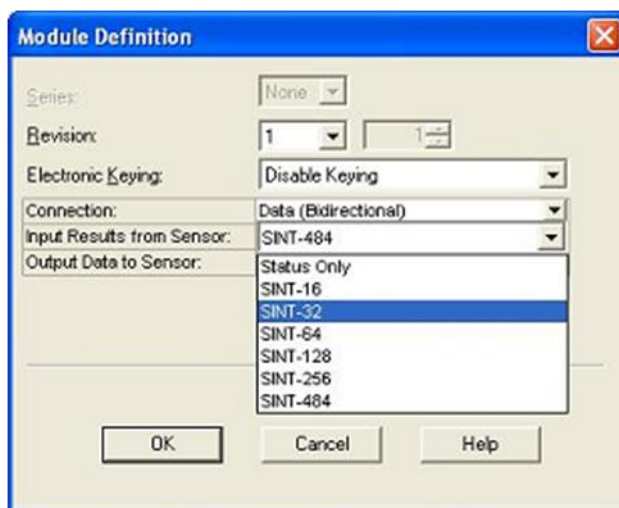
Disable Keying – The controller will not employ keying at all.

Connection: Defines the type of data flow.

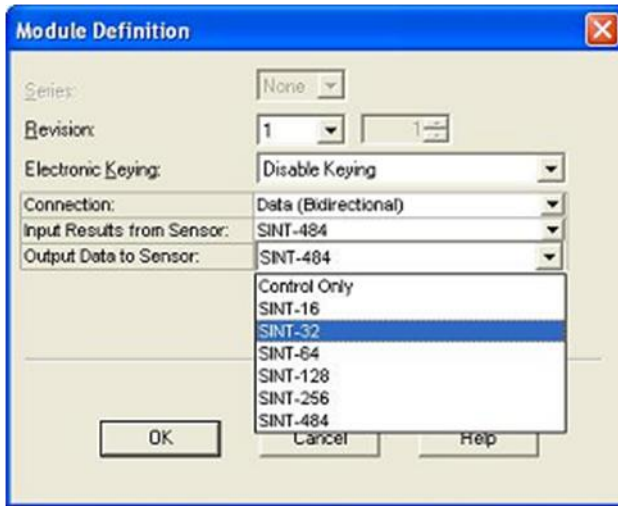
Data (Bidirectional) – The connection will send data (to the DataMan) and receive data (from the DataMan).

Input (Results only) – The connection will only receive data (from the DataMan). Generally used in situations where more than one PLC needs to receive data from the same DataMan device.

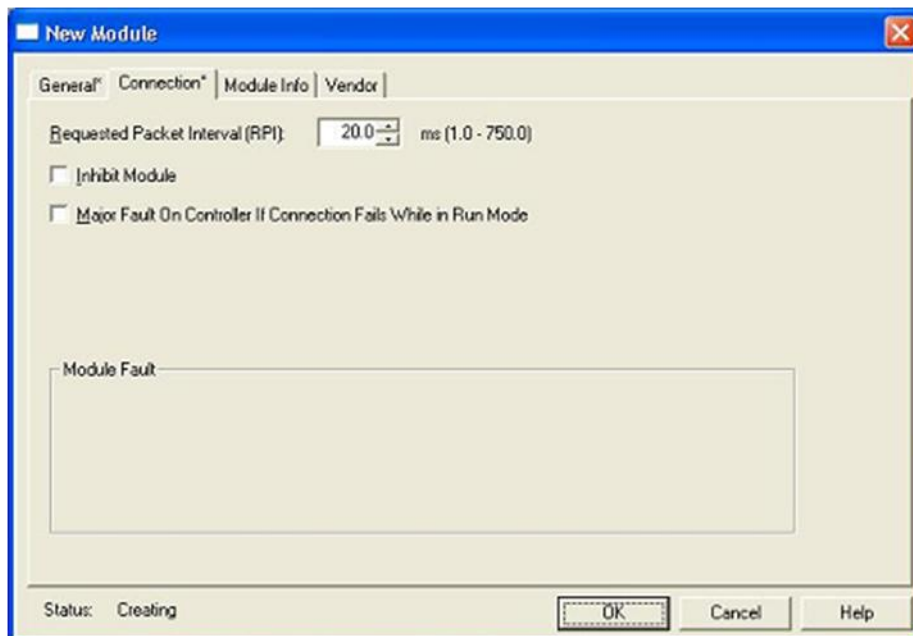
Input Results from Sensor: Defines the amount of data received on the connection (from the DataMan). The minimum amount is the Status data only. The connection can be configured to also receive read result data. The amount of result data received is defined in fixed increments (16 bytes, 32 bytes, 64 bytes etc). The size should be selected to return no more than the largest code size to be read by the application. Setting the size larger wastes network bandwidth and diminishes performance.



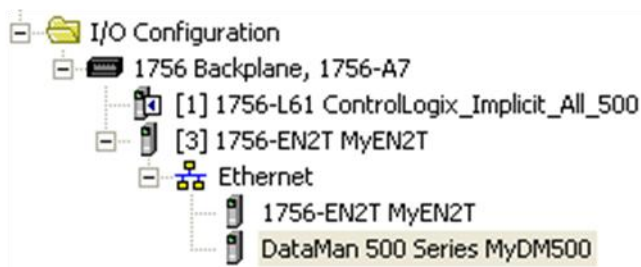
Output Data to Sensor: Defines the amount of data transmitted on the connection (to the DataMan). The minimum amount is the Control data only. The connection can be configured to also send user data. The amount of user data sent is defined in fixed increments (16 bytes, 32 bytes, 64 bytes etc).



5. The final step is configuring the connection rate. The rate at which data is transmitted/received is defined as the Requested Packet Interval (RPI). The RPI defines how frequently the data is transmitted/received over the connection. To optimize network performance this rate should be set no lower than absolutely required by a given application. In general it should be set no lower than $\frac{1}{2}$ the expected maximum read rate of the user application. Setting it lower wastes bandwidth and does not improve processing performance.
6. Select the "Connection" tab of the "New Module" dialog to set the rate.



7. After adding the module to ControlLogix, the I/O tree should appear as follows:



- When the DataMan module is added to the I/O tree RSLogix 5000 creates tags that map to the DataMan reader Input and Output Data (i.e. the Input & Output Assembly Objects in the DataMan Reader). These tags can be found under the "Controller Tags" node of the project tree.

NOTE

The base name of these tags is the name you gave to the DataMan Module that you added to the I/O Configuration in the earlier steps.

Name	Value	Style	Data Type
MyDM500:I	{...}		CC:DataMan
MyDM500:I.Status	{...}		CC:DataMan
MyDM500:I.Status.TriggerReady	0	Decimal	BOOL
MyDM500:I.Status.TriggerAck	0	Decimal	BOOL
MyDM500:I.Status.Acquiring	0	Decimal	BOOL
MyDM500:I.Status.MissedAcq	0	Decimal	BOOL
MyDM500:I.Status.Decoding	0	Decimal	BOOL
MyDM500:I.Status.DecodeCompleted	0	Decimal	BOOL
MyDM500:I.Status.ResultsBufferOverrun	0	Decimal	BOOL
MyDM500:I.Status.ResultsAvailable	0	Decimal	BOOL
MyDM500:I.Status.GeneralFault	0	Decimal	BOOL
MyDM500:I.Status.TrainCodeAck	0	Decimal	BOOL
MyDM500:I.Status.TrainMatchStringAck	0	Decimal	BOOL
MyDM500:I.Status.TrainFocusAck	0	Decimal	BOOL
MyDM500:I.Status.TrainBrightnessAck	0	Decimal	BOOL
MyDM500:I.Status.UntrainAck	0	Decimal	BOOL
MyDM500:I.Status.ExecuteDmccAck	0	Decimal	BOOL
MyDM500:I.Status.SetMatchStringAck	0	Decimal	BOOL
MyDM500:I.Status.TriggerID	0	Decimal	INT
MyDM500:I.Status.ResultID	0	Decimal	INT
MyDM500:I.Status.ResultCode	0	Decimal	INT
MyDM500:I.Status.ResultExtended	0	Decimal	INT
MyDM500:I.Status.ResultLength	0	Decimal	INT
MyDM500:I.ResultData	{...}	ASCII	SINT[484]
MyDM500:O	{...}		CC:DataMan
MyDM500:O.Control	{...}		CC:DataMan
MyDM500:O.Control.TriggerEnable	0	Decimal	BOOL
MyDM500:O.Control.Trigger	0	Decimal	BOOL
MyDM500:O.Control.ResultsBufferEnable	0	Decimal	BOOL
MyDM500:O.Control.ResultsAck	0	Decimal	BOOL
MyDM500:O.Control.TrainCode	0	Decimal	BOOL
MyDM500:O.Control.TrainMatchString	0	Decimal	BOOL
MyDM500:O.Control.TrainFocus	0	Decimal	BOOL
MyDM500:O.Control.TrainBrightness	0	Decimal	BOOL
MyDM500:O.Control.Untrain	0	Decimal	BOOL

The tags are organized in two groups: Status and Control. The Status group represents all the data being received (from the DataMan). The Control group represents all the data being sent (to the DataMan).

These tags are the symbolic representation of the DataMan Assembly Object contents. The PLC ladder is written to access these tag values. By monitoring or changing these tag values the PLC ladder is actually monitoring and changing the DataMan Assembly Object contents.

NOTE

There is a time delay between the DataMan and these PLC tag values (base on the configured RPI). All PLC ladder must be written to take that time delay into account.

Accessing Implicit Messaging Connection Data

The section above details establishing an implicit message connection between a ControlLogix and a DataMan ID Reader. This example assumes that the DataMan Add-On-Profile is being utilized. One aspect of the Add-On-Profile is that it will automatically generate ControlLogix tags representing the connection data.

The generated tags are divided into two groups: Status & Control. The Status group represents all the data being received (from the DataMan). The Control group represents all the data being sent (to the DataMan).

A description of the Status tag group follows. This is the data received by the ControlLogix from the DataMan reader.

- MyDM500:I	{...}		CC:DataMan
- MyDM500:I.Status	{...}		CC:DataMan
MyDM500:I.Status.TriggerReady	0	Decimal	BOOL
MyDM500:I.Status.TriggerAck	0	Decimal	BOOL
MyDM500:I.Status.Acquiring	0	Decimal	BOOL
MyDM500:I.Status.MissedAcq	0	Decimal	BOOL
MyDM500:I.Status.Decoding	0	Decimal	BOOL
MyDM500:I.Status.DecodeCompleted	0	Decimal	BOOL
MyDM500:I.Status.ResultsBufferOvrrun	0	Decimal	BOOL
MyDM500:I.Status.ResultsAvailable	0	Decimal	BOOL
MyDM500:I.Status.GeneralFault	0	Decimal	BOOL
MyDM500:I.Status.TrainCodeAck	0	Decimal	BOOL
MyDM500:I.Status.TrainMatchStringAck	0	Decimal	BOOL
MyDM500:I.Status.TrainFocusAck	0	Decimal	BOOL
MyDM500:I.Status.TrainBrightnessAck	0	Decimal	BOOL
MyDM500:I.Status.UntrainAck	0	Decimal	BOOL
MyDM500:I.Status.ExecuteDmccAck	0	Decimal	BOOL
MyDM500:I.Status.SetMatchStringAck	0	Decimal	BOOL
+ MyDM500:I.Status.TriggerID	0	Decimal	INT
+ MyDM500:I.Status.ResultID	0	Decimal	INT
+ MyDM500:I.Status.ResultCode	0	Decimal	INT
+ MyDM500:I.Status.ResultExtended	0	Decimal	INT
+ MyDM500:I.Status.ResultLength	0	Decimal	INT
+ MyDM500:I.ResultData	{...}	ASCII	SINT[484]

- **TriggerReady:** Indicates when the DataMan reader can accept a new trigger. This tag is True when the Control tag "TriggerEnable" has been set and the sensor is not currently acquiring an image.

- **TriggerAck:** Indicates when the DataMan reader has been triggered (i.e. the Control tag "Trigger" has been set to True). This tag will stay set until the Trigger tag is cleared.
- **Acquiring:** Indicates when the DataMan reader is currently acquiring an image; either by setting the Trigger bit or by an external trigger.
- **MissedAcq:** Indicates when the DataMan reader misses an acquisition trigger; cleared when the next successful acquisition occurs.
- **Decoding:** Indicates when the DataMan reader is decoding an acquired image.
- **DecodeCompleted:** Tag value is toggled (1→0 or 0→1) on the completion of a decode.
- **ResultsBufferOverrun:** Indicates when the DataMan reader has discarded a set of decode results because the results queue is full. Cleared when the next set of results are successfully queued.
- **ResultsAvailable:** Indicates when a set of decode results are available (i.e. the ResultID, ResultCode, ResultLength and ResultsData tags contain valid data).
- **GeneralFault:** Indicates when a fault has occurred (i.e. Soft event "SetMatchString" or "ExecuteDMCC" error has occurred).
- **TrainCodeAck:** Indicates that the soft event "TrainCode" has completed.
- **TrainMatchStringAck:** Indicates that the soft event "TrainMatchString" has completed.
- **TrainFocusAck:** Indicates that the soft event "TrainFocus" has completed.
- **TrainBrightnessAck:** Indicates that the soft event "TrainBrightness" has completed.
- **UnTrainAck:** Indicates that the soft event "UnTrain" has completed.
- **ExecuteDmccAck:** Indicates that the soft event "ExecuteDMCC" has completed.
- **SetMatchStringAck:** Indicates that the soft event "SetMatchString" has completed.
- **TriggerID:** Value of the next trigger to be issued. Used to match triggers issued with corresponding result data received later.
- **ResultID:** The value of TriggerID when the trigger that generated these results was issued. Used to match TriggerID's with result data.
- **ResultCode:** Indicates success/failure of this set of results.
 - Bit 0** ,1=read 0=no read
 - Bit 1** ,1=validated 0=not validated (or validation not in use)
 - Bit 2** ,1=verified 0=not verified (or verification not in use)
 - Bit 3** ,1=acquisition trigger overrun
 - Bit 4** ,1=acquisition buffer overflow (not the same as result buffer overflow).
 - Bits 5-15** ,reserved (future use)
- **ResultExtended:** Currently unused.
- **ResultLength:** Number of bytes of result data contained in the ResultData tag.
- **ResultData:** Decode result data.

A description of the Control tag group follows. This is the data sent from the ControlLogix to the DataMan reader.

Name	△	Value ←	Style	Data Type
[-] MyDM200:0		{ ... }		CC:DataMan...
[-] MyDM200:0.Control		{ ... }		CC:DataMan...
MyDM200:0.Control.TriggerEnable		0	Decimal	BOOL
MyDM200:0.Control.Trigger		0	Decimal	BOOL
MyDM200:0.Control.ResultsBufferEnable		0	Decimal	BOOL
MyDM200:0.Control.ResultsAck		0	Decimal	BOOL
MyDM200:0.Control.TrainCode		0	Decimal	BOOL
MyDM200:0.Control.TrainMatchString		0	Decimal	BOOL
MyDM200:0.Control.TrainFocus		0	Decimal	BOOL
MyDM200:0.Control.TrainBrightness		0	Decimal	BOOL
MyDM200:0.Control.Untrain		0	Decimal	BOOL
MyDM200:0.Control.ExecuteDMCC		0	Decimal	BOOL
MyDM200:0.Control.SetMatchString		0	Decimal	BOOL
[+] MyDM200:0.Control.UserDataOption		0	Decimal	INT
[+] MyDM200:0.Control.UserDataLength		0	Decimal	INT
[+] MyDM200:0.UserData		{ ... }	ASCII	SINT[484]

- **TriggerEnable:** Setting this tag enables EtherNet/IP triggering. Clearing this field disables the EtherNet/IP triggering.
- **Trigger:** Setting this tag triggers an acquisition when the following conditions are met:
 - The TriggerEnable tag is set.
 - No acquisition/decode is currently in progress.
 - The device is ready to trigger.
- **ResultsBufferEnable:** When set, the decode results will be queued. Results are pulled from the queue (made available) each time the current results are acknowledged. until acknowledged by the PLC. The Decode ID, Decode Result and Decode ResultsData fields are held constant until the DecodeResultsAck field has acknowledged them and been set. The DataMan reader will respond to the acknowledgement by clearing the ResultsValid bit. Once the DecodeResultsAck field is cleared the next set of decode results will be posted.
- **ResultsAck:** The ResultsAck tag is used to acknowledge that the PLC has read the latest results. When ResultsAck is set, the ResultsAvailable tag will be cleared. If results buffering is enabled the next set of results will be made available when the ResultsAck tag is again cleared.
- **TrainCode:** Changing this tag from 0 to 1 will cause the train code operation to be invoked.
- **TrainMatchString:** Changing this tag from 0 to 1 will cause the train match string operation to be invoked.
- **TrainFocus:** Changing this tag from 0 to 1 will cause the train focus operation to be invoked.

- **TrainBrightness:** Changing this tag from 0 to 1 will cause the train brightness operation to be invoked.
- **Untrain:** Changing this tag from 0 to 1 will cause the un-train operation to be invoked.
- **ExecuteDMCC:** Changing this tag from 0 to 1 will cause the DMCC operation to be invoked. A valid DMCC command string must be written to UserData prior to invoking this soft event.
- **SetMatchString:** Changing this tag from 0 to 1 will cause the set match string operation to be invoked. The match string data must be written to UserData prior to invoking this soft event.
- **UserDataOption:** Currently unused.
- **UserDataLength:** Number of bytes of user data contained in the UserData tag.
- **UserData:** This data is sent to the DataMan reader to support acquisition and/or decode.

Verifying Implicit Messaging Connection Operation

The DataMan reader has been added as an I/O device in a ControlLogix project. After this project is downloaded to the controller, the I/O connection will be established. Once a successful connection has been established, cyclic data transfers will be initiated, at the requested RPI.

To verify a proper I/O connection, follow these steps:

1. Download the project created above to the ControlLogix controller.
2. Upon the completion of the download, the project I/O indicator should be "I/O OK". This signifies that the I/O connection has been completed successfully.



To verify the correct, 2-way transfer of I/O data, in RSLogix, go to the controller tags and change the state of the TriggerEnable bit from 0 to 1:

MyDM200:0	{...}		CC:DataMan...
MyDM200:0.Control	{...}		CC:DataMan...
MyDM200:0.Control.TriggerEnable	1	Decimal	BOOL
MyDM200:0.Control.Trigger	0	Decimal	BOOL
MyDM200:0.Control.ResultsBufferEnable	0	Decimal	BOOL
MyDM200:0.Control.ResultsAck	0	Decimal	BOOL
MyDM200:0.Control.TrainCode	0	Decimal	BOOL
MyDM200:0.Control.TrainMatchString	0	Decimal	BOOL
MyDM200:0.Control.TrainFocus	0	Decimal	BOOL
MyDM200:0.Control.TrainBrightness	0	Decimal	BOOL
MyDM200:0.Control.Untrain	0	Decimal	BOOL
MyDM200:0.Control.ExecuteDMCC	0	Decimal	BOOL
MyDM200:0.Control.SetMatchString	0	Decimal	BOOL
MyDM200:0.Control.UserDataOption	0	Decimal	INT
MyDM200:0.Control.UserDataLength	0	Decimal	INT

- The TriggerReady tag changes to 1.
- Triggering is now enabled. Whenever the Trigger tag is changed from 0 to 1, the DataMan reader will acquire an image. Note that the current TriggerID value is 1. The results of the next trigger to be issued should come back with a corresponding ResultID of 1.
- After the acquisition/decode has completed, the DecodeCompleted tag will toggle and the ResultsAvailable tag will go to 1. In the example shown here a successful read has occurred (ResultCode bit 0 = 1) and the read has returned 16 bytes of data (ResultLength=16). The data can be found in the ResultData tag.

MyDM200:1	{...}		CC:DataMan...
MyDM200:1.Status	{...}		CC:DataMan...
MyDM200:1.Status.TriggerReady	1	Decimal	BOOL
MyDM200:1.Status.TriggerAck	0	Decimal	BOOL
MyDM200:1.Status.Acquiring	0	Decimal	BOOL
MyDM200:1.Status.MissedAcq	0	Decimal	BOOL

[-] MyDM500:I	{...}		CC:DataMan_
[-] MyDM500:I.Status	{...}		CC:DataMan_
MyDM500:I.Status.TriggerReady	1	Decimal	BOOL
MyDM500:I.Status.TriggerAck	0	Decimal	BOOL
MyDM500:I.Status.Acquiring	0	Decimal	BOOL
MyDM500:I.Status.MissedAcq	0	Decimal	BOOL
MyDM500:I.Status.Decoding	0	Decimal	BOOL
MyDM500:I.Status.DecodeCompleted	1	Decimal	BOOL
MyDM500:I.Status.ResultsBufferOverrun	0	Decimal	BOOL
MyDM500:I.Status.ResultsAvailable	1	Decimal	BOOL
MyDM500:I.Status.GeneralFault	0	Decimal	BOOL
MyDM500:I.Status.TrainCodeAck	0	Decimal	BOOL
MyDM500:I.Status.TrainMatchStringAck	0	Decimal	BOOL
MyDM500:I.Status.TrainFocusAck	0	Decimal	BOOL
MyDM500:I.Status.TrainBrightnessAck	0	Decimal	BOOL
MyDM500:I.Status.UntrainAck	0	Decimal	BOOL
MyDM500:I.Status.ExecuteDmccAck	0	Decimal	BOOL
MyDM500:I.Status.SetMatchStringAck	0	Decimal	BOOL
+ MyDM500:I.Status.TriggerID	297	Decimal	INT
+ MyDM500:I.Status.ResultID	296	Decimal	INT
+ MyDM500:I.Status.ResultCode	1	Decimal	INT
+ MyDM500:I.Status.ResultExtended	0	Decimal	INT
+ MyDM500:I.Status.ResultLength	5	Decimal	INT
+ MyDM500:I.ResultData	{...}	ASCII	SINT[484]

Explicit Messaging

Unlike implicit messaging, explicit messages are sent to a specific device and that device always responds with a reply to that message. As a result, explicit messages are better suited for operations that occur infrequently. Explicit messages can be used to read and write the attributes (data) of the ID Reader Object. They may also be used for acquiring images, sending DMCC commands and retrieving result data.

Issuing DMCC Commands

One of the more common explicit messages sent to a DataMan ID Reader is an instruction to execute a DMCC command. Explicit messages are sent from ControlLogix to a DataMan using MSG instructions. There are two different paths for invoking DMCC messages with explicit messaging; via the PCCC Object or via the ID Reader Object "SendDMCC" service. In this example we show the SendDMCC service.

The CIP STRING2 format is required for transmission across EtherNet/IP (that is, 16-bit length value followed by actual string characters, no null terminator). But Logix stores strings in a slightly different format (i.e. 32-bit length value followed by actual string characters, no null terminator). Therefore some of the sample ladder involves converting to/from the two different string formats.

NOTE

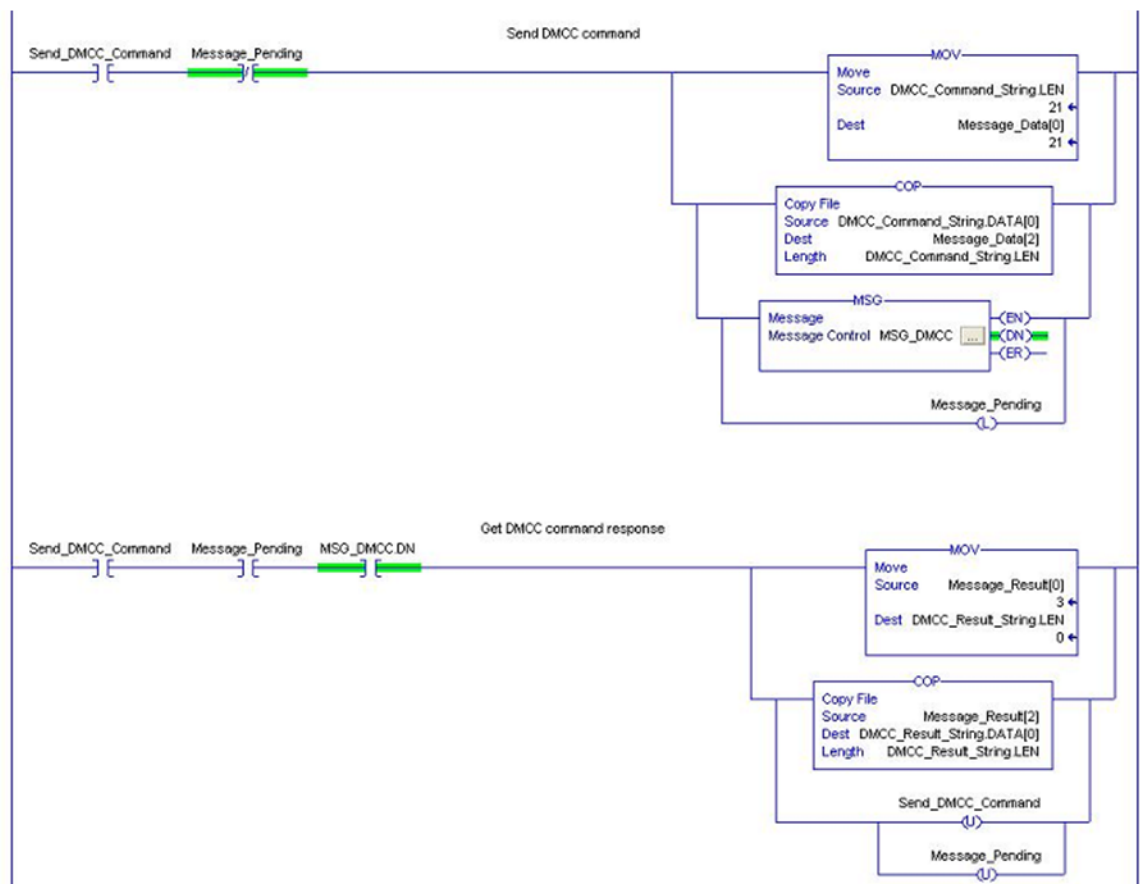
This example is intended as a demonstration of DataMan explicit messaging behavior. This same operation could be written in much more efficient ladder but would be less useful as a learning tool.

1. Add the following tags to the ControlLogix **Controller Tags** dialog:

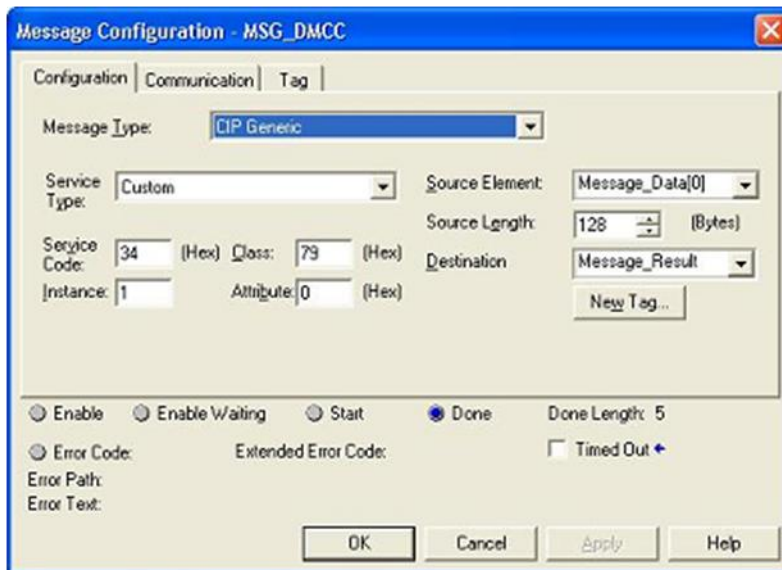
Name	Data Type	Style
Send_DMCC_Command	BOOL	Decimal
+ DMCC_Command_String	STRING	
+ DMCC_Result_String	STRING	
+ Message_Data	SINT[128]	Decimal
+ Message_Result	SINT[128]	Decimal
Message_Pending	BOOL	Decimal
+ MSG_DMCC	MESSAGE	

- Send_DMCC_Command: Boolean flag used to initiate the command.
- DMCC_Command_String: String containing the DMCC command to execute.
- DMCC_Result_String: String receiving the DMCC command results
- Message_Data: Temp buffer holding the data to send via the MSG instruction.
- Message_Result: Temp buffer holding the data received via the MSG instruction.
- Message_Pending: Boolean flag used to indicate that a message is in process.
- MSG_DMCC: Data structure required by the Logix MSG instruction.

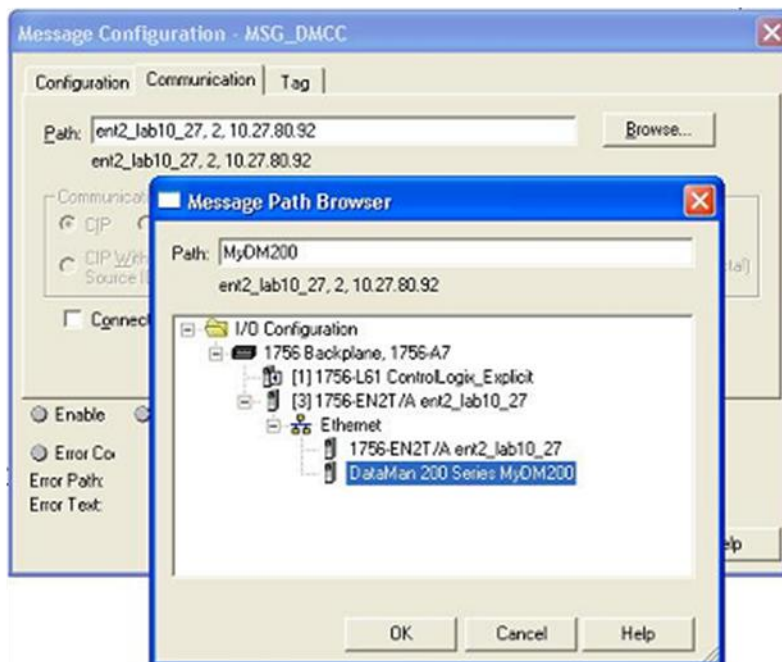
2. Add the following two rungs to the MainRoutine of your ControlLogix project:



3. Edit the MSG instruction. Configure it for "CIP Generic", service 0x34 "SendDMCC", class 0x79 "ID Reader Object" and instance 1. Set the source to "Message_Data" and the destination to "Message_Result".



4. On the MSG instruction "Communication" tab, browse for and select the DataMan which you added to the project I/O Configuration tree. This tells Logix where to send the explicit message.



5. Download to the ControlLogix and place in "Run Mode".
6. To operate:

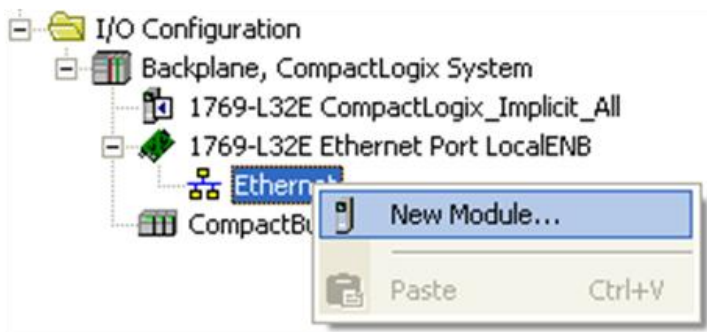
- Place a DMCC command in the "DMCC_Command_String" tag. For example "|>GET TRIGGER.TYPE\$r\$!". Note the \$r\$! at the end of the string. This is how Logix represents a CRLF.
- Toggle the "Send_DMCC_Command" tag to 1.
- When the "Send_DMCC_Command" tag goes back to 0 execution is complete. The DMCC command results will be found in "DMCC_Result_String".

Rockwell CompactLogix Examples

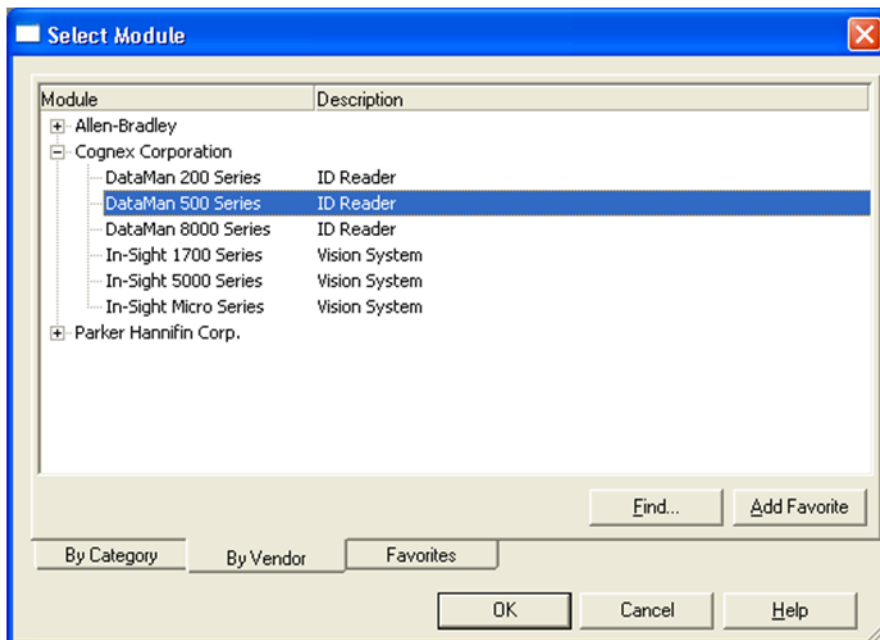
CompactLogix differs very little from ControlLogix in terms of programming. The ControlLogix examples apply to equally to CompactLogix systems. There is only a slight difference in adding the DataMan device in the project I/O tree.

The I/O Configuration tree in a CompactLogix project looks a bit different from a ControlLogix project. Regarding the Ethernet connection, the difference is that the Ethernet logic module is actually embedded in the CompactLogix processor module. It is displayed in the I/O Configuration tree as if it were a separate module on the backplane. This module is also configured exactly like a ControlLogix Ethernet module.

The DataMan module is added in the same way for CompactLogix as for ControlLogix. Right-click on the Ethernet node in the I/O Configuration tree and select "New Module".



From the "Select Module" dialog, choose your model of DataMan ID Reader from the list.



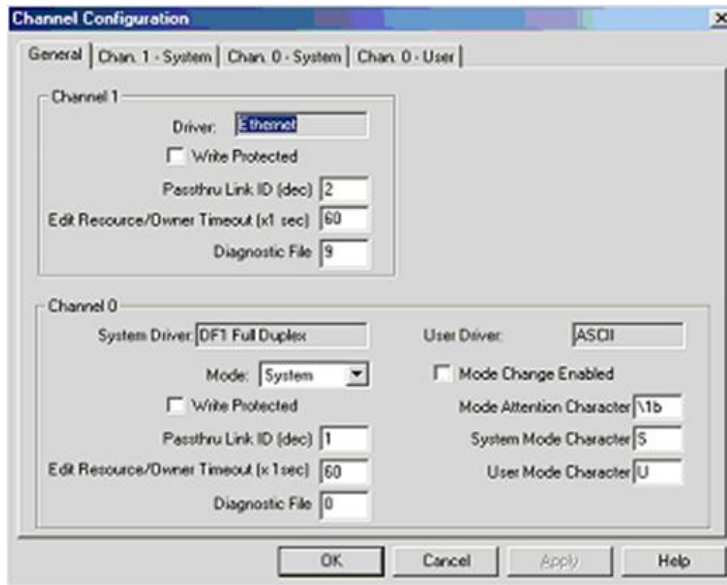
After the selection is made, the configuration dialog for the DataMan ID Reader system will be displayed. From this point on configuration and programming are done exactly as shown in the ControlLogix section above.

Rockwell SLC 5/05 Examples

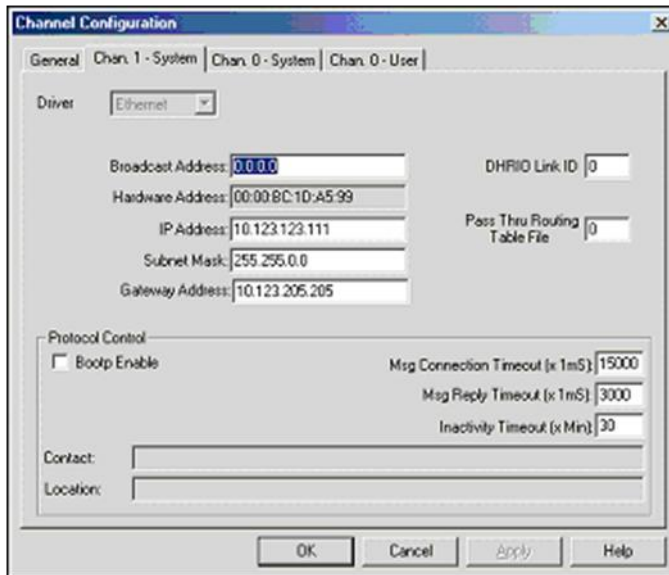
This section outlines a PCCC (PC³) Communications configuration between a DataMan reader and the PLC. This example uses the Allen-Bradley SLC5/05 and Rockwell 500 software.

Setting up the PLC for Ethernet Communication

1. From within the RSLogix 500 software program, open the .RSS file, then open the Channel Configuration dialog (*Project Folder > Controller Folder > Channel Configuration*)



2. The Allen-Bradley SLC has 2 channels available for configuration: Channel 1 (Ethernet); and Channel 0 (DF1 Full Duplex - serial). Click on the **Chan. 1 - System** tab.
3. Configure Channel 1 (Ethernet) as necessary. Consult with a network administrator for proper settings.



4. Configure the Timeouts as required.

Message Instruction (MSG)

Message instructions may now be constructed within the application. Refer to the RSLogix 500 documentation for expanded instructions for developing messages.

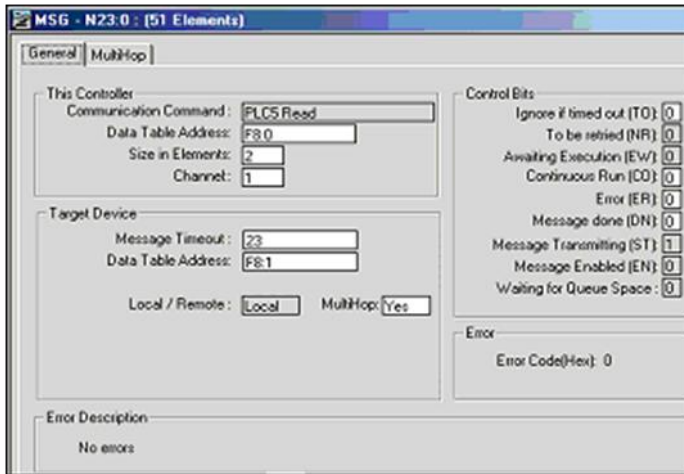
The following setup parameters can be configured within a Message (MSG) Instruction.



- **Type:** *Peer-To-Peer*. This cannot be modified.
- **Read/Write:** Select the function you want to perform on a DataMan reader. *Read* retrieves data from the DataMan; *Write* sends data to the DataMan.
- **Target Device:** Choose *PLC5* to talk to a DataMan reader. This tells the SLC which communication protocol to use. The DataMan reader acts much like a ControlLogix controller (see *Rockwell document 13862*).
- **Local/Remote:** Choose *Local* to indicate that the DataMan reader is on the same network as the SLC; *Remote* tells the SLC that you will be communicating to a DataMan on another network. For remote communication, you must direct the message through another device acting as a gateway to that secondary network. Typically, this could be an Allen-Bradley ControlLogix controller. (Refer to *Rockwell documentation on how to address devices on other networks through a gateway.*)
- **Control Block:** This is a temporary integer file that the MSG instruction uses to store data (i.e., IP address, message type, etc.). This is typically not the user data to be sent.
- **Control Block Length:** This is automatically computed by the MSG instruction.

- **Setup Screen:** Selecting *Setup Screen* will open the *Message Instruction Setup* dialog.

The following setup parameters can be configured within an *MSG Instruction Setup* screen.



This Controller section:

- **Communication Command:** Should be the same command (READ/WRITE) that was chosen on the first screen (as seen in MSG Instruction screen).
- **Data Table Address:** This is the location of the data file on the SLC where data will be written to (READ) or sent from (WRITE) (as seen in MSG Instruction screen). In this instance, 'F8:0', 'F' indicates the float file, '8' indicates the file number 8, and '0' indicates the offset into that file (in this case, start at the 0th element). The figure below shows an example of the Float Table accessed from the RSLogix 500 main screen.

Offset	0	1	2	3	4
F8:0	3.14	78.87	96.69	0	0
F8:5	0	0	0	0	0

- **Size in Elements:** This is the number of elements (or individual data) to send. In this example, two elements are being sent (3.14 and 78.87).
- **Channel:** Depends on the configuration of the SLC. In the SLC, Channel 1 is the Ethernet port.

Target Device section:

- **Message Timeout:** Choose an appropriate length of time in which the DataMan reader will be able to respond. If the DataMan does not respond within this length of time, the MSG instruction will error out. This parameter cannot be changed from this screen. Message Timeout is determined by the parameters entered in the Channel 1 setup dialog.

Data Table Address: This is the location on the DataMan reader where data will be read or written to. In this instance, 'N7:1', 'N' indicates that the data is of type integer (16-bit); '7' is ignored by the DataMan (data is always being written to the Output Assembly, and read from the Input Assembly); and the '1' is the element offset from the start of the target buffer. For example: If the message were a READ, 'N7:2' would instruct to read the 3rd integer (the ':2' indicates the 3rd element, due to the SLC's 0-based index) from the Input Assembly (because a READ gets data from the DataMan's Input Assembly). If the message were a WRITE, 'N7:12' would indicate to write a (16-bit) integer value to the 13 integer location of the Output Assembly.

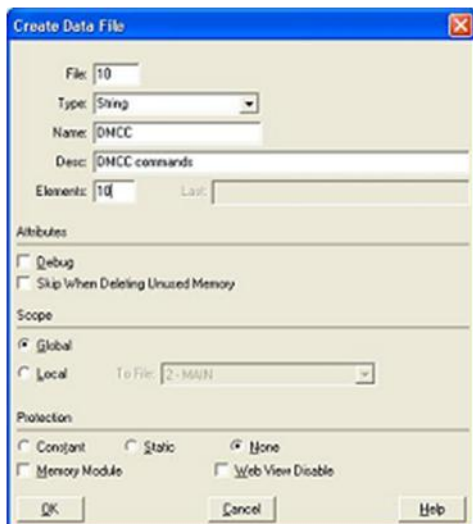
NOTE

The ST10:0 destination address is a special case used for sending DMCC commands to a DataMan reader. Any string sent to ST10:0 will be interpreted as a DMCC command.

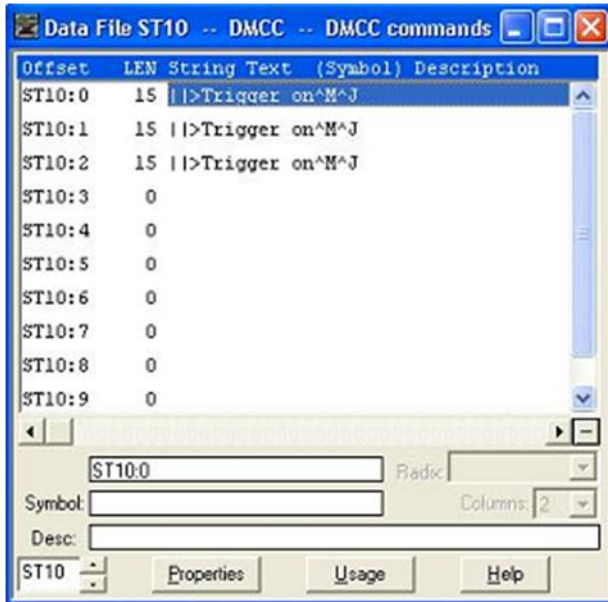
- **Local/Remote:** Set to *Local* or *Remote*, depending on the application.
- **MultiHop:** This setting is dependent on the information previously entered. For successful In-Sight communication, this should YES at this time.

Sending DMCC Commands from an SLC 5/05

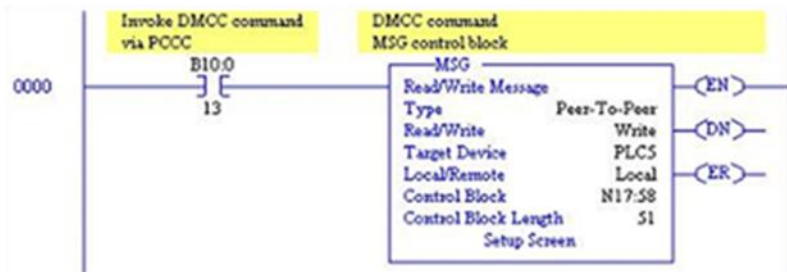
1. Configure the SLC5/05 as necessary.
2. Create a String Table that will hold your DMCC commands.



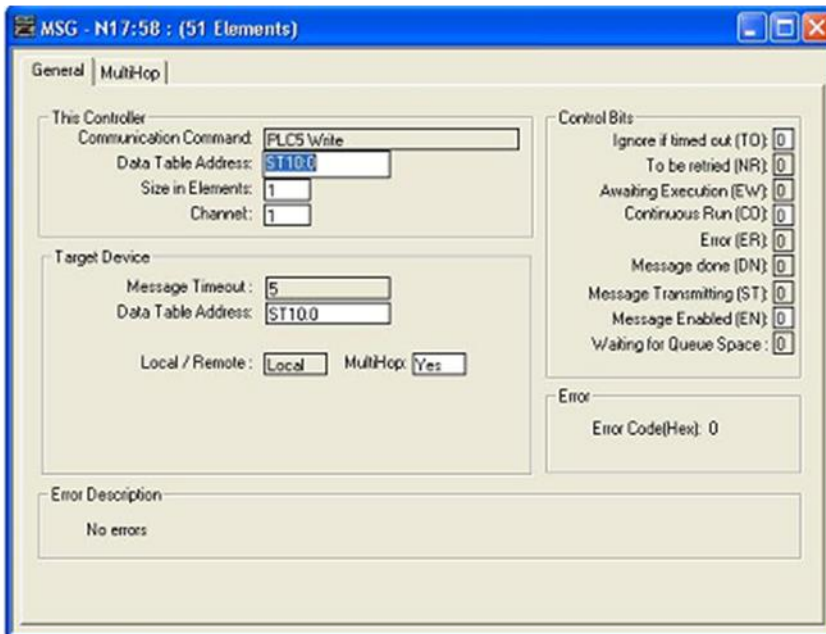
3. Add the required DMCC command strings to the Data File.



4. Add a new Message (MSG) instruction to your ladder logic and configure it as shown in the following example:



5. Enter the **MSG Setup Screen** and configure it as follows:



This Controller	Parameter	Description
Data Table Address	ST10:0	First element from the String Table (ST) created above
Size in Elements	1	Always set to 1. PCCC MSG only allows 1 string (therefore 1 command) to be sent at a time.
Channel	1	Set this to the Ethernet channel of your controller.

Target Device	Parameter	Description
Message Timeout	(From channel configuration dialog)	
Data Table Address	ST10:0	This is the destination address. For DMCC commands, this will always be ST10:0

6. Click the **MultiHop** tab and configure it as required (i.e. set IP address of DataMan).
7. When everything is configured, close the MSG window.
8. Save your ladder logic, download it to the controller, then go online and set the controller in RUN mode.
9. Trigger the message to send it to the DataMan reader.

Message Instruction Results



The Enable (EN) bit of the message instruction will be set to 1 when the input to the instruction is set high. The Done (DN) bit will be set to 1 when DataMan has replied that the DMCC command was received and executed with success. If the Error bit (ER) is enabled (set to 1), there has been a problem with the message instruction. If an error occurs, click the Setup Screen for the MSG instruction. The Error Code will be shown at the bottom of the window.

Using the Generic EtherNet/IP Profile

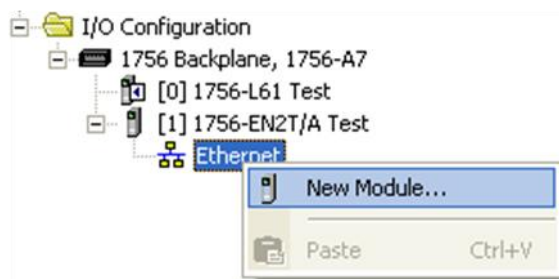
For devices without a specific Add-On-Profile Rockwell provides a Generic EtherNet/IP profile. This profile allows you to create implicit messaging connections but lacks the automatic tag generation feature of a specific product Add-On-Profile.

Establishing a Generic Implicit Messaging Connection

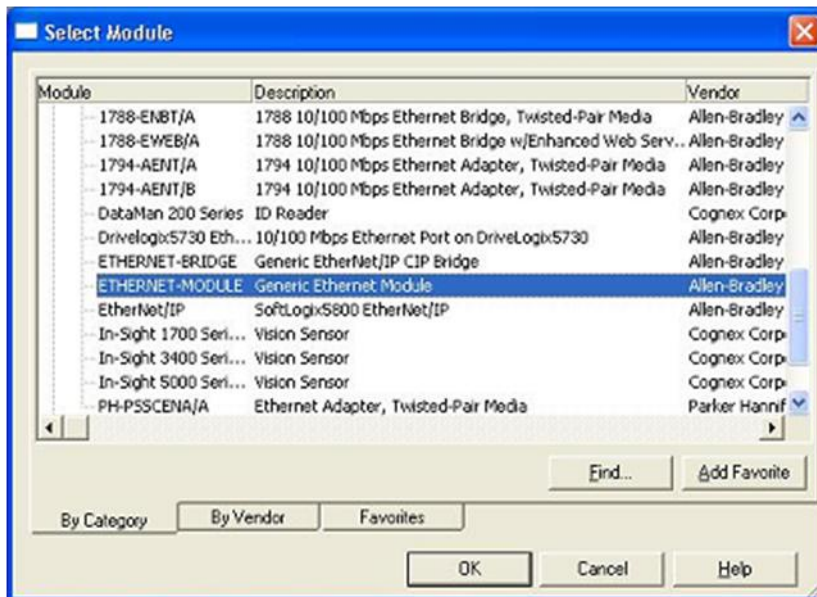
To setup an EtherNet/IP implicit messaging connection between a DataMan and a ControlLogix controller, the DataMan reader must first be added to the ControlLogix I/O Configuration tree. This can be accomplished with the Rockwell provided generic profile.

To establish a generic implicit messaging connection with a ControlLogix PLC:

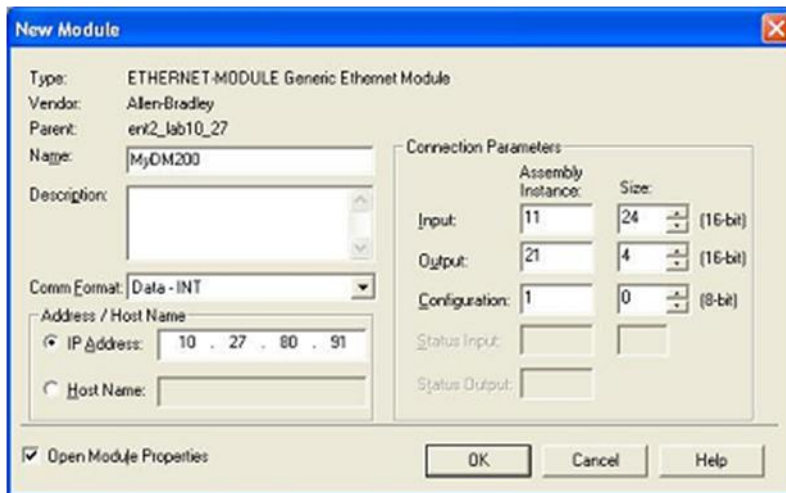
1. Open RSLogix5000 and load your project (or select "File->New..." to create a new one).
2. From the I/O Configuration node, select the *Ethernet* node under the project Ethernet Module, right-click on the icon and select New Module from the menu:



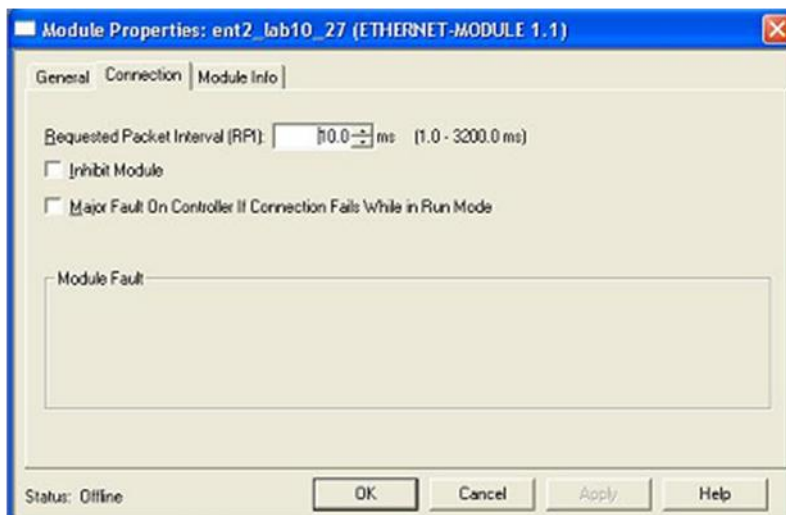
3. From the Select Module dialog, choose the Allen-Bradley Generic Ethernet Module.



4. After the selection is made, the configuration dialog for the Generic Ethernet Module will be displayed. Configure the following:
- Give the module a name.
 - Enter your DataMan's IP address.
 - Set the Comm Format to "Data - INT". This tells the module to treat the data as an array of 16-bit integers.
 - Input Assembly: Set instance 11. Set the size to the amount of Input Assembly data you want the PLC to receive. Basic "Status" data requires 8 integers. The amount beyond that will be the actual decode result data. In the example below the size is set to 24 (8 for status + 16 for result data). This connection will receive the status info plus 32 bytes of result data.
 - Output Assembly: Set instance 21. Set the size to 4 integers. This size is sufficient to send all required "Control" data to the DataMan.
 - Configuration Assembly: Set instance 1. Set size to zero (no used).



- The final step is configuring the connection rate. The rate at which data is transmitted/received is defined as the Requested Packet Interval (RPI). The RPI defines how frequently the data is transmitted/received over the connection. To optimize network performance this rate should be set no lower than absolutely required by a given application. In no case should it be set to lower than $\frac{1}{2}$ the median scan rate of the PLC ladder program. Setting it lower wastes bandwidth and does not improve processing performance.



- After adding the generic module to ControlLogix, the I/O tree should appear as follows.



- When the Generic Module is added to the I/O tree RSLogix 5000 creates tags that map to the DataMan reader Input and Output Data (i.e. the Input & Output Assembly Objects in the DataMan Reader). These tags can be found under the "Controller Tags" node of the project tree.

NOTE

The base name of these tags is the name you gave to the Generic Module that you added to the I/O Configuration earlier.

Name	Value	Style	Data Type
+ MyDM200:C	{...}		AB:ETHERNET_MODULE:C:0
- MyDM200:I	{...}		AB:ETHERNET_MODULE_INT_48Bytes:I:0
+ MyDM200:I.Data	{...}	Decimal	INT[24]
- MyDM200:O	{...}		AB:ETHERNET_MODULE_INT_88Bytes:O:0
+ MyDM200:O.Data	{...}	Decimal	INT[4]

The tags are organized in three groups: Config "MyDM200:C", Input "MyDM200:I", and Output "MyDM200:O". You can ignore the Config tags (no used). The Input tags represent all the data being received (from the DataMan). The Output tags represent all the data being sent (to the DataMan).

These tags are the data table representation of the DataMan Assembly Object contents. The PLC ladder is written to access these tag values. By monitoring or changing these tag values the PLC ladder is actually monitoring and changing the DataMan Assembly Object contents.

NOTE

There is a time delay between the DataMan and these PLC tag values (based on the configured RPI). All PLC ladder must be written to take that time delay into account.

Accessing Generic Implicit Messaging Connection Data

The section above details establishing an implicit message connection between a ControlLogix and a DataMan ID Reader using the Generic Module profile. Unlike the DataMan Add-On-Profile the Generic profile does not automatically generate named tags representing the individual data items within an Assembly Object. Instead it simply generates an array of data according to the size of the connection you defined.

To access individual data items within an Assembly Object you must manually select the correct tag offset and data subtype (if necessary) within the tag array that the Generic profile provided. This can be awkward and error prone since it requires you to manually reference the vendor documentation which defines the Assembly Objects.

NOTE

The start of the Input tags "MyDM200:I.Data[0]" maps directly to the start of the DataMan Input Assembly. Likewise, the start of the Output tags "MyDM200:O.Data[0]" maps directly to the start of the DataMan Output Assembly.

Examples

Input Assembly "TriggerReady": Bit 0 of word 0 of the Input Assembly. From the Input tag array for the DataMan select bit 0 of word 0.

MyDM200:I	{...}		AB:ETHERNET_MODULE_INT_48Bytes:I:0
MyDM200:I.Data	{...}	Decimal	INT[24]
MyDM200:I.Data[0]	0	Decimal	INT
MyDM200:I.Data[0].0	0	Decimal	BOOL
MyDM200:I.Data[0].1	0	Decimal	BOOL
MyDM200:I.Data[0].2	0	Decimal	BOOL
MyDM200:I.Data[0].3	0	Decimal	BOOL
MyDM200:I.Data[0].4	0	Decimal	BOOL
MyDM200:I.Data[0].5	0	Decimal	BOOL
MyDM200:I.Data[0].6	0	Decimal	BOOL
MyDM200:I.Data[0].7	0	Decimal	BOOL
MyDM200:I.Data[0].8	0	Decimal	BOOL
MyDM200:I.Data[0].9	0	Decimal	BOOL
MyDM200:I.Data[0].10	0	Decimal	BOOL
MyDM200:I.Data[0].11	0	Decimal	BOOL
MyDM200:I.Data[0].12	0	Decimal	BOOL
MyDM200:I.Data[0].13	0	Decimal	BOOL
MyDM200:I.Data[0].14	0	Decimal	BOOL
MyDM200:I.Data[0].15	0	Decimal	BOOL

Input Assembly "ResultLength": Word 7 of the Input Assembly. From the Input tag array for the DataMan select word 7.

MyDM200:I	{...}		AB:ETHERNET_MODULE_INT_48Bytes:I:0
MyDM200:I.Data	{...}	Decimal	INT[24]
MyDM200:I.Data[0]	0	Decimal	INT
MyDM200:I.Data[1]	0	Decimal	INT
MyDM200:I.Data[2]	0	Decimal	INT
MyDM200:I.Data[3]	0	Decimal	INT
MyDM200:I.Data[4]	0	Decimal	INT
MyDM200:I.Data[5]	0	Decimal	INT
MyDM200:I.Data[6]	0	Decimal	INT
MyDM200:I.Data[7]	0	Decimal	INT
MyDM200:I.Data[8]	0	Decimal	INT

Output Assembly "Trigger": Bit 1 of word 0 of the OutputAssembly. From the Output tag array for the DataMan select bit 1 of word 0.

MC Protocol

The MC Protocol uses standard Ethernet hardware and software to exchange I/O data, alarms, and diagnostics. It is Mitsubishi Electric's publicly available, standardized communication format for communicating with Q, iQ and L Series PLCs through Ethernet or serial connections. DataMan supports MC Protocol on Ethernet only.

By default the DataMan has MC Protocol disabled. The protocol can be enabled in the Setup Tool, via DMCC, or by scanning a parameter code.

DMCC

The following commands can be used to enable/disable MC Protocol. The commands can be issued via RS-232 or Telnet connection.

NOTE

Because you have to make changes to the Telnet client provided by Windows to communicate with DataMan, it is recommended you use third party clients such as PuTTY.

Enable:

```
||>SET MC-PROTOCOL.ENABLED ON
||>CONFIG.SAVE
||>REBOOT
```

Disable:

```
||>SET MC-PROTOCOL.ENABLED OFF
||>CONFIG.SAVE
||>REBOOT
```

Reader Configuration Code

Scanning the following reader configuration codes will enable/disable PROFINET.

NOTE

You must reboot the device for the change to take effect.



Setup Tool

MC Protocol can be enabled by checking **Enabled** on the Industrial Protocols pane's MC Protocol tab. Make sure you save the new selection by clicking Yes to the Reboot Required message window.

NOTE

You must reboot your reader for the new settings to take effect.

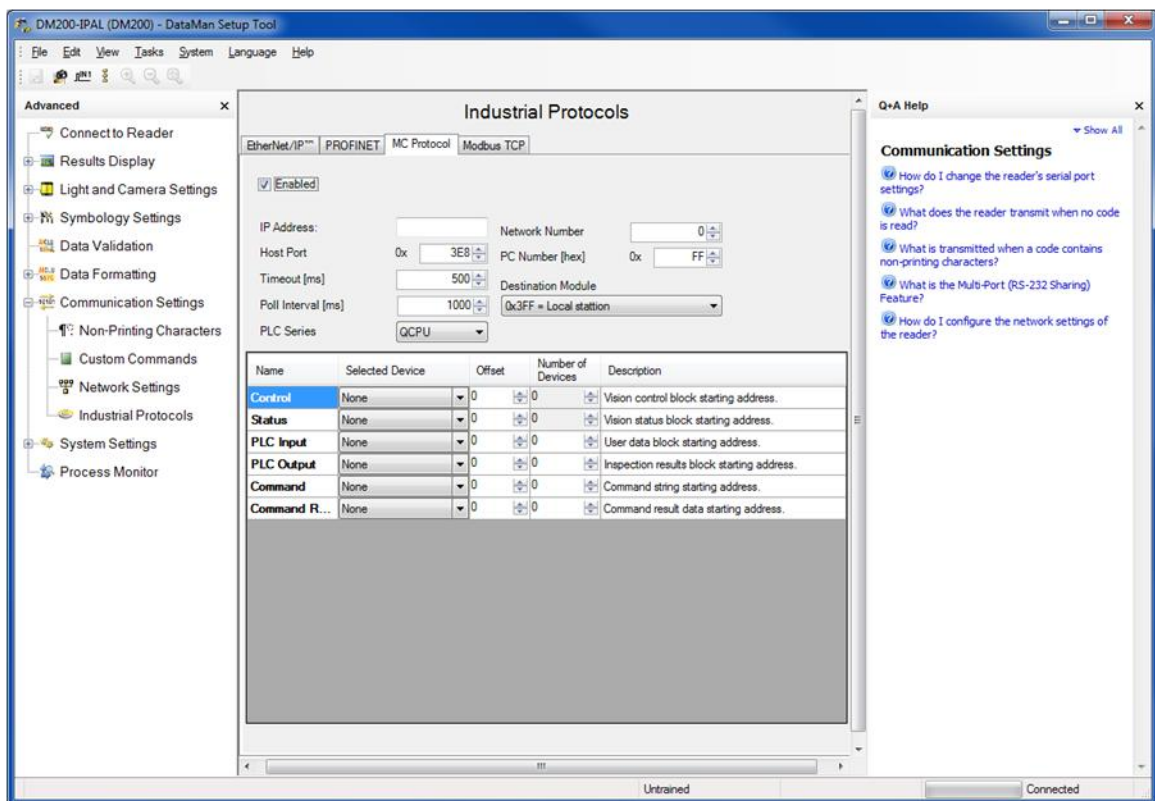
MC Protocol Scanner

MC Protocol on DataMan is implemented as a client type device also referred to as a scanner. All communication is initiated by the DataMan reader in the form of read and write requests. The PLC acts as a passive server reacting to the read and write requests. Since the PLC cannot initiate communication it relies on the reader to periodically ask (scan) the PLC for any actions or information that the PLC requires (such as triggering or retrieving read results).

Getting Started

By default, MC Protocol is not enabled on the DataMan reader. The protocol must be enabled and the protocol configuration parameters must be set to correctly interact with a PLC. Protocol configuration is accomplished via the DataMan Setup Tool.

1. From the Windows Start menu, start the Setup Tool.
2. Under Communication Settings, click the Industrial Protocols node.
3. Select the MC Protocol tab.



4. Enable the protocol and set the proper configuration settings.

MC Protocol configuration consists of two aspects; defining the network information and defining the data to be exchanged. All configuration parameters are accessed via the MC Protocol tab.

You must modify the "IP Address" to match the address of your PLC. Also, modify "Network Number", "PC Number" and "Destination Module" if they differ from your network.

NOTE

Make sure you select System → Save Settings to save any changes made to the MC Protocol configuration settings. Also, the reader must be rebooted for the new settings to take effect.

Network Configuration

The network configuration defines all the information that the DataMan reader needs to establish a connection with a PLC.

Name	Default	Range	Description
IP Address	<empty >	Any valid IP address	IP Address of the PLC to connect to.
Host Port (Hex)	3000	Any Hex port number 1000-FFFF	Port number of the MC Protocol channel on the PLC
Timeout (ms)	1000	5 - 30000	Time to in milliseconds for a response from the PLC to an MC Protocol message.
Poll Interval (ms)	1000	10 - 30000	Requested time in milliseconds between successive polls of the Control Block from the PLC.
PLC Series	QCPU	QCPU or LCPU	Defines frame type used. Currently only 3E supported.
Network Number	0	0 - 239	MC Protocol network number to communicate with (0 = local network)
PC Number	0xFF	1 - 120 = station on CC-Link IE field network adapter 126 = Master station on CC-Link IE field network 255 = Direct connect to local station	Station identifier on the specified network of the destination module.

Name	Default	Range	Description
Destination Module	0x3FF	0x3ff = Local station (default) 0x3d0 = Control system CPU 0x3d1 = Standby system CPU 0x3d2 = System A CPU 0x3d3 = System B CPU 0x3e0 = CPU 1 0x3e1 = CPU 2 0x3e2 = CPU 3 0x3e3 = CPU 4	Module identifier of the device to connect to.

Data Block Configuration

The data block configuration defines the data that will be exchanged between the DataMan reader and the PLC. Six data blocks are available. Each block has a predefined function.

Not all data blocks are required. Configure only those data blocks which are needed by your application. Typically the Control and Status blocks are defined because they control most data flow. However, there are some use cases where even these blocks are not required.

A data block is configured by defining the PLC Device type (that is, memory type), Device offset and Number of Devices contained in the data block. If either the Device type or Number of Devices is undefined, that block will not be used (that is, no data will be exchanged for that block).

Block Name	Supported Device Types	Offset	Number of Devices
Control	<none>, D, W, R, ZR, M, X, Y, L, F, B	0 - 65535	0, if type <none> 32, if bit type 2, if word type (read-only)
Status	<none>, D, W, R, ZR, M, X, Y, L, F, B	0 - 65535	0, if type <none> 32, if bit type 2, if word type (read-only)
PLC Input	None, D, W, R, ZR	0 - 65535	0 - 960
PLC Output	None, D, W, R, ZR	0 - 65535	0 - 960
Command	None, D, W, R, ZR	0 - 65535	0 - 960
Command Result	None, D, W, R, ZR	0 - 65535	0 - 960

Interface

This section describes the interface to the DataMan reader as seen by the PLC via MC Protocol. The interface model consists of 6 data blocks grouped in 3 logical pairs:

- Control and Status
- Input Data and Output Data
- String Command and String Response

Not all of the blocks are required. You may select which blocks are appropriate for your particular application. However, Control and Status will generally be included for most applications.

You can define the starting address and device type for each interface block that you choose to use in your application. Undefined blocks will not be exchanged. For any transfer (read or write) the entire block is sent, even if only one field within the block has changed value. The protocol implementation will minimize network use by grouping as many value changes as logically possible into a single transfer.

Control Block

The Control block contains bit type data. However, the block may be defined to exist in either bit or word memory in the PLC. This block consists of the control signals sent from the PLC to the reader. It is used by the PLC to initiate actions and acknowledge certain data transfers.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved				Results Ack	Buffer Results Enable	Trigger	Trigger Enable
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved							
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved						Initiate String Cmd	Set User Data
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Soft Event 7	Soft Event 6	Soft Event 5	Soft Event 4	Soft Event 3	Soft Event 2	Soft Event 1	Soft Event 0

Control Block Field Descriptions

Bit	Name	Description
0	Trigger Enable	This field is set to enable triggering via the <i>Trigger</i> bit. Clear this field to disable the network triggering mechanism.
1	Trigger	Setting this bit triggers an acquisition. Note, the <i>Trigger Ready</i> bit must be set high before triggering an acquisition.
2	Buffer Results Enable	When this bit is set, each read result set (<i>ResultID</i> , <i>ResultCode</i> , <i>ResultLength</i> and <i>ResultData</i> fields) will be held in the <i>Output Block</i> until it is acknowledged. Once acknowledged, the next set of read results will be made available from the buffer. If new read results arrive before the earlier set is acknowledged the new set will be queued in the reader's buffer. Up to 6 sets of read results can be held in the reader's buffer. Refer to Section Operation for a description of the acknowledgement handshake sequence.
3	ResultsAck	Set by the PLC to acknowledge that it has received the latest results (<i>ResultID</i> , <i>ResultCode</i> , <i>ResultLength</i> and <i>ResultData</i> fields). When the reader sees this bit transition from 0→1 it clears the <i>ResultsAvailable</i> bit. This forms a logical handshake between the PLC and reader. If result buffering is enabled, the acknowledgement will cause the next set of queued results to be moved from the buffer. See Section Operation for a description of the acknowledgement handshake sequence.
4-15	Reserved	Future use
16	SetUserData	Set by the PLC to signal that new <i>UserData</i> is available. After reading the new <i>UserData</i> the reader sets <i>SetUserDataAck</i> to signal that the transfer is complete. This forms a logical handshake between the PLC and reader.
17	Initiate StringCmd	Set by the PLC to signal that a new <i>StringCommand</i> is available. After processing the command the reader sets <i>StringCmdAck</i> to signal that the command result is available. This forms a logical handshake between the PLC and reader.
18-23	Reserved	Future use
24-31	SoftEvents	Bits act as virtual discrete inputs. When a bit transitions from 0→1 the associated action is executed. After executing the action the reader sets the corresponding <i>SoftEventAck</i> to signal that the action is complete. This forms a logical handshake between the PLC and reader. Bit0: Train code Bit1: Train match string Bit2: Train focus Bit3: Train brightness Bit4: Un-Train Bit5: Reserved (future use) Bit6: Execute DMCC command Bit7: Set match string

Status Block

The status block contains bit type data. However, the block may be defined to exist in either bit or word memory in the PLC. This block consists of the status signals sent from the reader to the PLC. It is used by the reader to signal status and handshake certain data transfers.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved				Missed Acq	Acquiring	Trigger Ack	Trigger Ready
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
General Fault	Reserved			Results Available	Results Buffer Overrun	Decode Complete Toggle	Decoding
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved						String Cmd Ack	Set User Data Ack
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
SoftEvent Ack 7	SoftEvent Ack 6	SoftEvent Ack 5	SoftEvent Ack 4	SoftEvent Ack 3	SoftEvent Ack 2	SoftEvent Ack 1	SoftEvent Ack 0

Status Block Field Descriptions

Bit	Name	Description
0	Trigger Ready	Indicates when the reader is ready to accept a new <i>Trigger</i> . The reader sets this bit when <i>TriggerEnable</i> has been set and the reader is ready to accept a new trigger.
1	TriggerAck	Indicates when the reader recognizes that <i>Trigger</i> has been set. This bit will remain set until the <i>Trigger</i> bit has been cleared.
2	Acquiring	Set to indicate that the reader is in the process of acquiring an image.
3	Missed Acq	Indicates that the reader missed a requested acquisition trigger. The bit is cleared when the next acquisition is issued.
4-7	Reserved	Future use
8	Decoding	Set to indicate that the reader is in the process of decoding an image.
9	Decode Complete Toggle	Indicates new result data is available. Bit toggles state (0→1 or 1→0) each time new result data becomes available.
10	Results Buffer Overrun	Set to indicate that the reader has discarded a set of read results because the PLC has not acknowledged the earlier results. Cleared when the next set of result data is successfully queued in the buffer. This bit only has meaning if result buffering is enabled.
11	Results Available	Set to indicate that new result data is available. Bit will remain set until acknowledged with <i>ResultsAck</i> even if additional new read results become available.
12-14	Reserved	Future use

Bit	Name	Description
15	General Fault	Set to indicate that an Ethernet communications fault has occurred. Currently only used by soft event operations. Bit will remain set until the next successful soft event or until <i>TriggerEnable</i> is set low and then high again.
16	Set User Data Ack	Set to indicate that the reader has received new <i>UserData</i> . Bit will remain set until the corresponding <i>SetUserData</i> bit is cleared. This forms a logical handshake between the PLC and reader.
17	String Cmd Ack	Set to indicate that the reader has completed processing the latest string command and that the command response is available. Bit will remain set until the corresponding <i>InitiateStringCmd</i> bit is cleared. This forms a logical handshake between the PLC and reader.
18-23	Reserved	Future use
24-31	SoftEvent Ack	Set to indicate that the reader has completed the soft event action. Bit will remain set until the corresponding SoftEvent bit is cleared. This forms a logical handshake between the PLC and reader. Bit0: Ack train code Bit1: Ack train match string Bit2: Ack train focus Bit3: Ack train brightness Bit4: Ack untrain Bit5: Reserved (future use) Bit6: Ack Execute DMCC command Bit7: Ack set match string

Input Data Block

The Input Data block contains word type data. This is data sent from the PLC to the reader. The block consists of user defined data that may be used as input to the acquisition/decode operation.

Word 0	Word 1	Word 2..N
Reserved	User Data Length	User Data

Input Data Block Field Descriptions

Word	Name	Description
0	Reserved	Future use
1	User Data Length	Number of bytes of valid data actually contained in the <i>UserData</i> field.
2..N	User Data	User defined data that may be used as an input to the acquisition/decode.

Output Data Block

The Output Data block contains word type data. This is data sent from the reader to the PLC. The block consists primarily of read result data.

Word 0	Word 1	Word 2	Word 3	Word 4	Word 5..N
Reserved	Trigger ID	Result ID	Result Code	Result Length	Result Data

Output Data Block Field Descriptions

Word	Name	Description
0	Reserved	Future use
1	Trigger ID	Trigger identifier. Identifier of the next trigger to be issued. Used to match issued triggers with result data that is received later. This same value will be returned as the <i>ResultID</i> of the corresponding read.
2	Result ID	Result set identifier. This is the value of <i>TriggerID</i> when the corresponding trigger was issued. Used to match up triggers with corresponding result data.
3	Result Code	Indicates the success or failure of the read that produced this result set. Bit0: 1=Read, 0=No read Bit1: 1=Validated, 0=Not Validated Bit2: 1=Verified, 0=Not Verified Bit3: 1=Acquisition trigger overrun Bit4: 1=Acquisition buffer overrun Bit5-15: Reserved (future use)
4	Result Data Length	Number of bytes of valid data actually in the <i>ResultData</i> field.
5..N	Result Data	Result data from this acquisition/decode.

String Command Block

The String Command block contains word type data. This is data sent from the PLC to the reader. The block is used to transport string based commands (DMCC) to the reader.

NOTE

Do not send string commands that change the reader configuration at the same time that reads are being triggered. Changing configuration during acquisition/decode can lead to unpredictable results.

Word 0	Word 1..N
Length	String Command

String Command Block Field Descriptions

Word	Name	Description
0	Length	Number of bytes of valid data in the <i>StringCommand</i> field.
1..N	String Command	ASCII text string containing the command to execute. No null termination required.

String Command Result Block

The String Command Result block contains word type data. This is data sent from the reader to the PLC. The block is used to transport the response from string based commands (DMCC) to the PLC.

Word 0	Word 1	Word 2..N
Result Code	Length	String Command Result

String Command Result Block Field Descriptions

Word	Name	Description
0	Result Code	Code value indicating the success or failure of the command. Refer to the <i>Command Reference</i> , available through the Windows Start menu, for specific values.
1	Length	Number of bytes of valid data in the <i>StringCommand</i> field.
2..N	String Command Result	ASCII text string containing the command to execute. No null termination required.

Operation

MC Protocol is a command/response based protocol. All communications are originated from the DataMan reader. The reader must send read requests to the PLC at a periodic interval to detect changes in the control bits.

Scanning

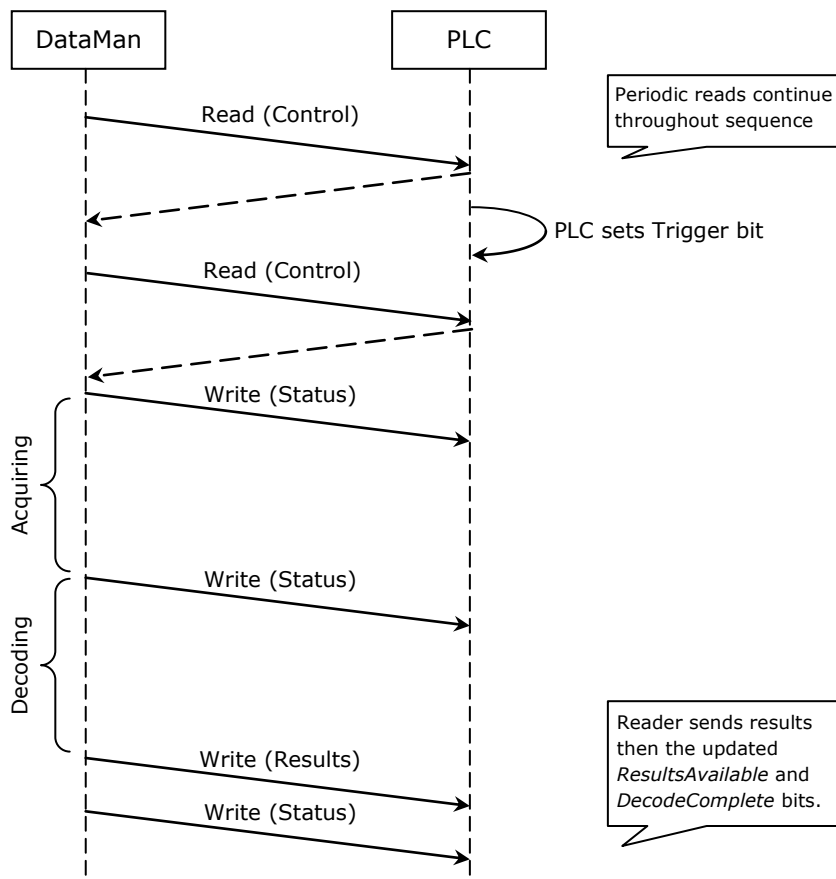
To initiate actions or control data transfer, the PLC changes the state of certain bits of the Control block. Since only the reader can initiate communications, the reader scans (that is, reads the Control block from the PLC) at a periodic rate. This rate is defined by the user.

After each scan, the reader will process changes in state of the bits in the Control block. Some state changes require additional communications with the PLC, such as writing updated acknowledge bit values or reading a new string command. These additional communications are handled automatically by the reader. Other state changes initiate

activities such as triggering a read or executing a soft event. The reader performs the requested action and later reports the results.

For any transfer (read or write), the entire interface block is sent, even if only one field within the block has changed value. The protocol implementation will minimize network usage by grouping as many value changes as logically possible into a single transfer.

Typical Sequence Diagram



Handshaking

A number of actions are accomplished by means of a logical handshake between the reader and PLC (triggering, transferring results, executing soft events, string commands, and so on). This is done to ensure that both sides of a transaction know the state of the operation on the opposite side. Network transmission delays will always introduce a finite time delay in transfer data and signals. Without this handshaking, it is possible that one side of a transaction might not detect a signal state change on the other side. Any operation that has both an initiating signal and corresponding acknowledge signal will use this basic handshake procedure.

The procedure involves a four-way handshake.

1. Assert signal
2. Signal acknowledge

3. De-assert signal
4. De-assert acknowledge

The requesting device asserts the signal to request an action (set bit 0→1). When the target device detects the signal and the requested operation has completed, it asserts the corresponding acknowledge (set bit 0→1). When the requesting device detects the acknowledge, it de-asserts the original signal (1→0). Finally, when the target device detects the original signal de-asserted, it de-asserts its acknowledge (bit 0→1). To function correctly both sides must see the complete assert/de-assert cycle (0→1 and 1→0). The requesting device should not initiate a subsequent request until the cycle completes.

Acquisition Sequence

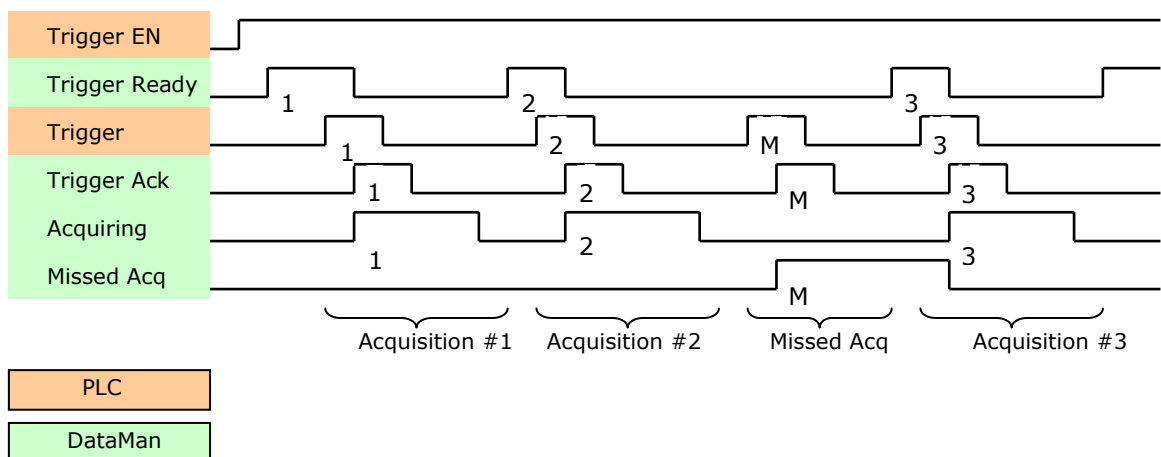
DataMan can be triggered to acquire images by several methods. It can be done via the MC Protocol by setting the *Trigger* bit or issuing a trigger String Command. It can also be done via DMCC command (Telnet) or hardwired trigger signal. The *Trigger* bit method will be discussed here.

On startup the *TriggerEnable* will be False. It must be set to True to enable triggering via the MC Protocol *Trigger* bit. When the device is ready to accept triggers, the reader will set the *TriggerReady* bit to True.

While the *TriggerReady* bit is True, each time the reader detects the *Trigger* bit change from 0→1, it will initiate a read. The *Trigger* bit should be held in the new state until that same state value is seen in the *TriggerAck* bit (this is a necessary handshake to guarantee that the trigger is seen by the reader).

During an acquisition, the *TriggerReady* bit will be cleared and the *Acquiring* bit will be set to True. When the acquisition is completed, the *Acquiring* bit will be cleared. When the device is ready to begin another image acquisition, the *TriggerReady* bit will again be set to True.

If results buffering is enabled, the reader will allow overlapped acquisition and decoding operations. *TriggerReady* will be set high after acquisition is complete but while decoding is still in process. This can be used to achieve faster overall trigger rates. If result buffering is not enabled, the *TriggerReady* bit will remain low until both the acquisition and decode operations have completed.



To force a reset of the trigger mechanism set the *TriggerEnable* to False until *TriggerReady* is also set to False. Then, *TriggerEnable* can be set to True to re-enable acquisition.

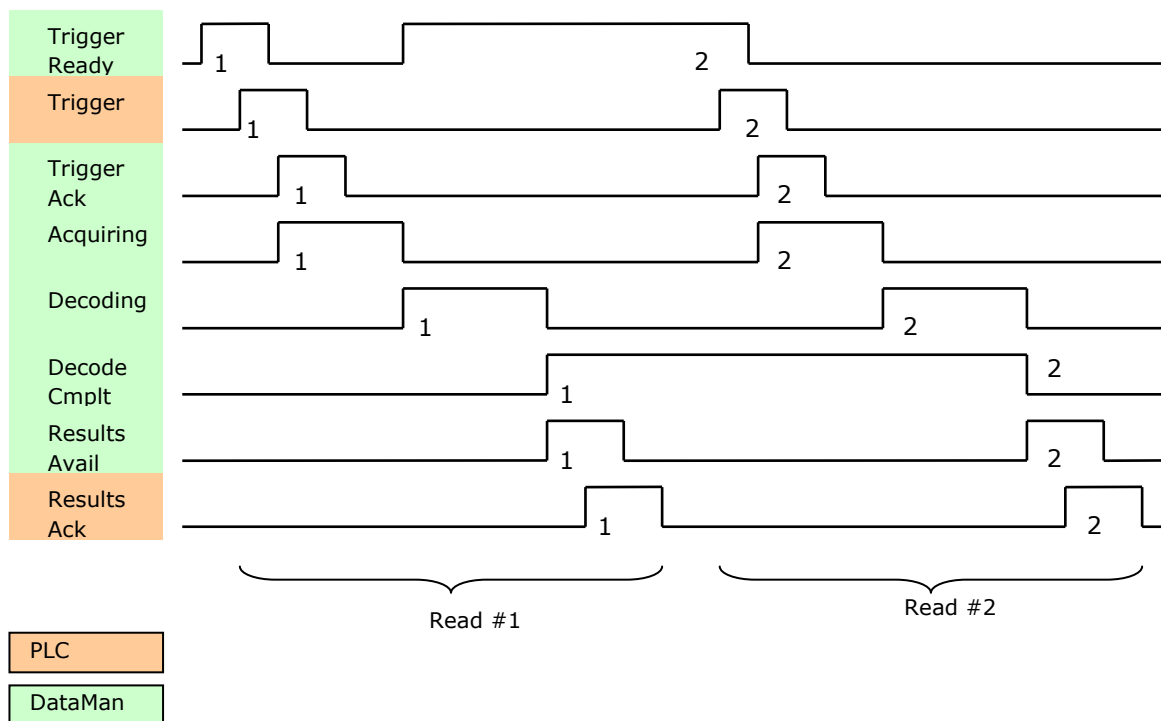
As a special case, an acquisition can be cancelled by clearing the *Trigger* signal before the read operation has completed. This allows for the cancellation of reads in Presentation and Manual mode if no code is in the field of view. To ensure that a read is not unintentionally cancelled, it is advised that the PLC hold the *Trigger* signal True until both *TriggerAck* and *ResultsAvailable* are True (or *DecodeComplete* toggles state).

Decode / Result Sequence

After an image is acquired, it is decoded. While being decoded, the *Decoding* bit is set. When the decode operation has completed, the *Decoding* bit is cleared. The *ResultsBufferEnable* determines how decode results are handled by the reader.

If *ResultsBufferEnable* is set to False, then the read results are immediately placed into the Output Data block, *ResultsAvailable* is set to True and *DecodeComplete* is toggled.

If *ResultsBufferEnable* is set to True, the new results are queued in a buffer and *DecodeComplete* is toggled. The earlier read results remain in the Output Data block until they are acknowledged by the PLC. After the acknowledgment handshake, if there are more results in the queue, the next set of results will be placed in the Output Data block and *ResultsAvailable* is set to True.



Results Buffering

There is an option to enable a queue for read results. If enabled, this allows a finite number of sets of result data to be queued up until the PLC has time to read them. This is useful to smooth out data flow if the PLC slows down for short periods of time.

Also, if result buffering is enabled the reader will allow overlapped acquisition and decode operations. Depending on the application this can be used to achieve faster overall trigger rates. See Acquisition Sequence description above for further detail.

In general, if reads are occurring faster than results can be sent out, the primary difference between buffering or not buffering determines which results get discarded. If buffering is not enabled, the most recent results are kept and the earlier result (which was not read by the PLC quickly enough) is lost. The more recent result will overwrite the earlier result. If buffering is enabled (and the queue becomes full) the most recent results are discarded until room becomes available in the results queue.

SoftEvents

SoftEvents act as “virtual” inputs. When the value of a *SoftEvent* bit changes from 0 → 1 the action associated with the event will be executed. When the action completes, the corresponding *SoftEventAck* bit will change from 0 → 1 to signal completion.

The *SoftEvent* and *SoftEventAck* form a logical handshake. After *SoftEventAck* changes to 1, the original *SoftEvent* should be set back to 0. When that occurs, *SoftEventAck* will automatically be set back to 0.

NOTE

Do not execute soft events that change the reader configuration at the same time that reads are being triggered. Changing configuration during acquisition/decode can lead to unpredictable results.

The “ExecuteDMCC” and “SetMatchString” soft event actions require user supplied data. This data must be written to the *UserData* and *UserDataLength* area of the Input Data block prior to invoking the soft event. Since both of these soft events depend on the *UserData*, only one may be invoked at a time.

String Commands

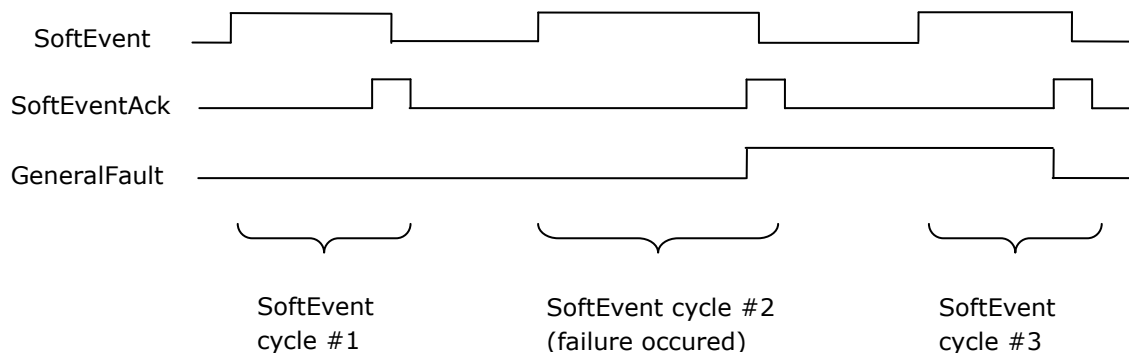
The DataMan MC Protocol implementation includes a String Command feature. This feature allows you to execute string-based DMCC commands over the MC protocol connection. The DMCC command is sent to the reader via the String Command block. The DMCC command result is returned via the String Command Result block. Initiating a command and notification of completion is accomplished by signaling bits in the Control and Status blocks.

To execute a DMCC command, the command string is placed in the data field of the String Command block. The command string consists of standard ASCII text. The command format is exactly the same as would be used for a serial (RS-232) or Telnet connection. The string does not need to be terminated with a null character. Instead, the length of the string (that is, the number of ASCII characters) is placed in the length field of the String Command block.

After executing the DMCC command, the result string is returned in the String Command Result block. Similar to the original command, the result string consists of ASCII characters in the same format as would be returned via serial or Telnet. Also, there is no terminating null character. Instead the length of the result is returned in the Command String Result length field. The Command String Result block also contains a numeric result code. This allows you to determine the success or failure of the command without having to parse the text string. The values of the result code are defined in the DMCC documentation.

General Fault Indicator

When an MC Protocol communication-related fault occurs, the "GeneralFault" bit will change from 0 → 1. Currently the only fault conditions supported are soft event operations. If a soft event operation fails, the fault bit will be set. The fault bit will remain set until the next successful soft event operation, or, until *TriggerEnable* is set to 0 and then back to 1.



Examples

Included with the Setup Tool installer is an example PLC program created with Mitsubishi (GX Works2) software. This simple program clearly demonstrates DataMan ID readers' capabilities and proper operation. The same operations can be achieved by using more advanced features and efficient programming practices with Mitsubishi PLCs. However, such an advanced program is less useful for demonstration purposes.

Function

The example application demonstrates the following operations:

1. Triggering a read
2. Getting read results
3. Executing string commands (DMCC)
4. Executing soft event operations
 - a. Train code
 - b. Train match string
 - c. Train focus
 - d. Train brightness
 - e. Un-train
 - f. Execute DMCC
 - g. Set match string

The "Main" program contains a PLC ladder rung to invoke each of these operations. The operation is invoked by toggling the control bit on the rung from 0 → 1. This will invoke the associated subroutine to perform the operation. When the operation is complete, the subroutine will set the control bit back to 0.



Triggering a Read

The example provides two trigger options; "Continuous Trigger" and "Single Trigger". As the name implies, enabling the "Continuous Trigger" bit will invoke a continuous series of read operations. Once enabled, the "Continuous Trigger" control bit will remain set until you disable it. The "Single Trigger" control bit invokes a single read operation. This control bit will automatically be cleared when the read is completed.

Primarily, the trigger subroutine manages the trigger handshake operation between the PLC and the reader. The control *Trigger* bit is set, the PLC waits for the corresponding *TriggerAck* status bit from the reader, and the control *Trigger* bit is reset. Refer to a description of handshaking in Section [Operation](#).

The trigger subroutine contains a delay timer. This is not required for operation. It exists simply to add an adjustable artificial delay between reads for demonstration purposes.

Getting Read Results

For this example the operation of triggering a read and getting read results was intentionally separated. This is to support the situation where the PLC is not the source of the read trigger. For example, the reader may be configured to use a hardware trigger. In such a case, only the get results subroutine would be needed.

Like the triggering subroutine, the get results subroutine manages the results handshake operation between the PLC and the reader. However, it also copies the result data to internal storage. The routine waits for the *ResultsAvailable* status bit to become active, it copies the result data to internal storage, and then executes the *ResultsAck* handshake. Refer to a description of handshaking in Section [Operation](#).

The read result consists of a *ResultCode*, *ResultLength*, and *ResultData*. Refer to Section [Output Data Block Field Descriptions](#) for details of the *ResultCode* values. The *ResultLength* field indicates how many bytes of actual result data exist in the *ResultData* field. The subroutine converts this byte length to word length before copying the results to internal storage.

The get results subroutine gathers read statistics (number of good reads, number of no-reads, and so on). This is not required for operation. It is simply for demonstration purposes.

Execute String Commands (DMCC)

The string command feature provides a simple way to invoke DMCC commands from the PLC. The command format and command result format is exactly identical to that used for serial or Telnet DMCC operation.

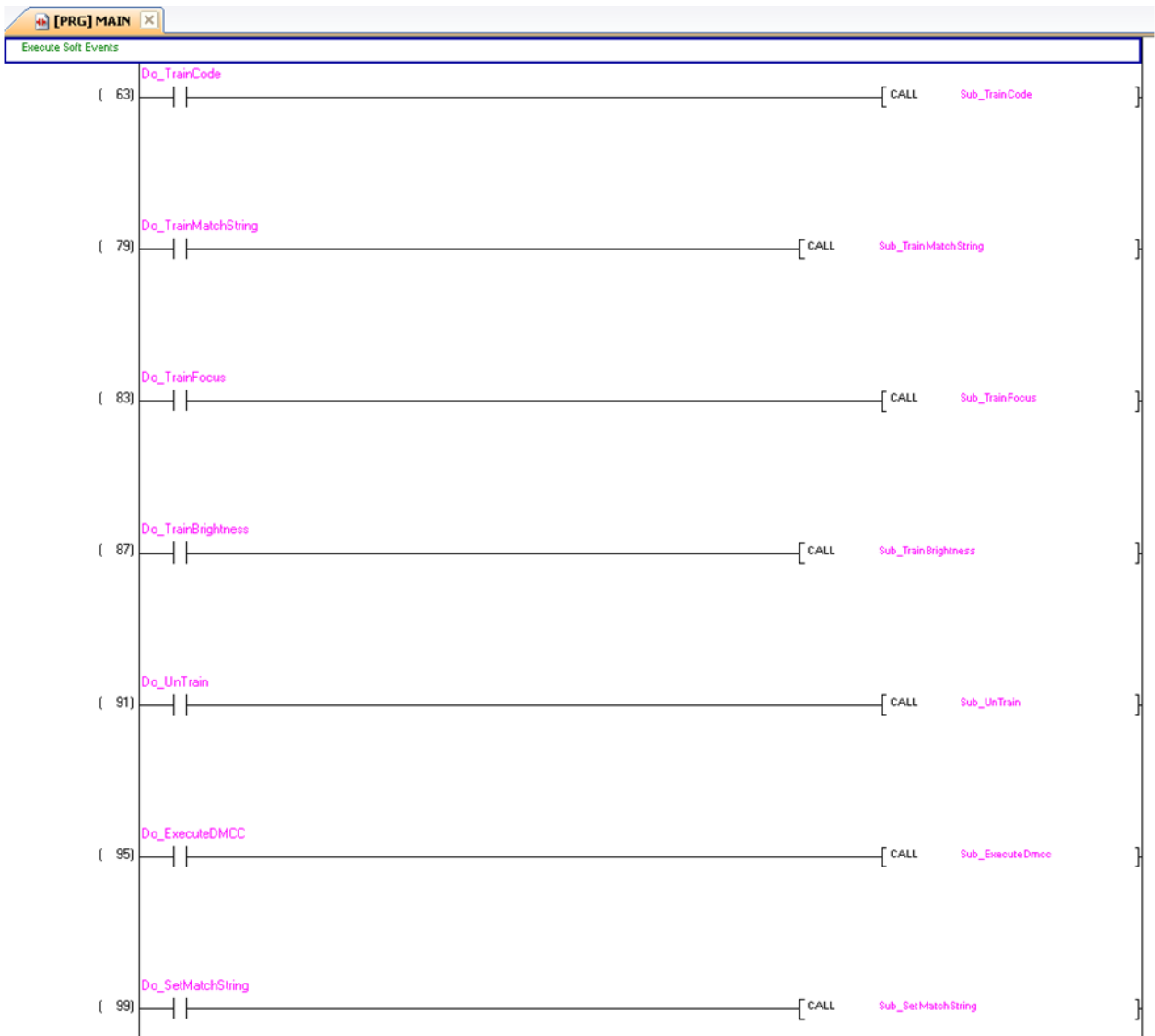
This subroutine copies an example DMCC command (`||>GET CAMERA.EXPOSURE`) to the String Command block and then manages the string command handshake operation between the PLC and the reader to invoke the command and retrieve the command result. Any valid DMCC command may be invoked with this mechanism. Refer to the *DataMan Command Reference* document available through the Windows Start menu.

Execute Soft Events

Soft Events are used to invoke a predefined action. Each Soft Event is essentially a virtual input signal. Each of the soft event subroutines manages the handshake operation between the PLC and the reader to invoke the predefined action. The associated action is invoked when the *SoftEvent* bit toggles from 0 → 1. The subroutine then watches for the associated *SoftEventAck* bit from the reader which signals that the action is complete. For a description of handshaking, see Section [Operation](#).

NOTE

The “Execute DMCC” and “Set Match String” soft events make use of the Input Data block. The subroutine for these two events copies the relevant data into the User Data fields of the Input Data block and then invokes the User Data subroutine to transfer the data to the reader. Only after the user data is transferred is the actual soft event action invoked. It is required that the user data be transferred before invoking either of these events.



NOTE

The "Train Match String" soft event only prepares the training mechanism. The actual training occurs on the next read operation. Therefore, a trigger must be issued following "Train Match String".

Modbus TCP

Modbus is an application layer protocol. It provides client/server communication between devices connected to different types of buses or networks. Modbus is a request/response protocol, whose services are specified by using function codes.

Modbus TCP provides the Modbus protocol using TCP/IP. System port 502 is reserved for Modbus communication. It uses standard Ethernet hardware and software to exchange I/O data and diagnostics. DataMan provides Modbus TCP server functionality only.

By default, DataMan has the Modbus TCP protocol disabled. The protocol can be enabled in the Setup Tool, via DMCC, or by scanning a parameter code.

DMCC

The following commands can be used to enable/disable Modbus TCP. The commands can be issued via RS-232 or Telnet connection.

NOTE

Because you have to make changes to the Telnet client provided by Windows to communicate with DataMan, it is recommended you use third party clients such as PuTTY.

Enable:

```
||>SET MODBUSTCP.ENABLED ON  
||>CONFIG.SAVE  
||>REBOOT
```

Disable:

```
||>SET MODBUSTCP.ENABLED OFF  
||>CONFIG.SAVE  
||>REBOOT
```

Reader Configuration Code

Scanning the following reader configuration codes will enable/disable Modbus TCP.

NOTE

You must reboot the device for the change to take effect.



Setup Tool

Modbus TCP can be enabled by checking **Enabled** on the Industrial Protocols pane's Modbus TCP tab. Make sure you save the new selection by clicking Yes to the Reboot Required message window.

NOTE

You must reboot your reader for the new settings to take effect.

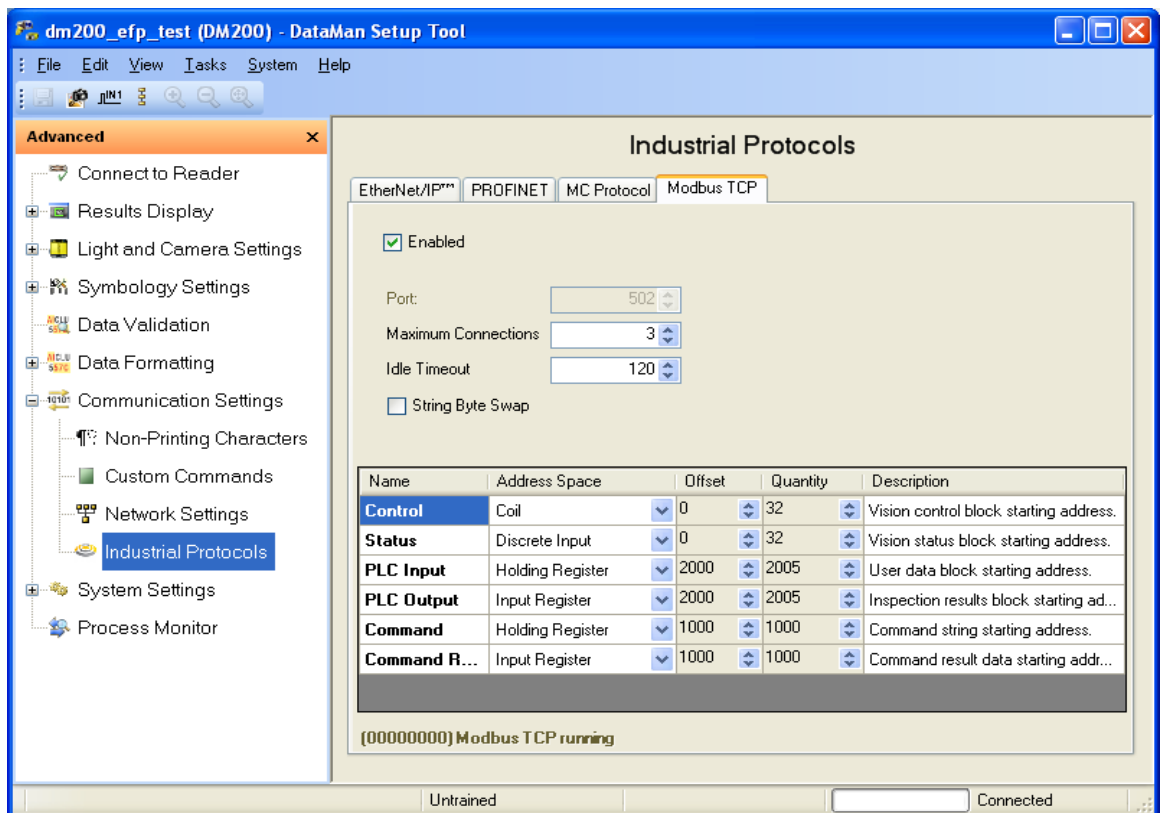
Modbus TCP Handler

Modbus TCP on DataMan is implemented as a server type device. All communication is initiated by the PLC in the form of read and write requests. The PLC acts as a client which actively sends read and write requests.

Getting Started

By default, Modbus TCP is not enabled on the DataMan reader. The protocol must be enabled and the protocol configuration parameters must be set to correctly interact with a PLC. Protocol configuration is accomplished via the DataMan Setup Tool.

1. From the Windows Start menu, start the Setup Tool.
2. Under Communication Settings, click the Industrial Protocols node.
3. Select the Modbus TCP tab.



4. Enable the protocol and set the proper configuration settings.

Modbus TCP configuration consists of two aspects; defining the network information and defining the data to be exchanged. All configuration parameters are accessed via the Modbus TCP tab.

NOTE

Make sure you select System → Save Settings to save any changes made to the Modbus TCP configuration settings. Also, the reader must be rebooted for the new settings to take effect.

Network Configuration

The network configuration defines all the information that the DataMan reader needs to establish a connection with a PLC. In most cases the default values may be used and no changes are need.

Name	Default	Range	Description
Host Port	502	Fixed	Port number where Modbus TCP can be accessed on this reader.
Max Connections	3	1 - 6	Maximum number of simultaneous Modbus TCP connections.
Idle Timeout (seconds)	120	1 - 3600	Timeout period after which the Modbus TCP connection will be closed. If no traffic is received on a Modbus TCP connection for this amount of time, the connection will automatically be closed.
String byte swap	False	True / False	String byte swap enable. If set to True, bytes within each register that forms a string will be swapped.

Data Block Configuration

The data block configuration defines the data that will be exchanged between the DataMan reader and the PLC. Six data blocks are available. Each block has a predefined function.

DataMan only supports Modbus TCP server operation. For server operation, only two configuration options exist. By default the 'Control' and 'Status' data blocks are located in bit address space (Coil and Discrete Input). If needed, one or both of these data blocks may be redefined to exist in register address space (Holding Register and Input Register). All other data block configurations are fixed.

Block Name	Address Space	Offset	Quantity
Control	Coil or Holding Register	0	32, if coil 2, if holding register
Status	Discrete Input or Input Register	0	32, if discrete input 2, if input register
PLC Input	Holding Register	2000	1 - 2005
PLC Output	Input Register	2000	1 - 2005

Block Name	Address Space	Offset	Quantity
Command String	Holding Register	1000	1 - 1000
Command String Result	Input Register	1000	1 - 1000

Interface

This section describes the interface to the DataMan reader as seen by the PLC via Modbus TCP. The interface model consists of 6 data blocks grouped in 3 logical pairs:

- Control and Status
- Input Data and Output Data
- String Command and String Response

The following list contains the data blocks, their start/end addresses and their length.

Data block Name	Start Address	End address	Size	Description
Control block	0	31	2 words	bit access
Status block	0	31	2 words	bit access
Input data block	2000	4004	2005 words	word access
Output data block	2000	4004	2005 words	word access
String Command block	1000	1999	1000 words	word access
String Response block	1000	1999	1000	word access

Control Block

The Control block contains bit type data. This block consists of the control signals sent from the PLC to the reader. It is used by the PLC to initiate actions and acknowledge certain data transfers.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved				Results Ack	Buffer Results Enable	Trigger	Trigger Enable
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved							
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved						Initiate String Cmd	Set User Data
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Soft Event 7	Soft Event 6	Soft Event 5	Soft Event 4	Soft Event 3	Soft Event 2	Soft Event 1	Soft Event 0

Control Block Field Descriptions

Bit	Name	Description
0	Trigger Enable	This field is set to enable triggering via the <i>Trigger</i> bit. Clear this field to disable the network triggering mechanism.
1	Trigger	Setting this bit triggers an acquisition. Note that the <i>Trigger Ready</i> bit must be set high before triggering an acquisition.
2	Buffer Results Enable	When this bit is set, each read result (<i>ResultID</i> , <i>ResultCode</i> , <i>ResultLength</i> and <i>ResultData</i> fields) will be held in the <i>Output Block</i> until it is acknowledged. Once acknowledged, the next set of read results will be made available from the buffer. If new read results arrive before the earlier set is acknowledged the new set will be queued in the reader's buffer. Up to 6 read results can be held in the reader's buffer. Refer to Section Operation for a description of the acknowledgement handshake sequence.
3	ResultsAck	Set by the PLC to acknowledge that it has received the latest results (<i>ResultID</i> , <i>ResultCode</i> , <i>ResultLength</i> and <i>ResultData</i> fields). When the reader sees this bit transition from 0→1 it clears the <i>ResultsAvailable</i> bit. This forms a logical handshake between the PLC and reader. If result buffering is enabled, the acknowledgement will cause the next set of queued results to be moved from the buffer. See Section Operation for a description of the acknowledgement handshake sequence.
4-15	Reserved	Future use
16	SetUserData	Set by the PLC to signal that new <i>UserData</i> is available. After reading the new <i>UserData</i> the reader sets <i>SetUserDataAck</i> to signal that the transfer is complete. This forms a logical handshake between the PLC and reader.
17	Initiate StringCmd	Set by the PLC to signal that a new <i>StringCommand</i> is available. After processing the command, the reader sets <i>StringCmdAck</i> to signal that the command result is available. This forms a logical handshake between the PLC and reader.
18-23	Reserved	Future use

24-31	SoftEvents	<p>Bits act as virtual discrete inputs. When a bit transitions from 0→1 the associated action is executed. After executing the action the reader sets the corresponding <i>SoftEventAck</i> to signal that the action is complete. This forms a logical handshake between the PLC and reader.</p> <p>Bit0: Train code Bit1: Train match string Bit2: Train focus Bit3: Train brightness Bit4: Un-Train Bit5: Reserved (future use) Bit6: Execute DMCC command Bit7: Set match string</p>
-------	------------	---

Status Block

The status block contains bit type data. This block consists of the status signals sent from the reader to the PLC. It is used by the reader to signal status and handshake certain data transfers.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved				Missed Acq	Acquiring	Trigger Ack	Trigger Ready
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
General Fault	Reserved			Results Available	Results Buffer Overrun	Decode Complete Toggle	Decoding
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved						String Cmd Ack	Set User Data Ack
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
SoftEvent Ack 7	SoftEvent Ack 6	SoftEvent Ack 5	SoftEvent Ack 4	SoftEvent Ack 3	SoftEvent Ack 2	SoftEvent Ack 1	SoftEvent Ack 0

Status Block Field Descriptions

Bit	Name	Description
0	Trigger Ready	Indicates when the reader is ready to accept a new <i>Trigger</i> . The reader sets this bit when <i>TriggerEnable</i> has been set and the reader is ready to accept a new trigger.
1	TriggerAck	Indicates when the reader recognizes that <i>Trigger</i> has been set. This bit will remain set until the <i>Trigger</i> bit has been cleared.
2	Acquiring	Set to indicate that the reader is in the process of acquiring an image.
3	Missed Acq	Indicates that the reader missed a requested acquisition trigger. The bit is cleared when the next acquisition is issued.
4-7	Reserved	Future use

Bit	Name	Description
8	Decoding	Set to indicate that the reader is in the process of decoding an image.
9	Decode Complete Toggle	Indicates new result data is available. Bit toggles state (0→1 or 1→0) each time new result data becomes available.
10	Results Buffer Overrun	Set to indicate that the reader has discarded a set of read results because the PLC has not acknowledged the earlier results. Cleared when the next set of result data is successfully queued in the buffer. This bit only has meaning if result buffering is enabled.
11	Results Available	Set to indicate that new result data is available. Bit will remain set until acknowledged with <i>ResultsAck</i> even if additional new read results become available.
12-14	Reserved	Future use
15	General Fault	Set to indicate that an Ethernet communications fault has occurred. Currently only used by soft event operations. Bit will remain set until the next successful soft event or until <i>TriggerEnable</i> is set low and then high again.
16	Set User Data Ack	Set to indicate that the reader has received new <i>UserData</i> . Bit will remain set until the corresponding <i>SetUserData</i> bit is cleared. This forms a logical handshake between the PLC and reader.
17	String Cmd Ack	Set to indicate that the reader has completed processing the latest string command and that the command response is available. Bit will remain set until the corresponding <i>InitiateStringCmd</i> bit is cleared. This forms a logical handshake between the PLC and reader.
18-23	Reserved	Future use
24-31	SoftEvent Ack	Set to indicate that the reader has completed the soft event action. Bit will remain set until the corresponding SoftEvent bit is cleared. This forms a logical handshake between the PLC and reader. Bit0: Ack train code Bit1: Ack train match string Bit2: Ack train focus Bit3: Ack train brightness Bit4: Ack untrain Bit5: Reserved (future use) Bit6: Ack Execute DMCC command Bit7: Ack set match string

Input Data Block

The Input Data block is sent from the PLC to the reader. The block consists of user defined data that may be used as input to the acquisition/decode operation.

Word 0	Word 1	Word 2..N
Reserved	User Data Length	User Data

Input Data Block Field Descriptions

Word	Name	Description
0	Reserved	Future use
1	User Data Length	Number of bytes of valid data actually contained in the <i>UserData</i> field.
2..N	User Data	User defined data that may be used as an input to the acquisition/decode.

Output Data Block

The Output Data block is sent from the reader to the PLC. The block consists primarily of read result data.

Word 0	Word 1	Word 2	Word 3	Word 4	Word 5..N
Reserved	Trigger ID	Result ID	Result Code	Result Length	Result Data

Output Data Block Field Descriptions

Word	Name	Description
0	Reserved	Future use
1	Trigger ID	Trigger identifier. Identifier of the next trigger to be issued. Used to match issued triggers with result data that is received later. This same value will be returned as the <i>ResultID</i> of the corresponding read.
2	Result ID	Result identifier. This is the value of <i>TriggerID</i> when the corresponding trigger was issued. Used to match up triggers with corresponding result data.
3	Result Code	Indicates the success or failure of the read that produced this result set. Bit0: 1=Read, 0=No read Bit1: 1=Validated, 0=Not Validated Bit2: 1=Verified, 0=Not Verified Bit3: 1=Acquisition trigger overrun Bit4: 1=Acquisition buffer overrun Bit5-15: Reserved (future use)
4	Result Data Length	Number of bytes of valid data actually in the <i>ResultData</i> field.
5..N	Result Data	Result data from this acquisition/decode.

String Command Block

The String Command block is sent from the PLC to the reader. The block is used to transport string based commands (DMCC) to the reader.

NOTE

Do not send string commands that change the reader configuration at the same time that reads are being triggered. Changing configuration during acquisition/decode can lead to unpredictable results.

Word 0	Word 1..N
Length	String Command

String Command Block Field Descriptions

Word	Name	Description
0	Length	Number of bytes of valid data in the <i>StringCommand</i> field.
1..N	String Command	ASCII text string containing the command to execute. No null termination required.

String Command Result Block

The String Command Result block is sent from the reader to the PLC. The block is used to transport the response from string based commands (DMCC) to the PLC.

Word 0	Word 1	Word 2..N
Result Code	Length	String Command Result

String Command Result Block Field Descriptions

Word	Name	Description
0	Result Code	Code value indicating the success or failure of the command. Refer to the <i>Command Reference</i> , available through the Windows Start menu, for specific values.
1	Length	Number of bytes of valid data in the <i>StringCommand</i> field.
2..N	String Command Result	ASCII text string containing the command to execute. No null termination required.

Operation

Modbus TCP is a request/response based protocol. All communications are originated from the PLC. The reader acts as server.

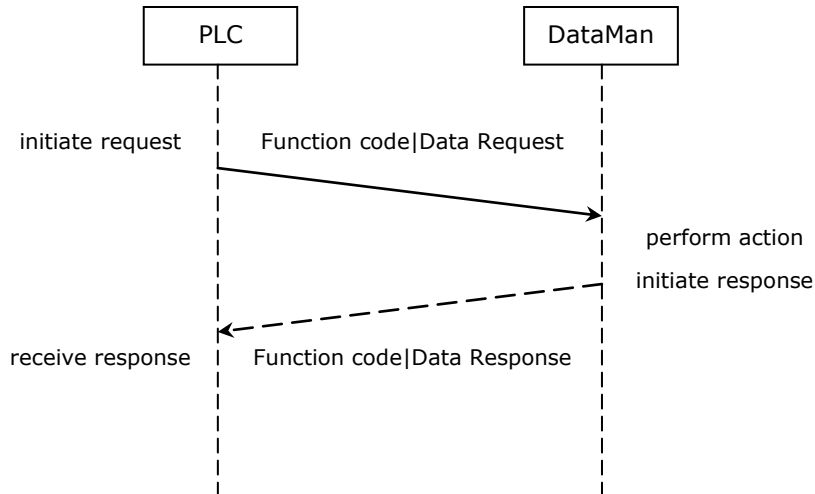
Requests

To initiate actions or control data transfer, the PLC changes the state of certain bits of the Control block and sends requests to the reader.

After each request, the reader will process changes in state of the bits in the Control block. Some state changes require additional communications with the PLC, such as writing updated acknowledge bit values or reading a new string command. These additional communications are handled automatically by the reader. Other state changes initiate

activities such as triggering a read or executing a soft event. The reader performs the requested action and later reports the results.

Typical Sequence Diagram



Handshaking

A number of actions are accomplished by means of a logical handshake between the reader and the PLC (triggering, transferring results, executing soft events, string commands, and so on). This is done to ensure that both sides of a transaction know the state of the operation on the opposite side. Network transmission delays will always introduce a finite time delay in transfer data and signals. Without this handshaking, it is possible that one side of a transaction might not detect a signal state change on the other side. Any operation that has both an initiating signal and corresponding acknowledge signal will use this basic handshake procedure.

The procedure involves a four-way handshake.

1. Assert signal
2. Signal acknowledge
3. De-assert signal
4. De-assert acknowledge

The requesting device asserts the signal to request an action (set bit 0→1). When the target device detects the signal and the requested operation has completed, it asserts the corresponding acknowledge (set bit 0→1). When the requesting device detects the acknowledge, it de-asserts the original signal (1→0). Finally, when the target device detects the original signal de-asserted, it de-asserts its acknowledge (bit 0→1). To function correctly both sides must see the complete assert/de-assert cycle (0→1 and 1→0). The requesting device should not initiate a subsequent request until the cycle completes.

Acquisition Sequence

DataMan can be triggered to acquire images by several methods. It can be done by setting the *Trigger* bit or issuing a trigger String Command. It can also be done via DMCC

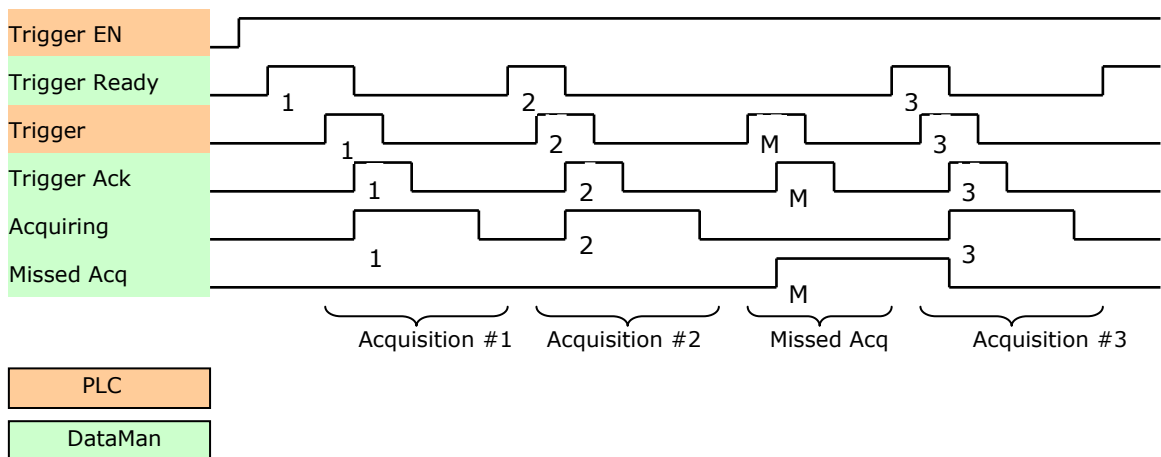
command (Telnet) or hardwired trigger signal. The *Trigger* bit method will be discussed here.

On startup the *TriggerEnable* will be False. It must be set to True to enable triggering via the *Trigger* bit. When the device is ready to accept triggers, the reader will set the *TriggerReady* bit to True.

While the *TriggerReady* bit is True, each time the reader detects the *Trigger* bit change from 0→1, it will initiate a read. The *Trigger* bit should be held in the new state until that same state value is seen in the *TriggerAck* bit (this is a necessary handshake to guarantee that the trigger is seen by the reader).

During an acquisition, the *TriggerReady* bit will be cleared and the *Acquiring* bit will be set to True. When the acquisition is completed, the *Acquiring* bit will be cleared. When the device is ready to begin another image acquisition, the *TriggerReady* bit will again be set to True.

If results buffering is enabled, the reader will allow overlapped acquisition and decoding operations. *TriggerReady* will be set high after acquisition is complete but while decoding is still in process. This can be used to achieve faster overall trigger rates. If result buffering is not enabled, the *TriggerReady* bit will remain low until both the acquisition and decode operations have completed.



To force a reset of the trigger mechanism set the *TriggerEnable* to False until *TriggerReady* is also set to False. Then, *TriggerEnable* can be set to True to re-enable acquisition.

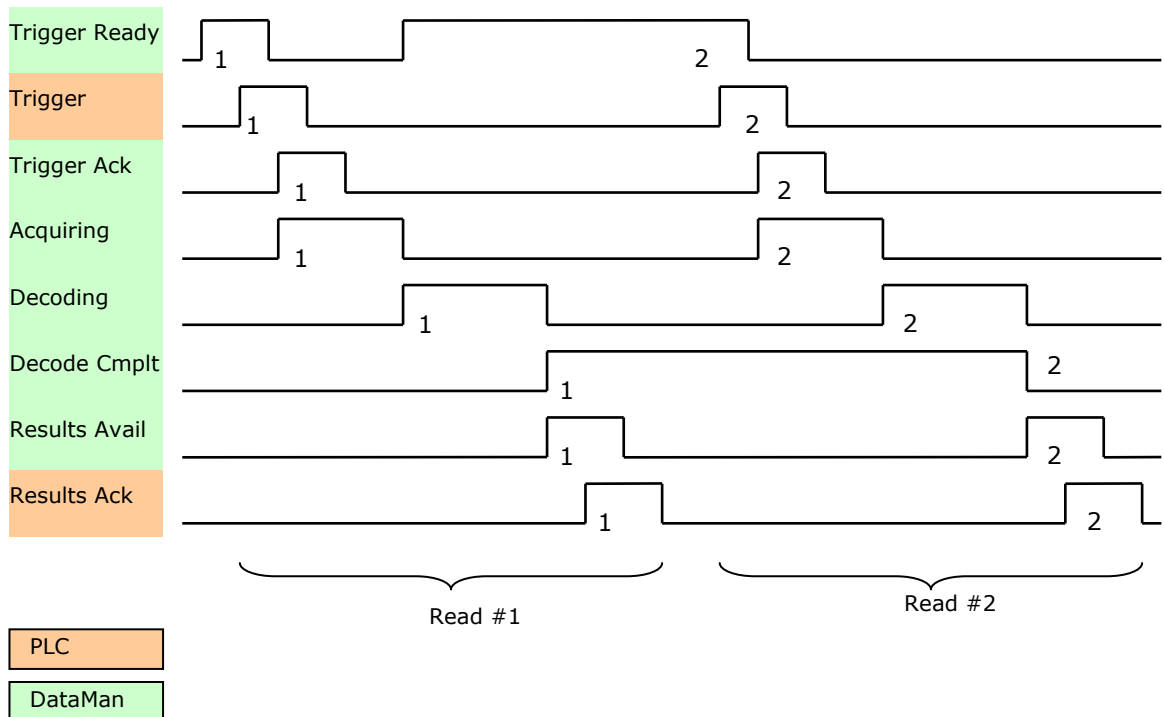
As a special case, an acquisition can be cancelled by clearing the *Trigger* signal before the read operation has completed. This allows for the cancellation of reads in Presentation and Manual mode if no code is in the field of view. To ensure that a read is not unintentionally cancelled, it is advised that the PLC hold the *Trigger* signal True until both *TriggerAck* and *ResultsAvailable* are True (or *DecodeComplete* toggles state).

Decode / Result Sequence

After an image is acquired, it is decoded. While being decoded, the *Decoding* bit is set. When the decode operation has completed, the *Decoding* bit is cleared. The *ResultsBufferEnable* determines how decode results are handled by the reader.

If *ResultsBufferEnable* is set to False, then the read results are immediately placed into the Output Data block, *ResultsAvailable* is set to True and *DecodeComplete* is toggled.

If *ResultsBufferEnable* is set to True, the new results are queued in a buffer and *DecodeComplete* is toggled. The earlier read results remain in the Output Data block until they are acknowledged by the PLC. After the acknowledgment handshake, if there are more results in the queue, the next set of results will be placed in the Output Data block and *ResultsAvailable* is set to True.



Results Buffering

There is an option to enable a queue for read results. If enabled, this allows a finite number of sets of result data to be queued up until the PLC has time to read them. This is useful to smooth out data flow if the PLC slows down for short periods of time.

Also, if result buffering is enabled the reader will allow overlapped acquisition and decode operations. Depending on the application this can be used to achieve faster overall trigger rates. See the Acquisition Sequence description for further details.

In general, if reads are occurring faster than results can be transferred to the PLC, some data will be lost. The primary difference between buffering or not buffering determines which results get discarded. If buffering is not enabled, the most recent results are kept and the earlier result (which was not read by the PLC quickly enough) is lost. The more recent result will overwrite the earlier result. If buffering is enabled (and the queue becomes full) the most recent results are discarded until room becomes available in the results queue.

SoftEvents

SoftEvents act as "virtual" inputs. When the value of a *SoftEvent* bit changes from 0 → 1 the action associated with the event will be executed. When the action completes, the corresponding *SoftEventAck* bit will change from 0 → 1 to signal completion.

The *SoftEvent* and *SoftEventAck* form a logical handshake. After *SoftEventAck* changes to 1, the original *SoftEvent* should be set back to 0. When that occurs, *SoftEventAck* will automatically be set back to 0.

NOTE

Do not execute soft events that change the reader configuration at the same time that reads are being triggered. Changing configuration during acquisition/decode can lead to unpredictable results.

The "ExecuteDMCC" and "SetMatchString" soft event actions require user supplied data. This data must be written to the *UserData* and *UserDataLength* area of the Input Data block prior to invoking the soft event. Since both of these soft events depend on the *UserData*, only one may be invoked at a time.

String Commands

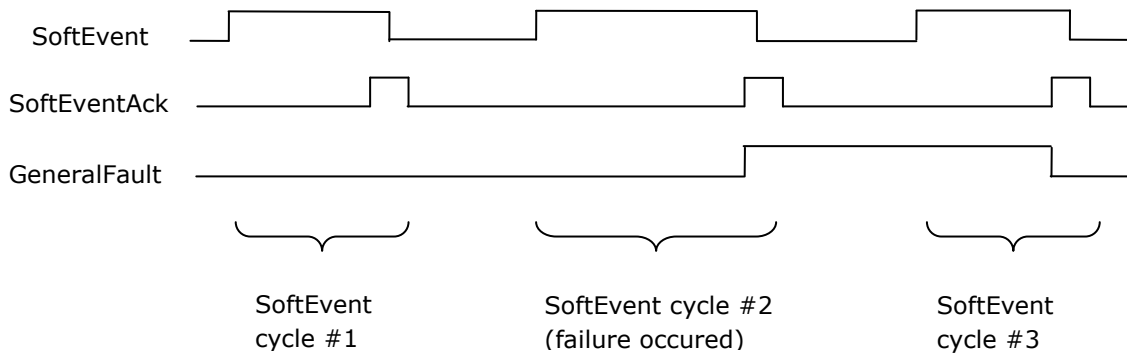
The DataMan MC Protocol implementation includes a String Command feature. This feature allows you to execute string-based DMCC commands. The DMCC command is sent to the reader via the String Command block. The DMCC command result is returned via the String Command Result block. Initiating a command and notification of completion is accomplished by signaling bits in the Control and Status blocks.

To execute a DMCC command, the command string is placed in the data field of the String Command block. The command string consists of standard ASCII text. The command format is exactly the same as would be used for a serial (RS-232) or Telnet connection. The string does not need to be terminated with a null character. Instead, the length of the string (that is, the number of ASCII characters) is placed in the length field of the String Command block.

After executing the DMCC command, the result string is returned in the String Command Result block. Similar to the original command, the result string consists of ASCII characters in the same format as would be returned via serial or Telnet. Also, there is no terminating null character. Instead, the length of the result is returned in the Command String Result length field. The Command String Result block also contains a numeric result code. This allows you to determine the success or failure of the command without having to parse the text string. The values of the result code are defined in the DMCC documentation.

General Fault Indicator

When a communication-related fault occurs, the "GeneralFault" bit will change from 0 → 1. Currently the only fault conditions supported are soft event operations. If a soft event operation fails, the fault bit will be set. The fault bit will remain set until the next successful soft event operation, or, until *TriggerEnable* is set to 0 and then back to 1.



Examples

Included with the Setup Tool installer are two example PLC programs created with CoDeSys v2.3 software. These samples are designed and tested on a Wago 750-841 PLC. These simple programs clearly demonstrate DataMan ID readers' capabilities and proper operation. The same operations can be achieved by using more advanced features and efficient programming practices with Wago PLCs.

However, such an advanced program is less useful for demonstration purposes. The examples try to show different approaches in the techniques used for the communication to the DataMan reader.

NOTE

All examples are designed to work only if the "Control" datablock is mapped to the Coil space and the "Status" datablock is mapped to the Discrete Input space.

ApplicationLayer Example

This sample realizes a generic data transfer between the DataMan reader and the PLC. Memory areas of the "Control", "Status" and "Output Area" are cloned in the PLC and synchronized as needed or cyclically. Each data area is synchronized with its own instance of "ETHERNETMODBUSMASTER_TCP". This causes 3 TCP connections to be open simultaneously. Make sure that the Modbus TCP related setting "Maximum Connections" on the DataMan reader is set to at least 3 for this example to work.

Function

The example application demonstrates the following operations:

1. Transfer the 32-bit "Control" register data from the PLC to the reader.
2. Transfer the 32-bit "Status" register data from the reader to the PLC.
3. Transfer "Output Data" from the reader to the PLC.

All actions are started when there is a connection to the reader.

Transferring "Control" Register Data

All data gets transferred when there is a change in the local PLC data. The local PLC data can be manipulated in the visualization.

NOTE

No synchronization is implemented from the reader to the PLC, so the local PLC data might be incorrect after building up the connection or if another Modbus TCP client manipulates the Control register simultaneously. There is a timeout setting that can lead to a disconnect if you do not manipulate the "Control" register during this timeframe.

Transferring Status Register Data

All data gets transferred cyclically. The poll interval can be specified in the visualization.

Transferring Output Data

All data gets transferred cyclically. The poll interval can be specified in the visualization.

DataManControl Example

This sample shows in a sequential manner the steps to do to achieve one of the functions named in the following subsection. To outline this chronological sequence "Sequential Function Chart" was chosen as programming language.

Function

The example application demonstrates the following operations:

1. Triggering a read
2. Getting read results
3. Executing string commands (DMCC)
4. Executing soft event operations
 - a. Train code
 - b. Train match string
 - c. Train focus
 - d. Train brightness
 - e. Untrain
 - f. Execute DMCC
 - g. Set match string

The "Main" program contains variables to invoke each of these operations. The operation is invoked by toggling the control bool directly or from the visualization (red=0, green=1) from 0 → 1. This will invoke the associated subroutine to perform the operation. When the operation 4 is complete, the subroutine will set the control bit back to 0.

Triggering a Read

The example provides a "Continuous Trigger". As the name implies, enabling the "xTrigger" bit will invoke a continuous series of read operations. Once enabled, the "xTrigger" control bit will remain set until you disable it.

Primarily, the trigger subroutine manages the trigger handshake operation between the PLC and the reader. The control *Result Ack* and *Trigger* bits are reset, the *Trigger Enable* bit is set, the PLC waits for the corresponding *TriggerReady* status bit from the reader, and the control *Trigger* bit is set. Refer to a description of handshaking in Section [Operation](#).

Getting Read Results

For this example the operation of triggering a read and getting read results was intentionally separated. This is to support the situation where the PLC is not the source of the read trigger. For example, the reader may be configured to use a hardware trigger. In such a case, only the get results subroutine would be needed.

Like the triggering subroutine, the get results subroutine manages the results handshake operation between the PLC and the reader. The routine waits for the *ResultsAvailable* status bit to become active, it copies the result data to internal storage, and then executes the *ResultsAck* handshake. Refer to a description of handshaking in Section [Operation](#).

The read result consists of a *ResultCode*, *ResultLength*, and *ResultData*. Refer to Section [Output Data Block Field Descriptions](#) for details of the *ResultCode* values. The *ResultLength* field indicates how many bytes of actual result data exist in the *ResultData* field. The subroutine converts this byte length to word length before copying the results to internal storage.

Execute String Commands (DMCC)

The string command feature provides a simple way to invoke DMCC commands from the PLC. The command format and command result format is exactly identical to that used for serial or Telnet DMCC operation.

This subroutine copies an example DMCC command (`||>GET DEVICE.TYPE`) to the String Command block and then manages the string command handshake operation between the PLC and the reader to invoke the command and retrieve the command result. Any valid DMCC command may be invoked with this mechanism. Refer to the *DataMan Command Reference* document available through the Windows Start menu.

Execute Soft Events

Soft Events are used to invoke a predefined action. Each Soft Event is essentially a virtual input signal. Each of the soft event subroutines manages the handshake operation between the PLC and the reader to invoke the predefined action. The associated action is invoked when the *SoftEvent* bit toggles from 0 → 1. The subroutine then watches for the associated *SoftEventAck* bit from the reader which signals that the action is complete. For a description of handshaking, see Section [Operation](#).

NOTE

The "Execute DMCC" and "Set Match String" soft events make use of the Input Data block. The subroutine for these two events copies the relevant data into the User Data fields of the Input Data block and then invokes the User Data subroutine to transfer the data to the reader. Only after the user data is transferred is the actual soft event action invoked. It is required that the user data be transferred before invoking either of these events.

NOTE

The "Train Match String" soft event only prepares the training mechanism. The actual training occurs on the next read operation. Therefore, a trigger must be issued following "Train Match String".

PROFINET

PROFINET is an application-level protocol used in industrial automation applications. This protocol uses standard Ethernet hardware and software to exchange I/O data, alarms, and diagnostics.

DataMan supports PROFINET I/O. This is one of the 2 “views” contained in the PROFINET communication standard. PROFINET I/O performs cyclic data transfers to exchange data with Programmable Logic Controllers (PLCs) over Ethernet. The second “view” in the standard, PROFINET CBA (Component Based Automation), is not supported.

A deliberate effort has been made to make the DataMan PROFINET communication model closely match the Cognex In-Sight family. Customers with In-Sight experience should find working with DataMan familiar and comfortable.

By default, the DataMan has the PROFINET protocol disabled. The protocol can be enabled via DMCC, scanning a parameter code or in the Setup Tool.

DMCC

The following commands can be used to enable/disable PROFINET. The commands can be issued via RS-232 or Telnet connection.

NOTE

Because you have to make changes to the Telnet client provided by Windows to communicate with DataMan, it is recommended you use third party clients such as PuTTY.

Enable:

```
||>SET PROFINET.ENABLED ON  
||>CONFIG.SAVE  
||>REBOOT
```

Disable:

```
||>SET PROFINET.ENABLED OFF  
||>CONFIG.SAVE  
||>REBOOT
```

Reader Configuration Code

Scanning the following reader configuration codes will enable/disable PROFINET.

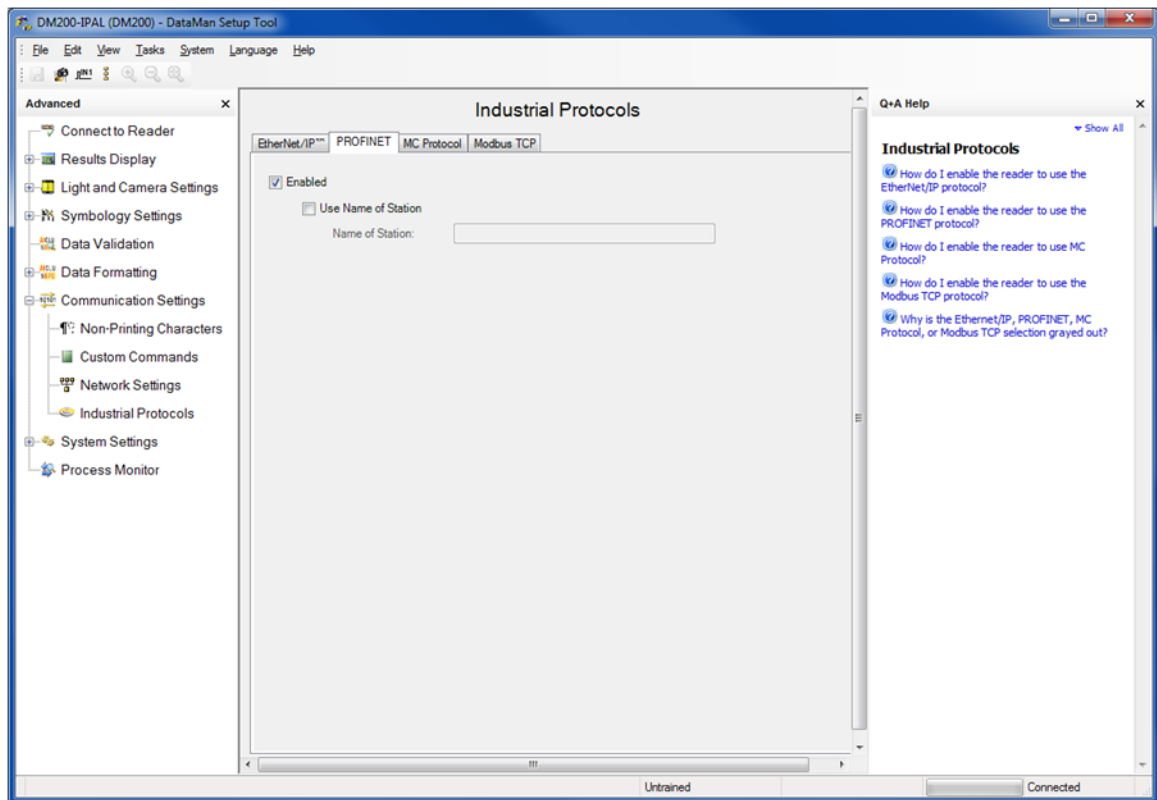
NOTE

You must reboot the device for the change to take effect.



Setup Tool

The PROFINET protocol can be enabled by checking **Enabled** on the Industrial Protocols pane's PROFINET tab.



Make sure you save the new selection by clicking Yes to the Reboot Required message window.

NOTE

You must reboot your reader for the new settings to take effect.

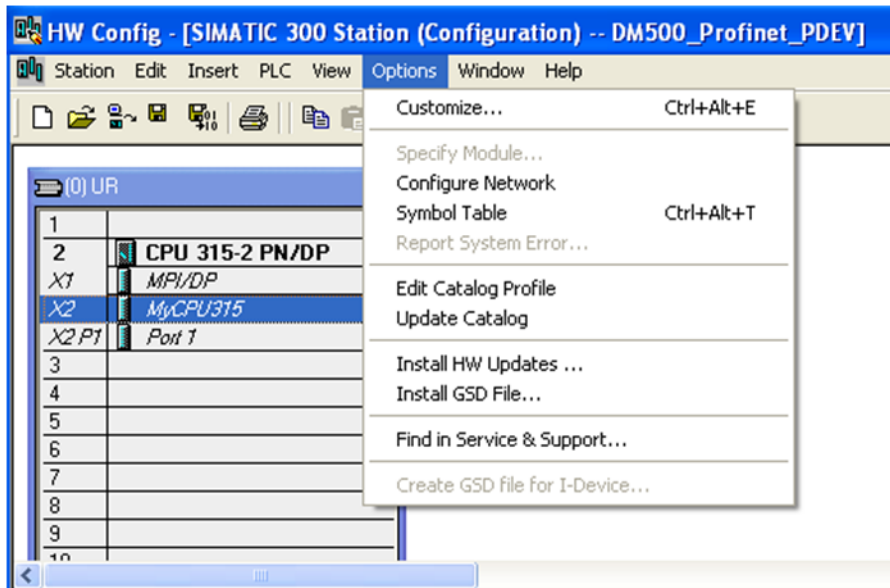
Getting Started

Preparing to use PROFINET involves the following main steps:

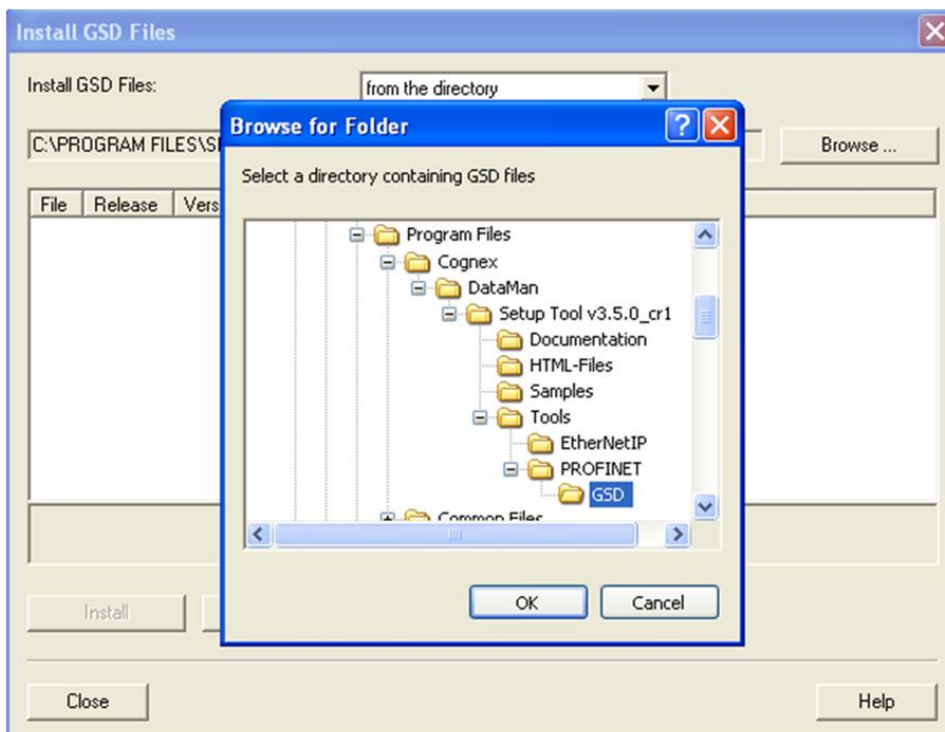
- Make sure you have the Siemens Step 7 programming software (SIMATIC) installed.
- Set up the Siemens Software tool so that it recognizes your DataMan device.
Install the Generic Station Description (GSD) file.

Perform the following steps to set up PROFINET:

1. Verify that SIMATIC is on your machine.
2. From the Windows Start menu, launch the SIMATIC Manager.
3. If you already have a project, select "Cancel" to skip past the New Project wizard. Otherwise, let the wizard guide you through creating a new project.
4. Once the Manager has opened the project, double-click on the "Hardware" icon to open the "HW Config" dialog screen. From the main menu, select "Options→Install GSD File...".



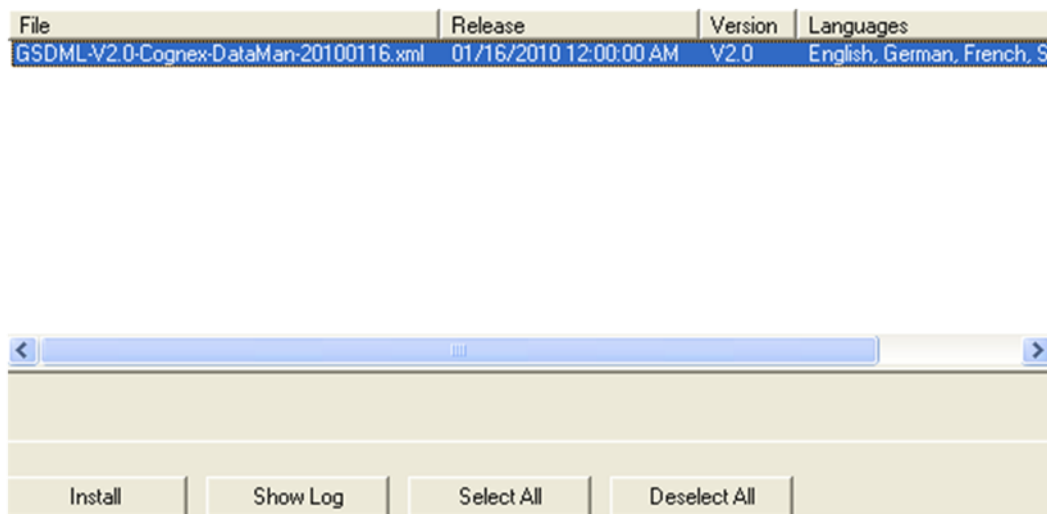
5. Browse to the location where the GSD file was installed (or the location where you saved the GSD file if it was downloaded from the web).



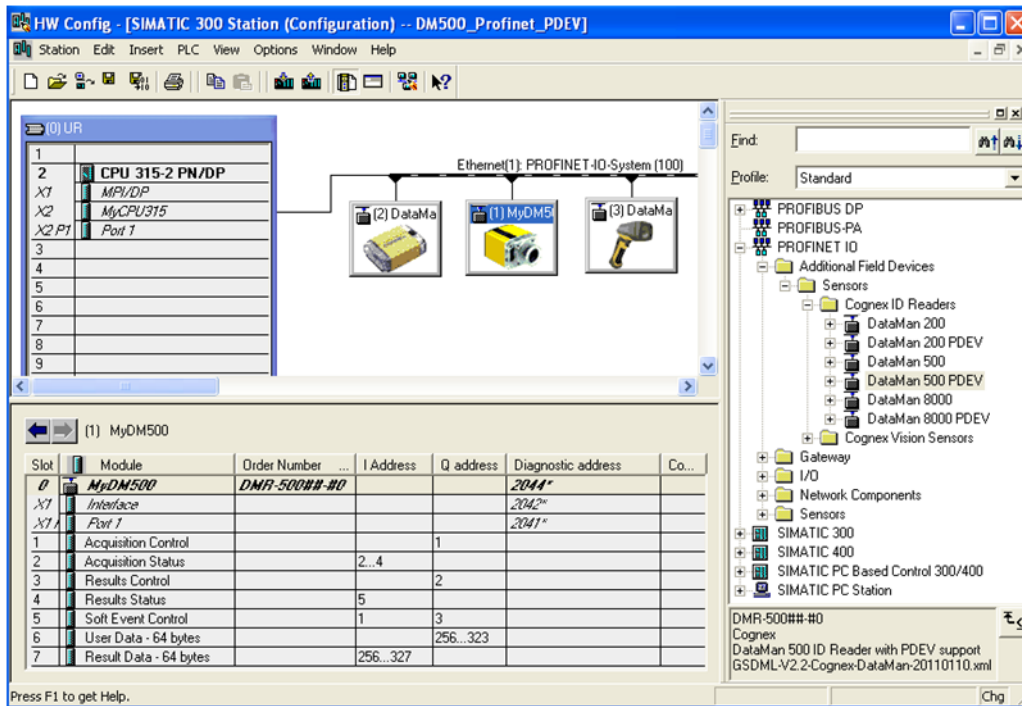
- Select the GSD file you wish to install and follow the displayed instructions to complete the installation.

NOTE

There may be more than one GSD file in the list. If you are unsure which to install, choose the one with the most recent date.



- Add your DataMan device to your project. This makes the DataMan available in the Hardware Catalog. Launch the SIMATIC Hardware Config tool.
- In the main menu, select View → Catalog.
- The catalog is displayed. Expand the "PROFINET IO" tree to the "Cognex ID Readers" node.
- With the left mouse button, drag the DataMan reader over and drop it on the PROFINET IO network symbol in the left pane.

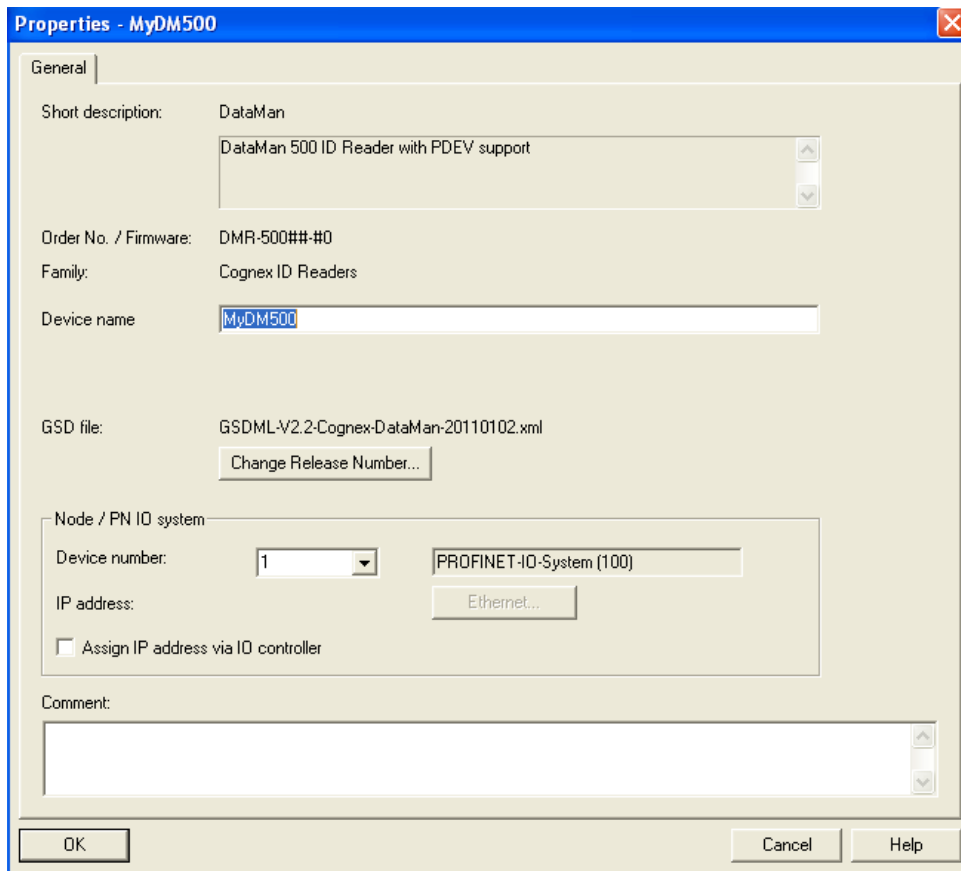


The HW Config tool automatically maps the DataMan I/O modules into the memory space.

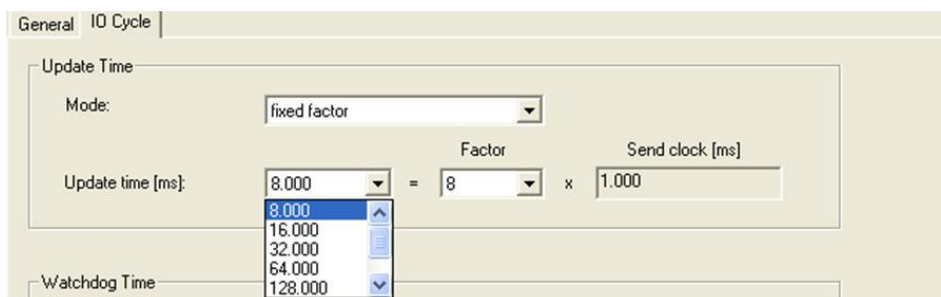
NOTE

By default, the 64 byte User Data and 64 byte Result Data Modules are inserted. There are multiple sizes available for both of these modules. To optimize performance use the module size that most closely matches the actual data requirements of your application. You can change the module simply by deleting the one in the table and inserting the appropriate sized module from the catalog.

11. Right-click on the DataMan icon and select "Object Properties...".
12. Give the reader a name. This must match the name of your actual DataMan reader. The name must be unique and conform to DNS naming conventions. Refer to the SIMATIC Software help for details.
13. If your DataMan reader is configured to use its own static IP, uncheck the "Assign IP address via IO controller" box. Otherwise if you wish the PLC to assign an IP address, select the Ethernet button and configure the appropriate address.

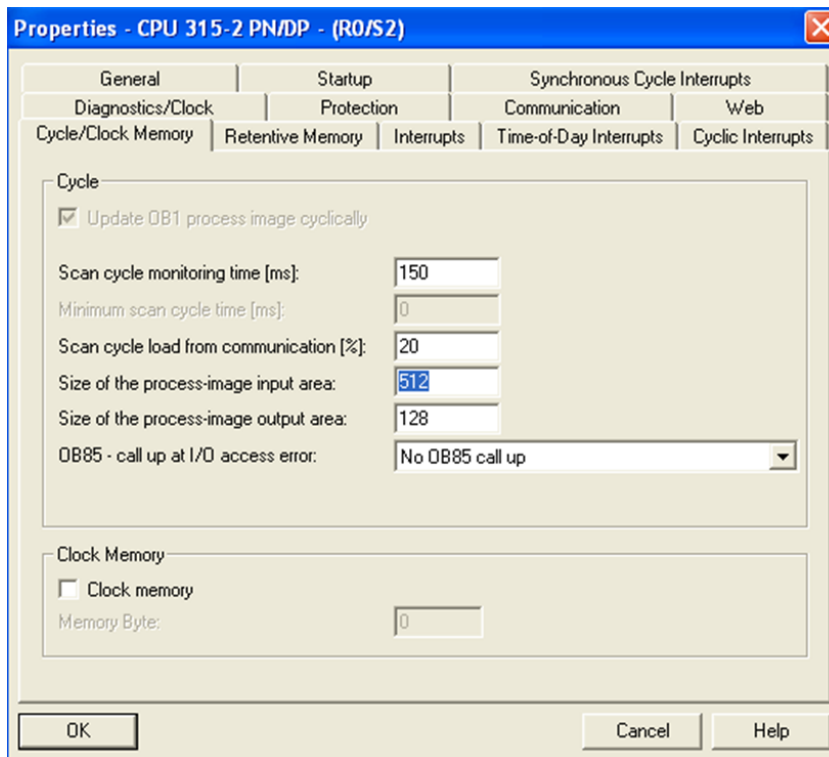


14. In the "IO Cycle" tab, select the appropriate cyclic update rate for your application.



15. By default, the SIMATIC software maps the User Data & Result Data Modules to offset 256. This is outside of the default process image area size of 128. That is, by default, data in these modules are inaccessible by some SFCs such as BLKMOV. As a solution, either remap the modules to lower offsets within the process image area or expand the process image area to include these modules.

If you choose to expand the process image area, make the size large enough for the module size plus the default 256 offset.



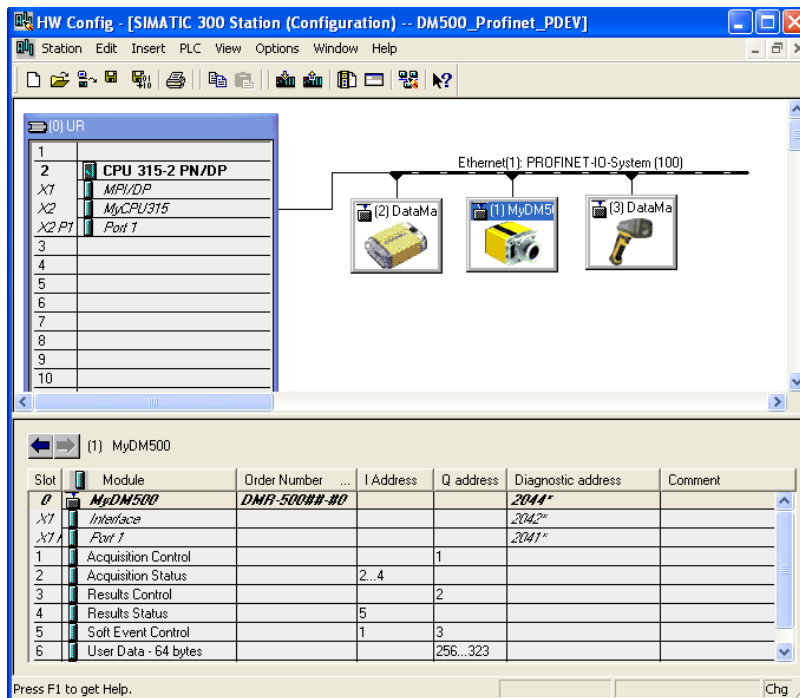
NOTE

Expanding the process image can have a performance impact on the PLC scan cycle time. If your scan time is critical, use the minimal acceptable module sizes and manually remap them down lower in the process image.

Modules

The PROFINET implementation on DataMan consists of seven I/O modules.

1. Acquisition Control Module
2. Acquisition Status Module
3. Results Control Module
4. Results Status Module
5. Soft Event Control Module
6. User Data Module
7. Result Data Module



Acquisition Control Module

Controls image acquisition. This module consists of data sent from the PLC to the DataMan device.

Slot number: 1

Total Module size: 1 byte

Bit	Name	Description
0	Trigger Enable	Setting this bit enables triggering via PROFINET. Clearing this bit disables triggering.
1	Trigger	Setting this bit triggers an acquisition when the following conditions are met: <ul style="list-style-type: none"> • Trigger Enable is set • No acquisition is currently in progress • The device is ready to trigger
2 - 7	Reserved	Reserved for future use

Acquisition Status Module

Indicates the current acquisition status. This module consists of data sent from the DataMan device to the PLC.

Slot number: 2

Total Module size: 3 bytes

Bit	Name	Description
0	Trigger Ready	Indicates when the device is ready to accept a new trigger. Bit is True when "Trigger Enable" has been set and the device is ready to accept a new trigger.
1	Trigger Ack	Indicates that the DataMan has received a new Trigger. This bit will remain True as long as the "Trigger" bit remains True (that is, it is interlocked with the Trigger bit).
2	Acquiring	Indicates that the DataMan is currently acquiring an image.
3	Missed Ack	Indicates that the DataMan was unable to successfully trigger an acquisition. Bit is cleared when the next successful acquisition occurs.
4 – 7	Reserved	Reserved for future use
8–23	Trigger ID	ID value of the next trigger to be issued (16-bit integer). Used to match issued triggers with corresponding result data received later. This same value will be returned in ResultID of the result data.

Results Control Module

Controls the processing of result data. This module consists of data is sent from the PLC to the DataMan device.

Slot number: 3

Total Module size: 1 byte

Bit	Name	Description
0	Results Buffer Enable	Enables queuing of "Result Data". If enabled, the current result data will remain until acknowledged (even if new results arrive). New results are queued. The next set of results are pulled from the queue (made available in the Result Data module) each time the current results are acknowledged. The DataMan will respond to the acknowledge by clearing the "Results Available" bit. Once the "Results Ack" bit is cleared the next set of read results will be posted and "Results Available" will be set True. If results buffering is not enabled newly received read results will simply overwrite the content of the Result Data module.

Bit	Name	Description
1	Results Ack	Bit is used to acknowledge that the PLC has successfully read the latest result data. When set True the "Result Available" bit will be cleared. If result buffering is enabled, the next set of result data will be pulled from the queue and "Result Available" will again be set True.
2 – 7	Reserved	Reserved for future use

Results Status Module

Indicates the acquisition and result status. This module consists of data sent from the DataMan device to the PLC.

Slot number: 4

Total Module size: 1 byte

Bit	Name	Description
0	Decoding	Indicates that the DataMan is decoding an acquired image.
1	Decode Complete	Bit is toggled on the completion of a decode operation when the new results are made available (0→1 or 1→0).
2	Result Buffer Overrun	Indicates that the DataMan has discarded a set of read results because the results queue is full. Cleared when the next set of results are successfully queued.
3	Results Available	Indicates that a new set of read results are available (i.e. the contents of the Result Data module are valid). Cleared when the results are acknowledged.
4 – 6	Reserved	Reserved for future use
7	General Fault	Indicates that a fault has occurred (i.e. Soft Event "Set Match String" or "Execute DMCC" error has occurred).

Soft Event Control Module

Used to initiate a Soft Event and receive acknowledgment of completion. Note, this is a bi-directional I/O module. Module data sent from the PLC initiates the Soft Event. Module data sent by the DataMan device acknowledges completion.

Slot number: 5

Total Module size: 1 byte (input) and 1 byte (output)

Data written from the PLC to DataMan:

Bit	Name	Description
0	Train Code	Bit transition from 0→1 will cause the train code operation to be invoked.
1	Train Match String	Bit transition from 0→1 will cause the train match string operation to be invoked.
2	Train Focus	Bit transition from 0→1 will cause the train focus operation to be invoked.

Bit	Name	Description
3	Train Brightness	Bit transition from 0→1 will cause the train brightness operation to be invoked.
4	Untrain	Bit transition from 0→1 will cause the untrain operation to be invoked.
5	Reserved	Reserved for future use
6	Execute DMCC	Bit transition from 0→1 will cause the DMCC operation to be invoked. Note that a valid DMCC command string must first be placed in "User Data" before invoking this event.
7	Set Match String	Bit transition from 0→1 will cause the set match string operation to be invoked. Note that match string data must first be placed in "User Data" before invoking this event.

Data written from the DataMan to PLC:

Bit	Name	Description
0	Train Code Ack	Indicates that the "Train Code" operation has completed
1	Train Match String Ack	Indicates that the "Train Match String" operation has completed
2	Train Focus Ack	Indicates that the "Train Focus" operation has completed
3	Train Brightness Ack	Indicates that the "Train Brightness" operation has completed
4	Untrain Ack	Indicates that the "Untrain" operation has completed
5	Reserved	Reserved for future use
6	Execute DMCC Ack	Indicates that the "Execute DMCC" operation has completed
7	Set Match String Ack	Indicates that the "Set Match String" operation has completed

User Data Module

Data sent from a PLC to a DataMan to support acquisition, decode and other special operations. Currently this module is only used to support the "Execute DMCC" and "Set Match String" soft events.

Note, there are actually 5 versions of the User Data module. Only one instance can be configured for use in a given application. The "User Data Option" and "User Data Length" fields are the same for each module. The "User Data" field varies in size based on the selected module. Choose the module which is large enough to exchange the amount of data required by your application.

Slot number: 6

Total Module size: 4 + 16 (16 bytes of User Data)
 4 + 32 (32 bytes of User Data)
 4 + 64 (64 bytes of User Data)

4 + 128 (128 bytes of User Data)

4 + 250 (250 bytes of User Data)

Byte	Name	Description
0 - 1	User Data Option	Currently only used by "Set Match String" soft event. Specifies which code target to assign the string (16-bit Integer). 0, assign string to all targets 1, assign string to 2D codes 2, assign string to QR codes 3, assign string to 1D / stacked / postal codes
2 - 3	User Data Length	Number of bytes of valid data actually contained in the "User Data" field (16-bit Integer).
4 ...	User Data	Data sent from the PLC to the DataMan to support acquisition, decode and other special operations (array of bytes).

Result Data Module

Read result data sent from a DataMan to a PLC.

NOTE

There are actually 5 versions of the Result Data module. Only a single instance can be configured for use in a given application. The "Result ID", "Result Code", "Result Extended" and "Result Length" fields are the same for each module. The "Result Data" field varies in size based on the selected module. Choose the module which is large enough to exchange the amount of result data required by your application.

Slot number: 7

- Total Module size: 8 + 16 (16 bytes of Result Data)
- 8 + 32 (32 bytes of Result Data)
- 8 + 64 (64 bytes of Result Data)
- 8 + 128 (128 bytes of Result Data)
- 8 + 246 (246 bytes of Result Data)

Byte	Name	Description
0 - 1	Result ID	The value of the "Trigger ID" when the trigger that generated these results was issued. Used to match up triggers with corresponding result data (16-bit Integer).

Byte	Name	Description
2 - 3	Result Code	Indicates the success or failure of the read that produced these results (16-bit Integer). Bit 0,1=read, 0=no read Bit 1,1=validated, 0=not validated (or validation not in use) Bit 2,1=verified, 0=not verified (or verification not in use) Bit 3,1=acquisition trigger overrun Bit 4,1=acquisition buffer overflow Bits 5-15 reserved
4 - 5	Result Extended	Currently unused (16-bit Integer).
6 - 7	Result Length	Actual number of bytes of read data contained in the "Result Data" field (16-bit Integer).
8 ...	Result Data	Decoded read result data (array of bytes)

Operation

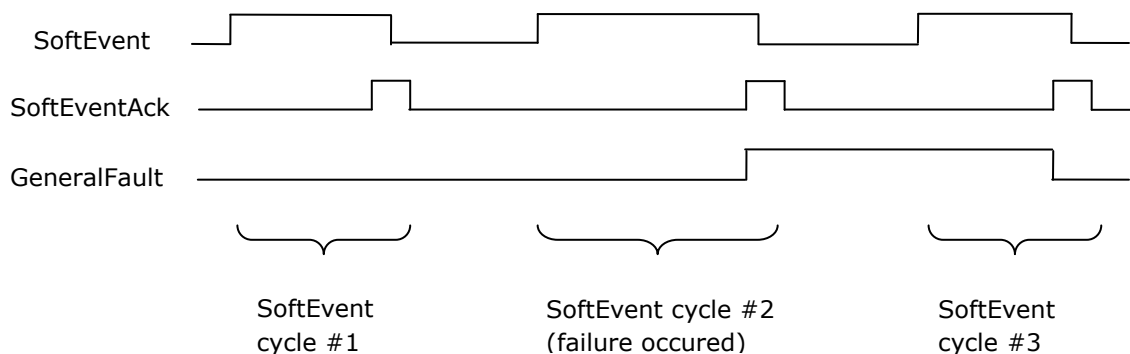
SoftEvents

SoftEvents act as "virtual" inputs. When the value of a SoftEvent changes from 0 → 1 the action associated with the event will be executed. When the action completes the corresponding SoftEventAck bit will change from 0 → 1 to signal completion. The acknowledge bit will change back to 0 when the corresponding SoftEvent bit is set back to 0.

The "ExecuteDMCC" and "SetMatchString" soft event actions require user supplied data. This data must be written to the UserData & UserDataLength area of the UserData Module prior to invoking the soft event. Since both of these soft events depend on the UserData, only one may be invoked at a time.

General Fault Indicator

When a communication related fault occurs the "GeneralFault" bit will change from 0 → 1. Currently the only fault conditions supported are soft event operations. If a soft event operation fails, the fault bit will be set. The fault bit will remain set until the next soft event operation or until triggering is disabled and again re-enabled.



Acquisition Sequence

DataMan can be triggered to acquire images by several methods. It can be done explicitly by manipulating the Trigger bit of the Acquisition Control Module, it can be triggered by external hard wired input, and finally it can be triggered via DMCC command. Manipulating the Acquisition Control Module bits will be discussed here.

On startup the "Trigger Enable" bit will be False. It must be set to True to enable triggering. When the device is ready to accept triggers, the "Trigger Ready" bit will be set to True.

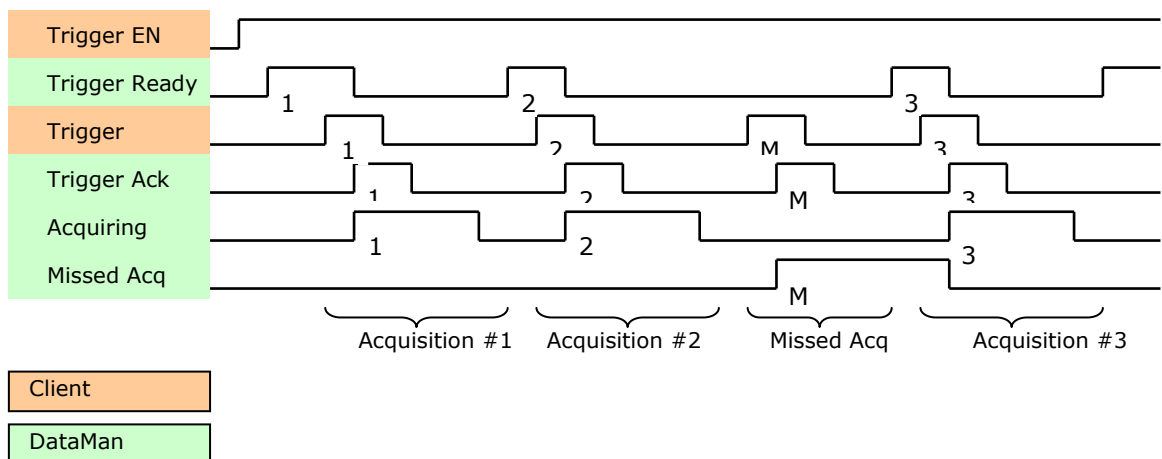
While the Trigger Ready bit is True, each time the reader sees the "Trigger" bit change from 0 to 1, it will initiate an image acquisition. The client (PLC) should hold the bit in the new state until that same state value is seen back in the Trigger Ack bit (this is a necessary handshake to guarantee that the change is seen by the reader).

During an acquisition, the Trigger Ready bit will be cleared and the Acquiring bit will be set to True. When the acquisition is completed, the Acquiring bit will be cleared. The Trigger Ready bit will again be set True once the device is ready to begin a new image acquisition.

If results buffering is enabled, the device will allow overlapped acquisition and decoding operations. Trigger Ready will be set high after acquisition is complete but while decoding is still in process. This can be used to achieve faster overall trigger rates. If result buffering is not enabled, the Trigger Ready bit will remain low until both the acquisition and decode operations have completed.

To force a reset of the trigger mechanism set the Trigger Enable bit to False, until the Trigger Ready bit is 0. Then, Trigger Enable can be set to True to re-enable acquisition.

As a special case, an acquisition can be cancelled by clearing the Trigger signal before the read operation has completed. This allows for the cancellation of reads in Presentation and Manual mode if no code is in the field of view. To ensure that a read is not unintentionally cancelled, it is advised that the PLC hold the Trigger signal True until both TriggerAck and ResultsAvailable are True (or DecodeComplete toggles state).



Decode / Result Sequence

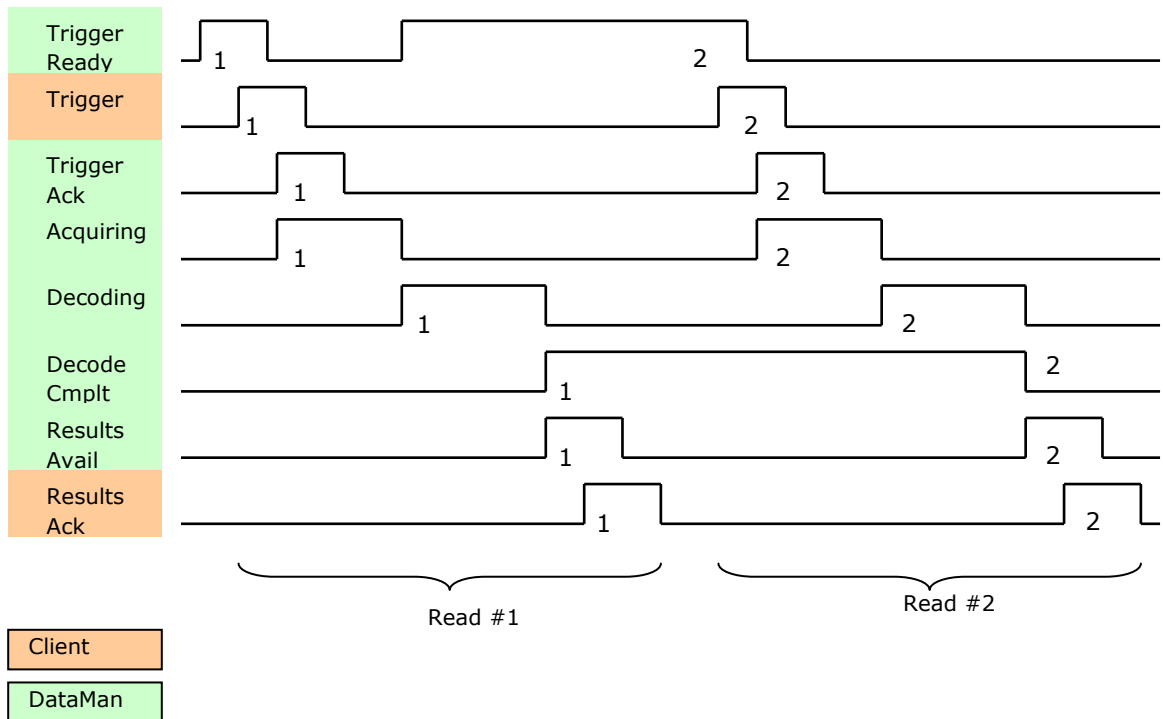
After an image is acquired it is decoded. While being decoded, the “Decoding” bit of the Result Status Module is set. When decode is complete, the Decoding bit is cleared and the “Decode Complete” bit is toggled.

The “Results Buffer Enable” bit determines how decode results are handled by the reader. If the Results Buffer Enable bit is set to False, then the decode results are immediately placed into the Results Module and Results Available is set to True.

If the Results Buffer Enable bit is set to True the new results are queued. The earlier decode results remain in the Results Module until they are acknowledged by the client by setting the “Results Ack” bit to True. After the Results Available bit is cleared, the client should set the Results Ack bit back to False to allow the next queued results to be placed in to the Results Module. This is a necessary handshake to ensure the results are received by the DataMan client (PLC).

Behavior of DecodeStatusRegister

Bit	Bit Name	Results if Buffering Disabled	Results if Buffering Enabled
1	Decoding	Set when decoding an image.	Set when decoding an image.
2	Decode Complete	Toggled on completion of an image decode.	Toggled on completion of an image decode.
3	Results Buffer Overflow	Remains set to zero.	Set when decode results could not be queued because the client failed to acknowledge a previous result. Cleared when the decode result is successfully queued.
4	Results Available	Set when new results are placed in the Results Module. Stays set until the results are acknowledged by setting Results Ack to true.	Set when new results are placed in the Results Module. Stays set until the results are acknowledged by setting Results Ack to true.



Results Buffering

There is an option to enable a queue for decode results. If enabled this allows a finite number of decode result data to queue up until the client (PLC) has time to read them. This is useful to smooth out data flow if the client (PLC) slows down for short periods of time or if there are surges of read activity.

Also, if result buffering is enabled the device will allow overlapped acquisition and decode operations. Depending on the application this can be used to achieve faster over all trigger rates. See Acquisition Sequence description above for further detail.

In general, if reads are occurring faster than results can be sent out, the primary difference between buffering or not buffering is determining which results get discarded. If buffering is not enabled the most recent results are kept and the earlier result (which was not read by the PLC fast enough) is lost. Essentially the more recent result will simply over write the earlier result. If buffering is enabled (and the queue becomes full) the most recent results are discarded until room becomes available in the results queue.

Siemens Examples

This section gives some examples of using the DataMan with a Siemens S7-300 PLC. It is assumed that the reader is familiar with the S7-300 and the SIMATIC programming software.

Symbol Table

Although not required, defining symbols for the DataMan I/O module elements can be extremely helpful. It makes the code much easier to read and reduces mistakes. This sample table shows symbols defined for a typical instance of a DataMan reader. Note,

DataMan I/O modules may be at different addresses in your project. Make sure to adjust your symbol definitions based on the specific offsets of the I/O modules.

Status	Symbol	Address	Data type	Comment
1	Reader_TriggerEnable	Q 1.0	BOOL	
2	Reader_Trigger	Q 1.1	BOOL	
3	Reader_TriggerReady	I 2.0	BOOL	
4	Reader_TriggerAck	I 2.1	BOOL	
5	Reader_Acquiring	I 2.2	BOOL	
6	Reader_MissedAck	I 2.3	BOOL	
7	Reader_TriggerID	IV 3	WORD	
8	Reader_BufferEnable	Q 2.0	BOOL	
9	Reader_ResultsAck	Q 2.1	BOOL	
10	Reader_Decoding	I 5.0	BOOL	
11	Reader_DecodeComplete	I 5.1	BOOL	
12	Reader_ResultsOverrun	I 5.2	BOOL	
13	Reader_ResultsAvailable	I 5.3	BOOL	
14	Reader_GeneralFault	I 5.7	BOOL	
15	Reader_TrainCode	Q 3.0	BOOL	
16	Reader_TrainMatchString	Q 3.1	BOOL	
17	Reader_TrainFocus	Q 3.2	BOOL	
18	Reader_TrainBrightness	Q 3.3	BOOL	
19	Reader_UnTrain	Q 3.4	BOOL	
20	Reader_ExecuteDmcc	Q 3.6	BOOL	
21	Reader_SetMatchString	Q 3.7	BOOL	
22	Reader_TrainCodeAck	I 1.0	BOOL	
23	Reader_TrainMatchStrAck	I 1.1	BOOL	
24	Reader_TrainFocusAck	I 1.2	BOOL	
25	Reader_TrainBrightAck	I 1.3	BOOL	
26	Reader_UnTrainAck	I 1.4	BOOL	
27	Reader_ExecuteDmccAck	I 1.6	BOOL	
28	Reader_SetMatchStringAck	I 1.7	BOOL	
29	Reader_UserDataOption	QW 256	WORD	
30	Reader_UserDataLength	QW 258	WORD	
31	Reader_ResultID	IV 256	WORD	
32	Reader_ResultCode	IV 258	WORD	
33	Reader_ResultExtended	IV 260	WORD	
34	Reader_Resultlength	IV 262	WORD	

Press F1 to get Help. NUM

Trigger and Get Results

Run the sample program "DM200_SampleRead" for the complete example program.

NOTE

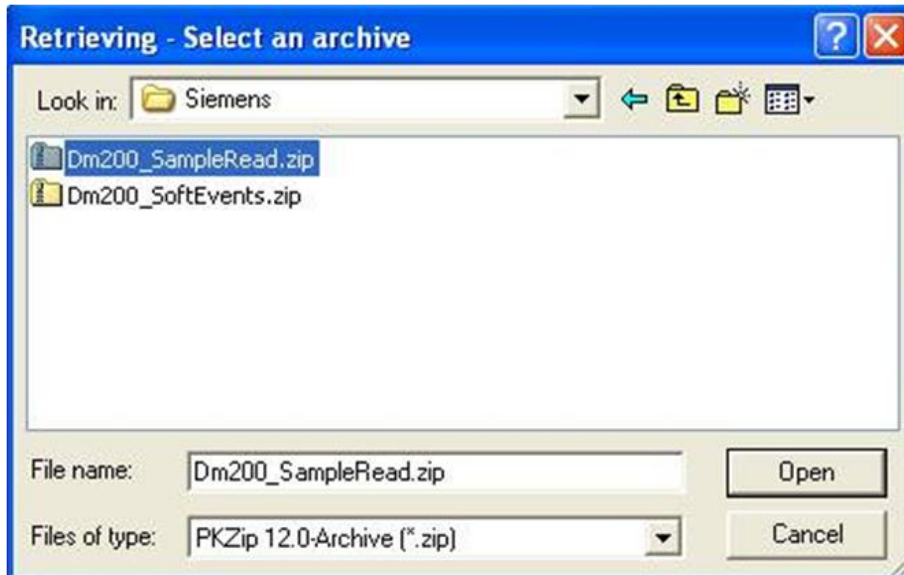
This sample can be used with any PROFINET enabled DataMan reader.

Perform the following steps to install the program:

1. Start the SIMATIC Manager software.
2. Close any open applications.
3. From the main menu, select File → Retrieve...
4. Browse to find the sample file on your PC.



5. Look for the Siemens folder and select the zip file.



6. Select a destination directory to save the project on your PC.



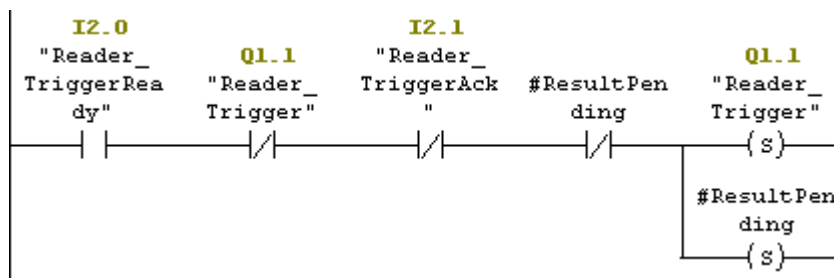
7. The Siemens software extracts the sample archive and makes it available.
8. Reduced to the basics the process of reading and retrieving results consists of the following:
9. Define an area in your application to save read results. There are many options regarding how and where result data can be stored. For our example we define a Data Block (DB) which contains the fields of the Result Data module that we are interested in for our application.

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	ID	INT	0	ID of this read result
+2.0	Flags	INT	0	Flags indicating success or failure
+4.0	Length	INT	0	Number of data bytes in the array Value[]
+6.0	Value	ARRAY[1..63]		Code value read
+1.0		CHAR		
=70.0		END_STRUCT		

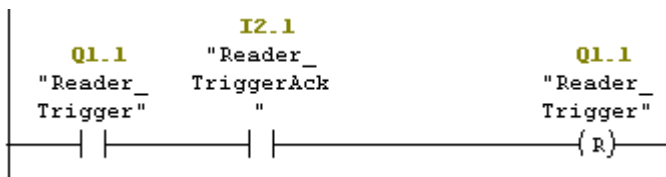
10. Enable the reader



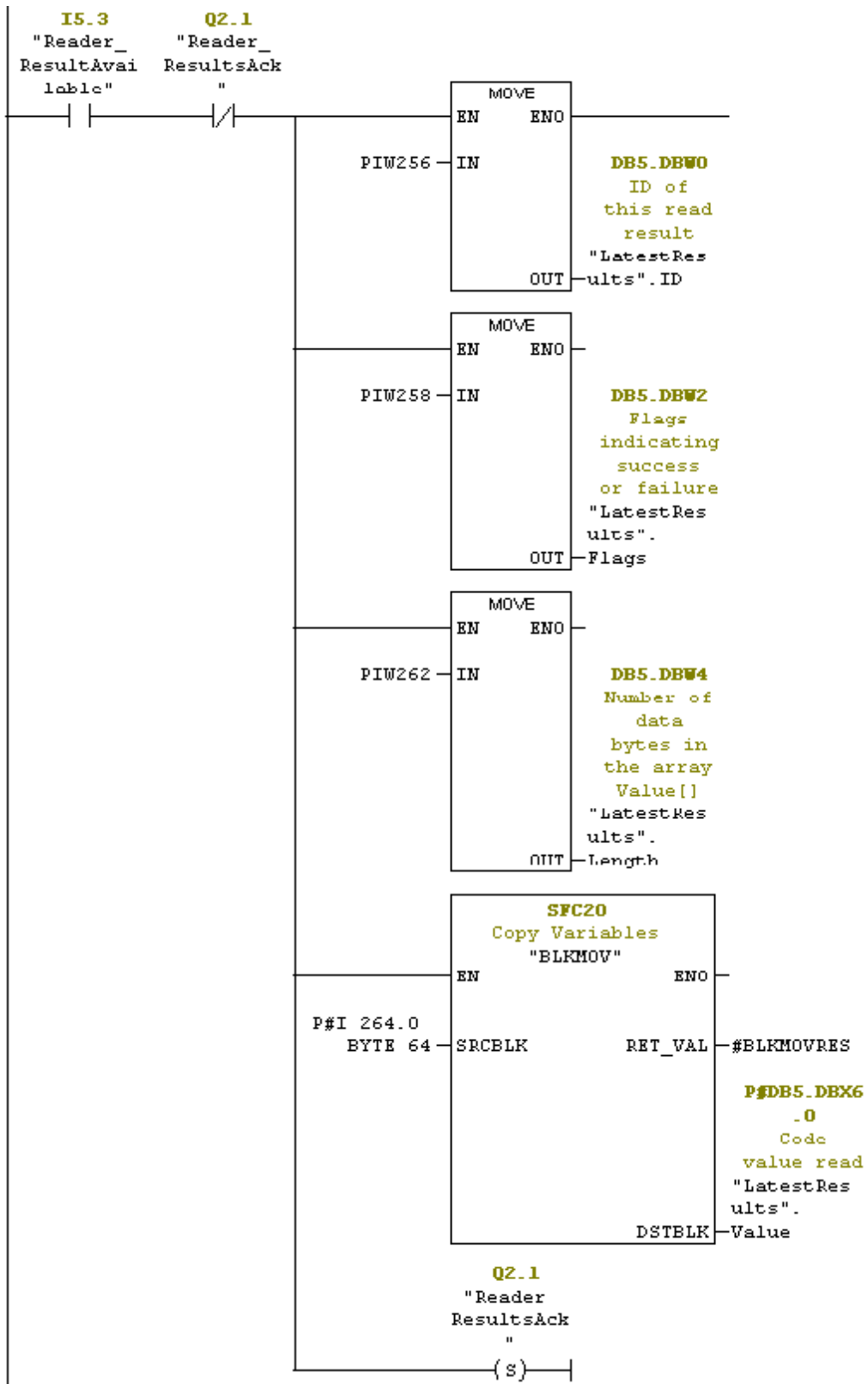
11. Set the trigger signal and set semaphore to indicate a read is pending.



12. As soon as the trigger signal is acknowledged, clear the trigger signal.



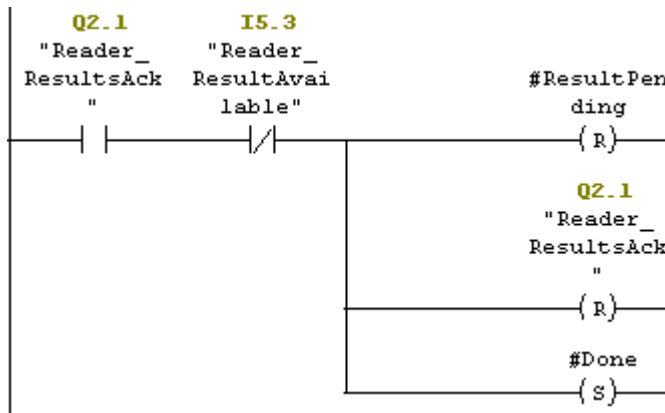
13. As soon as the results are available save a copy of the result data and set the results acknowledge signal.



14. When the reader sees the result acknowledge signal clear result acknowledge, clear the read pending semaphore, and signal that the read process has completed.

NOTE

The reader clears "Results Available" as soon as it sees the PLC's "Results Ack" signal.



Using Soft Events

Run the sample program "DM200_SoftEvents" for the complete example program.

NOTE

This sample can be used with any PROFINET enabled DataMan reader.

Perform the following steps to install the program:

1. Start the SIMATIC Manager software.
2. Close any open applications.
3. From the main menu, select File → Retrieve...
4. Browse to find the sample file on your PC.



5. Look for the Siemens folder and select Dm200_SoftEvents.zip.
6. Select a destination directory to save the project on your PC.



7. The Siemens software extracts the sample archive and makes it available.

Soft events are a means of invoking an activity by simply manipulating a single control bit. The activity for each bit is predefined (for more details, see Section [SoftEvents](#)). With the exception of "Execute DMCC" and "Set Match String" all soft events may be invoked in the same way. "Execute DMCC" and "Set Match String" require the added step of loading the User Data module with application data before invoking the event.

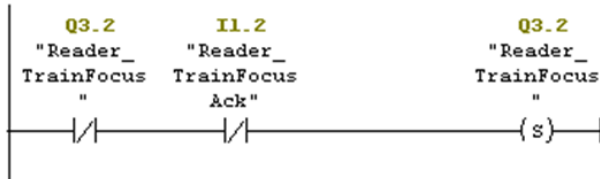
Reduced to the basics the process of invoking a Soft Event consists of the following:

FC3 : Train Focus

Initiate the "Train Focus" operation and monitor it to completion.

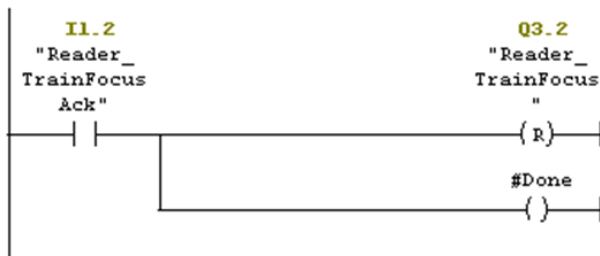
Network 1 : Title:

Issue "Train Focus" signal.



Network 2 : Title:

Release "Train Focus" signal as soon as it is acknowledged by the reader.



Network 3 : Title:

Set the return FC state.

Note, this will only return TRUE after the acknowledge has been received from the reader. Otherwise it will return FALSE.



Executing DMCC Commands

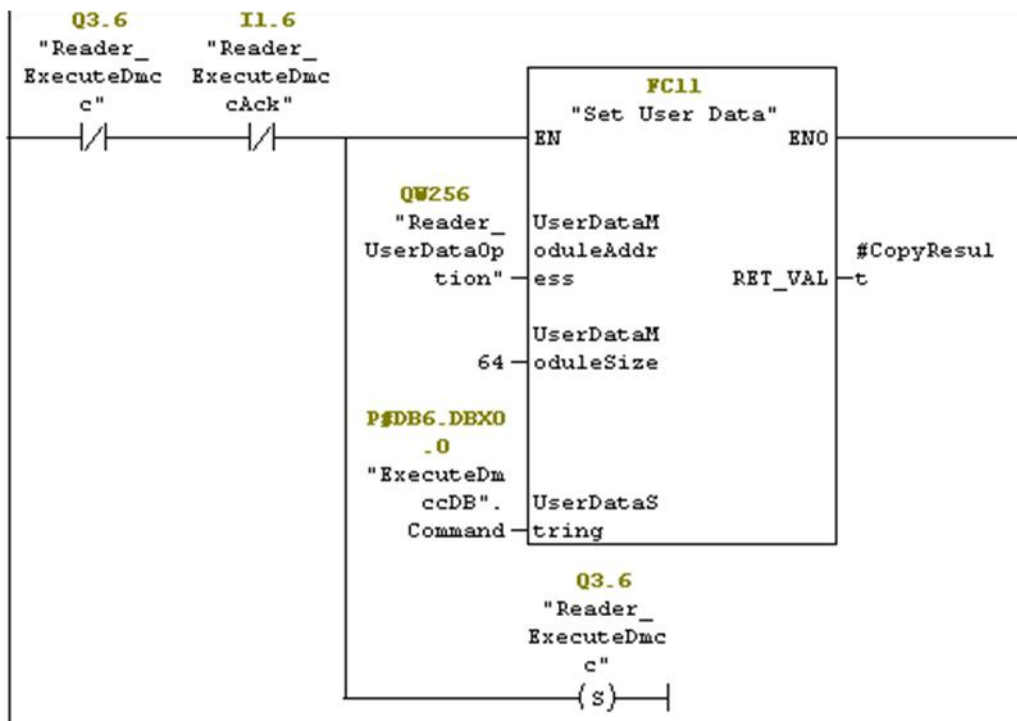
Refer to sample program "DM200_SoftEvents" for the complete example program (for information on how to install it, see Section [Using Soft Events](#)).

NOTE

This sample can be used with any PROFINET enabled DataMan reader.

"Execute DMCC" is a Soft Event which requires the added step of loading the User Data module with the desired DMCC command string before invoking the event. Note, the Soft Event mechanism does not provide a means of returning DMCC response data (other than a failure indication). So this mechanism cannot be used for DMCC ">GET..." commands.

The process of executing a DMCC command is the same as that for all other Soft Events (see example above) except the step of invoking the Soft Event also includes copying the command string to the User Data Module. In this example the command string is exists in a Data Block. This example could be expanded to utilize a Data Block with an array of command strings that the copy function could reference by an index value. That would allow the user to pre-define all DMCC commands that are required by the application and invoke them simply by index.



The function "Set User Data" (FC11) simply copies the provided string to the User Data module. Refer to the example program for the actual STL code.

DataMan Application Development

DataMan Control Commands (DMCC) are a method of configuring and controlling a DataMan reader from a COM port or through an Ethernet connection, either directly or programmatically through a custom application.

For a complete list of DMCC commands, click the windows Start menu and browse to *Cognex* → *DataMan Setup Tool v x.x* → *Documentation* → *Command Reference*.

DMCC Overview

Depending on the DataMan reader you are using, the COM port connection can be either RS232 or USB and an Ethernet connection can be established through the Telnet protocol. By default, the DataMan reader is configured to communicate over TCP port number 23, but you can use the Setup Tool to assign a different port number as necessary.

Command Syntax

All DMCC commands are formed of a stream of ASCII printable characters with the following syntax:

command-header command [arguments] footer

For example:

```
||>trigger on\CR\LF
```

Command Header Syntax

```
||checksum:command-id>
```

All options are colon separated ASCII text. A header without the header-option block will use header defaults.

checksum

0: no checksum (default)

1: last byte before footer is XOR of bytes

command-id

An integer command sequence that can be reported back in acknowledgement.

Header Examples

Example	Description
>	Default Header
0:123>	Header indicating no-checksum and ID of 123
1>	Header indicating checksum after <i>command</i> and <i>data</i> .

Command

The command is an ASCII typable string possibly followed by data. All command names and public parameters data are case insensitive. Only a single command may be issued

DataMan Application Development

within a header-footer block. Commands, parameters and arguments are separated by a space character.

Commands

Short names specifying an action. A commonly used command is GET or SET followed by a Parameter and Value.

Parameters

Short names specifying a device setting. Parameter names are organized with a group of similar commands with one level of structural organization separated by a period ('.').

Arguments

Boolean: *ON* or *OFF*

Integer: *123456*

String: ASCII text string enclosed by quotes (").The string content is passed to a function to translate the string to the final format. The following characters must be backslash escaped: quote (\"), backslash (\\), pipe (\|), tab (\t), CR(\r), LF (\n).

Footer

The *footer* is a carriage return and linefeed (noted as **\CR\LF** or **\r\n**).

Reader Response

The reader will have one of several response formats. The choice of response format is configured using the SET COM.DMCC-RESPONSE command.

Silent: (0, Default) No response will be sent from the reader. Invalid commands are ignored without feedback. Command responses are sent in space delimited ASCII text without a header or footer.

Extended: (1) The reader responds with a *header data footer* block similar to the command format.

NOTE

While the reader can process a stream of DMCC commands, it is typically more robust to either wait for a response, or insert a delay between consecutive commands.

`||checksum:command-id[status]`

checksum

The response uses the same checksum format as the command sent to the reader.

0: no checksum

1: last byte before footer is XOR of bytes

command-id

The command-id sent to the reader is returned in the response header.

status

An integer in ASCII text format.

0: no error

1: reader initiated read-string

100: unidentified error

DataMan Application Development

- 101:** command invalid
- 102:** parameter invalid
- 103:** checksum incorrect
- 104:** parameter rejected/alterd due to reader state

Examples

Command	Silent Response	Extended Response	Description
>GET SYMBOL.DATAMATRIX\r\n	ON	[0]ON\r\n	Is the DataMatrix symbology enabled?
>SET SYMBOL.DATAMATRIX ON\r\n	<i>no response</i>	[0]\r\n	Enable the DataMatrix symbology.
>TRIGGER ON\r\n	<i>decoded data or no-read response</i>	[0]\r\n [1] <i>decoded data or no-read response in base64</i> \r\n	Trigger Command

DMCC Application Development

You can use DMCCs as an application programming interface for integrating a reader into a larger automation system.

If you want to create your own application from scratch that you want to communicate with the DataMan reader through the serial port, make sure you set port.DtrEnable = true, if the port is an instance of the SerialPort class.

You can also use the DMCC .NET Software Development Kit (hereafter referred to as DMCC .NET). The following sections give detailed information about installing DMCC .NET, its contents, building the DMCC .NET sample code, and about the different DMCC recipes.

DMCC .NET Contents

The DMCC .NET comprises the Cognex.DataMan.SDK.dll and its documentation. In addition, a sample application is provided in the form of a complete Microsoft Visual Studio project (in .csproj format).

Building the DMCC .NET Sample Code

Open the Sample code in Microsoft Visual Studio and choose Build solution.

Using DMCC .NET

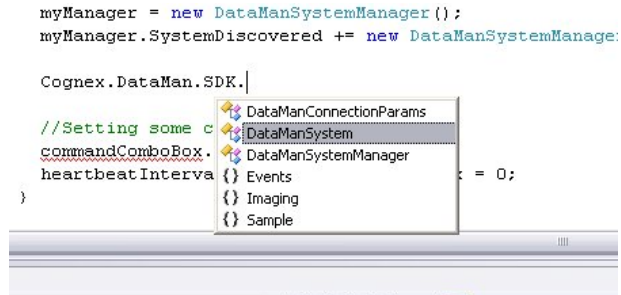
To use DMCC .NET for your own purposes, perform the following steps:

1. In Microsoft Visual Studio, click Create Solution/Project.
2. Under Project, right-click References and choose Add Reference...
3. In the pop-up window, click the Browse tab and look for the Cognex.DataMan.SDK.dll file in the directory where you installed the solution.
4. Add the following lines to the beginning of your code:

DataMan Application Development

```
using Cognex.DataMan.SDK;  
using Cognex.DataMan.SDK.Events;
```

to find the different elements belonging to the SDK in these namespaces. They will appear in the intellisense as seen in the following image:



Enumerating DataMan Devices

1. Create a new DataManSystemManager.

```
DataManSystemManager myManager = new DataManSystemManager();
```

2. Subscribe to its SystemDiscovered event.

```
myManager.SystemDiscovered += new  
DataManSystemManager.SystemDiscoveredEventHandler(myManager_NetworkDeviceApp  
eared);
```

3. Create event handler of type SystemDiscoveredEventHandler.
4. The event handler argument of type SystemDiscoveredEventArgs has a property System of type DataManSystem. These DataManSystem objects represent readers. You can store these DataManSystem objects in your own collection.
5. To start device discovery, call the myManager.Refresh() method.

NOTE

The SystemDiscovered event will be fired every time a device is detected.

Connecting to a DataMan Device

Connect to your device by performing the following steps:

1. Create DataManSystem instance

```
DataManSystem mySystem = new DataManSystem();
```

2. Create DataManConnectionParams instance

```
DataManConnectionParams myParams = new DataManConnectionParams(address);
```

where the address is a string. This can be a COM port (for example, "COM23") or an IP address (for example, "10.10.123.42").

3. Call the Connect() method of your DataManSystem instance.

DataMan Application Development

```
mySystem.Connect(myParams);
```

4. (Optional) Verify if you are connected:

```
if (mySystem.IsConnected())
```

5. To disconnect, call

```
mySystem.Disconnect();
```

NOTE

Check the events of DataManSystem. After calling Connect(), subscribe to the events that you need. Unsubscribe before calling Disconnect().

DataMan Device Settings

Use DataManSystem.SendDmcc() for different settings. Refer to the *DMCC Command Reference* available through the Windows Start menu.

There are two versions of the SendDmcc() method:

1. The SendDmcc(string) takes a string which is the DMCC including the header. This is a non-blocking, asynchronous method for sending DMCC. This returns an ID so that you can find the matching response in the event handler. You have to subscribe to the appropriate events of the DataManSystem instances and handle responses in the appropriate event handler, such as DmccResponseArrived.
2. The SendDmcc(string,int) takes a DMCC as the first argument and the second argument is a timeout value. This is the blocking version. It will not return until a response is received or the timeout expires (when a null response is returned). You can retrieve the DMCC result as the return value of the SendDmcc() method.

The following is an example for sending a DMCC:

```
int id = mySystem.SendDmcc("||>GET DEVICE.TYPE");
```

Displaying Static and Live Images from a DataMan Device

To have static images displayed, use DataManSystem.GetImage() to get images.

To have live images displayed, perform the following steps:

1. Turn on live image acquisition mode by setting the DataManSystem.Imaging.Live.Enabled to true.
2. Periodically poll the device for images by using DataManSystem.GetImage(). See an example implementation in the source of the Sample application. In the example, a new polling thread is created to avoid locking the GUI.
3. Turn off live image acquisition by setting the DataManSystem.Imaging.Live.Enabled to false.

NOTE

In addition, Live Image properties, such as size, quality and format, can be controlled through the DataManSystem.Imaging.Live properties.

Distribué par :



PRECONISATEUR DE SOLUTIONS DEPUIS 1986

2 rue René Laennec 51500 Taissy France
Fax: 03 26 85 19 08, Tel : 03 26 82 49 29

Email : hvssystem@hvssystem.com
Site web : www.hvssystem.com