COGNEX

DataMan® Communications and Programming Guide

2019 February 05
Revision: 6.1.6.27

Legal Notices

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.

Companies, names, and data used in examples herein are fictitious unless otherwise noted. 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 © 2019. Cognex Corporation. All Rights Reserved.

Portions of the hardware and software provided by Cognex may be covered by one or more U.S. and foreign patents, as well as pending U.S. and foreign patents listed on the Cognex web site at: cognex.com/patents.

The following are registered trademarks of Cognex Corporation:

Cognex, 2DMAX, Advantage, AlignPlus, Assemblyplus, Check it with Checker, Checker, Cognex Vision for Industry, Cognex VSOC, CVL, DataMan, DisplayInspect, DVT, EasyBuilder, Hotbars, IDMax, In-Sight, Laser Killer, MVS-8000, OmniView, PatFind, PatFlex, PatInspect, PatMax, PatQuick, SensorView, SmartView, SmartAdvisor, SmartLearn, UltraLight, Vision Solutions, VisionPro, VisionView

The following are trademarks of Cognex Corporation:

The Cognex logo, 1DMax, 3D-Locate, 3DMax, BGAII, CheckPoint, Cognex VSoC, CVC-1000, FFD, iLearn, In-Sight (design insignia with cross-hairs), In-Sight 2000, InspectEdge, Inspection Designer, MVS, NotchMax, OCRMax, PatMax RedLine, ProofRead, SmartSync, ProfilePlus, SmartDisplay, SmartSystem, SMD4, VisiFlex, Xpand

Portions copyright © Microsoft Corporation. All rights reserved.

Portions copyright © MadCap Software, Inc. All rights reserved.

Other product and company trademarks identified herein are the trademarks of their respective owners.

Table of Contents

Symbols

The following symbols indicate safety precautions and supplemental information:

WARNING: This symbol indicates a hazard that could cause death, serious personal injury or electrical shock.

CAUTION: This symbol indicates a hazard that could result in property damage.

Note: This symbol indicates additional information about a subject.

Tip: This symbol indicates suggestions and shortcuts that might not otherwise be apparent.

About This Manual

The *DataMan Communications and Programming Guide* provides information on how to integrate DataMan readers into your particular environment, including:

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

The DataMan reader connected to a network can be triggered to acquire images by several methods. It can be done by:

- the DataMan Setup Tool,
- it can be triggered by
    - trigger bits
    - manipulating objects through/by industrial protocols
    - or through a DMCC command.

This document provides a detailed description on how to do each.

Networking

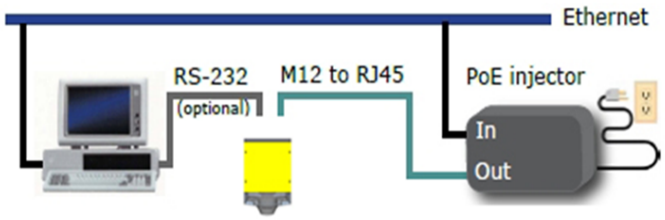
You can connect your DataMan device via a simple Ethernet connection. You can either set the IP address and subnet mask of your DataMan device manually or let them be configured automatically using DHCP.

Connecting Your DataMan to the Network

Connecting Your Corded 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.



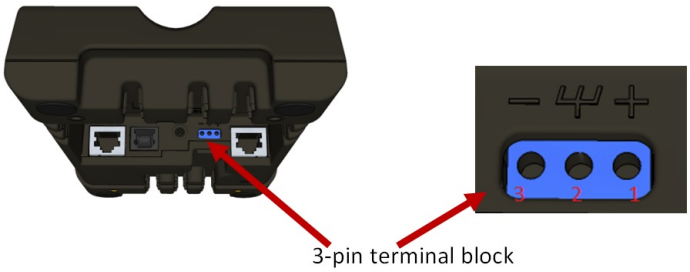
To disconnect the reader:

1. Disconnect the reader from the PoE injector.
2. Disconnect the power cord from the PoE injector.
3. Disconnect the PoE injector from the Ethernet network.

Connecting Your DataMan Intelligent Base Station to the Network

1. If you use **DMA-IBASE-00**, 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 (POE) adapter, that will power up your base station. If you use the **DMA-IBASE-BT-01** base station, use direct connection with a 24V power supply. **DMA-IBASE-BT-01** offers a 3-pin terminal block:

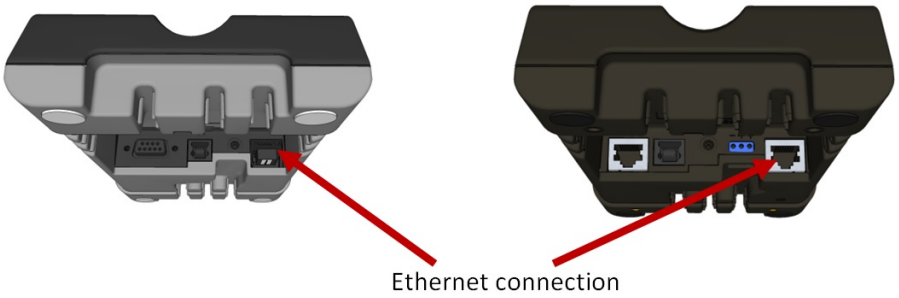


Pin #	Signal
1	+24V
2	Shield
3	GND

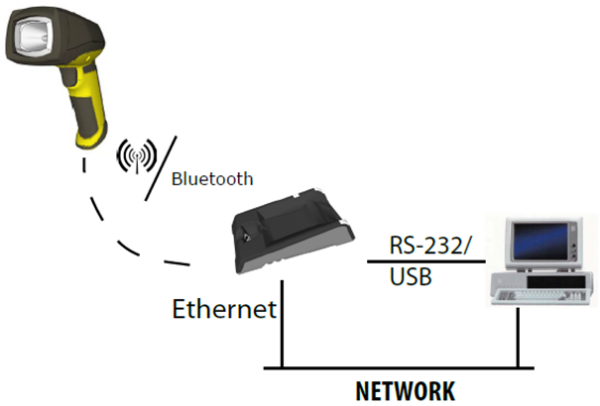
Note: Never connect the terminal block and barrel connector power supply at the same time.

2. Connect your base station to your PC with an Ethernet cable.

DMA-IBASE-00 *DMA-IBASE-BT-01*



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



Direct Connection to Your Computer

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

configure your reader and your PC.

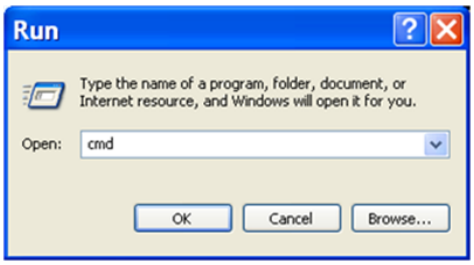
Link Local Addressing automatically requests and assigns an IP address. In the DataMan Setup Tool, this corresponds to the DHCP Server communication option. This is the default, you do not have to make any changes.

You can also manually configure your DataMan device 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.

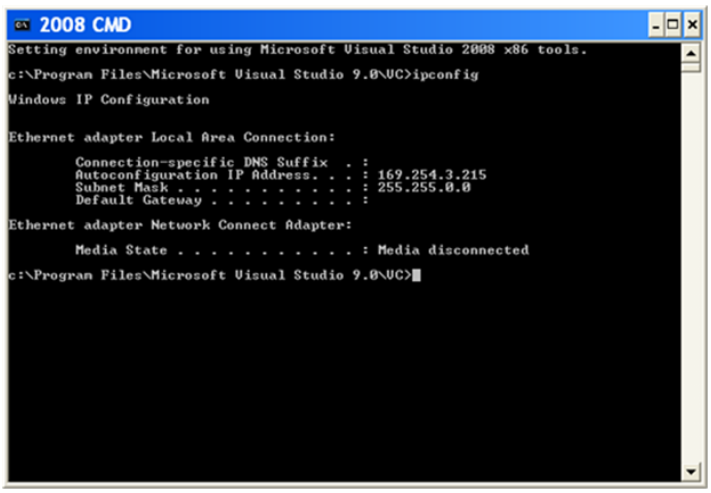
Configuring the DataMan to Reside on the Same Subnet as the PC

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.



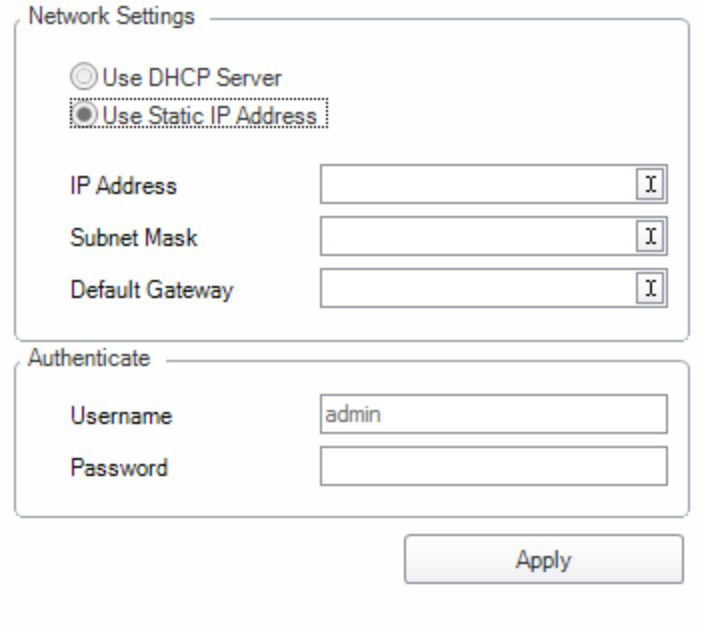
3. In the command prompt window, type “ipconfig” and press Enter. A listing of all network adaptors on the PC is shown.



4. Record your PC’s IP Address and Subnet Mask. In this example,

- IP Address is 169.254.3.215
- Subnet Mask is 255.255.0.0

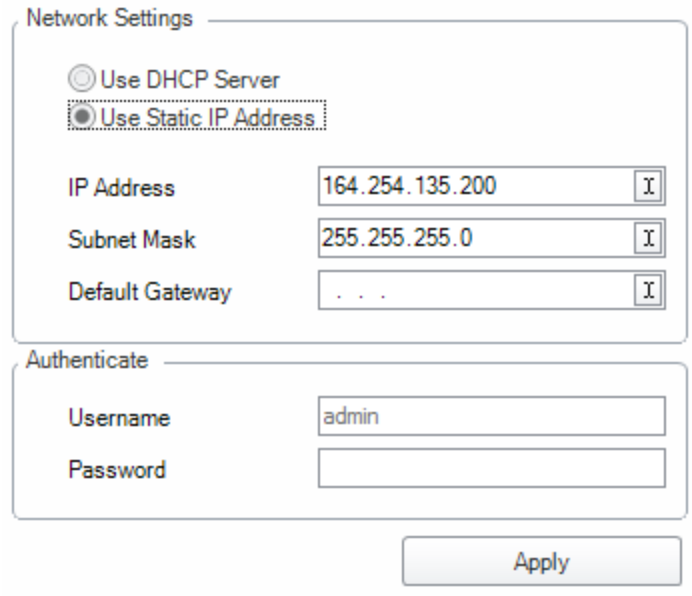
5. Go to **Reader Maintenance** in the DataMan Setup Tool, select the device to be used from the list, and use the **Network Settings** dialog to manually configure the network settings on the target DataMan reader.



6. To force the network settings on your DataMan:
   a. Select **Use Static IP Address**.
   b. 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.



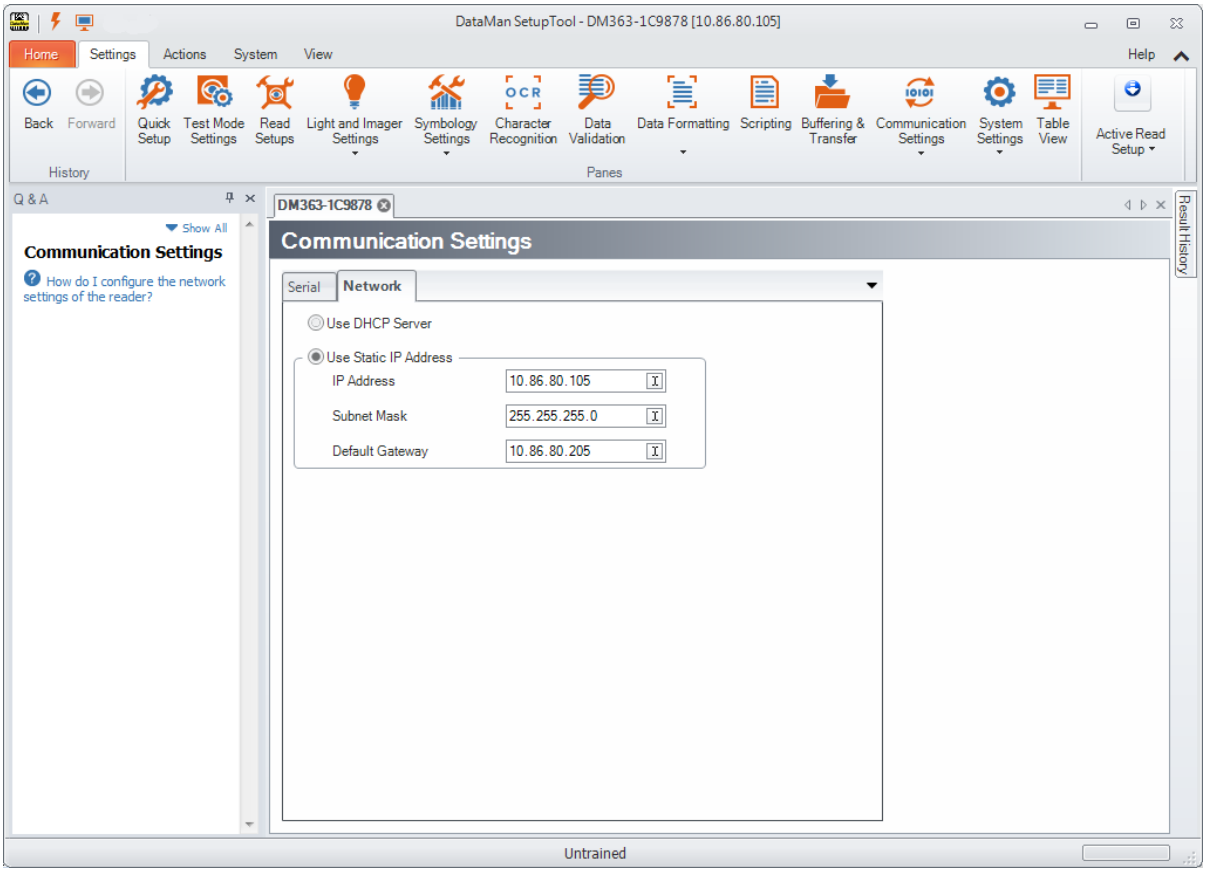
Click OK. Your DataMan device is configured to the network settings specified, and it reboots automatically.

Your DataMan reader appears under the **Discovered 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, press the **Refresh** button in the DataMan Setup Tool's **Connect** page. This will force the DataMan 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

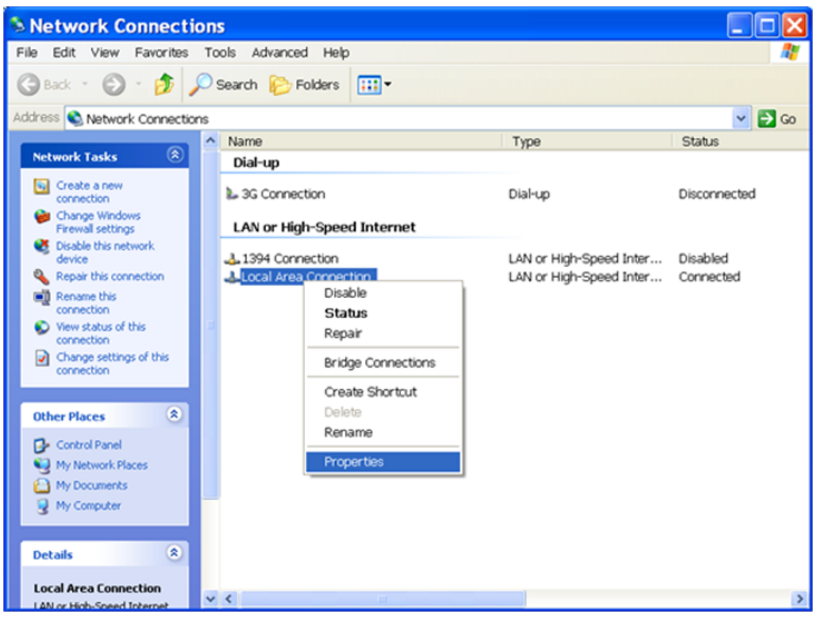
If it is preferred that the DataMan network settings remain unchanged, you must already know the IP Address and Subnet Mask of the DataMan or you must connect to the DataMan via RS-232 to find them out. The DataMan IP Address and Subnet Mask can be found under **Communication Settings**.



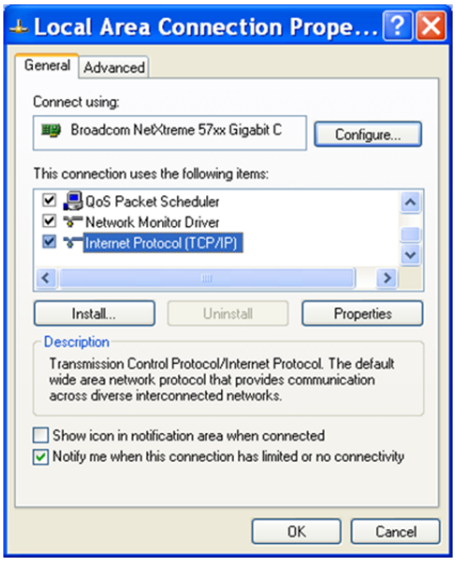
Once the IP Address and Subnet Mask of the DataMan device are known, the PC's network settings can be changed.

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

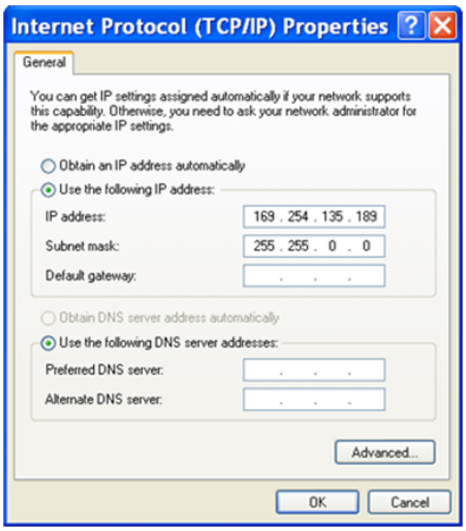
1. In the Start Menu, right click **My Network Places**, click the Properties menu option to launch **Network Connections**.
2. Right click on the network adapter connected to the DataMan and select the **Properties** menu option.



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



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

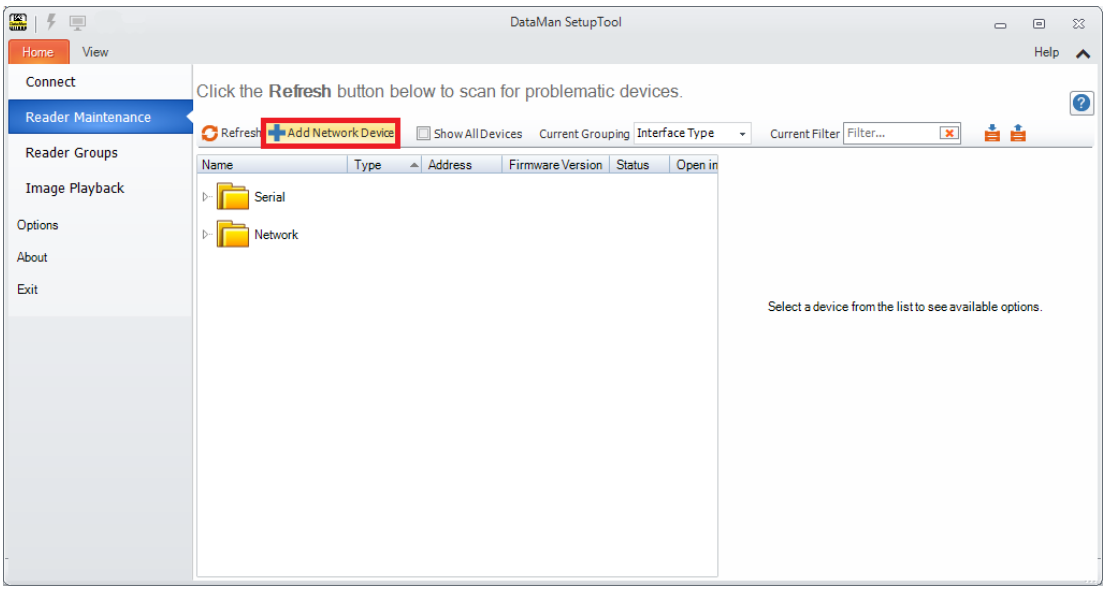


5. Click **Close**. The network settings of your PC will change to the new specified values.
6. Reboot the DataMan device. It appears under the **Discovered Devices** node on the **Connect** page after the network address has been resolved.
7. If the device does not appear after 1 or 2 minutes, click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

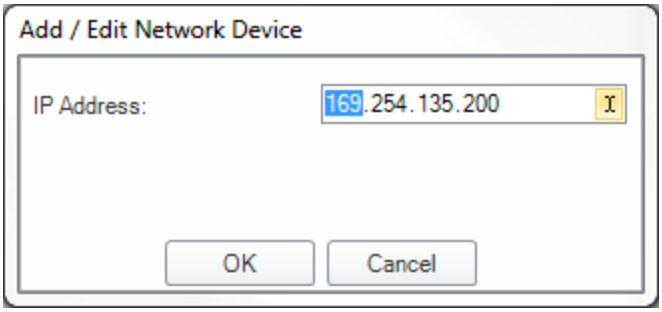
Connecting Your Reader Across Subnets

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

1. In the DataMan Setup Tool's **Reader Maintenance** page, click **Add Network Device**.



2. Enter the actual IP Address of the target DataMan device.



3. Click OK. The reader appears under the **Network** node. Double click the new node or select it and click the **Connect** button. If the device is available, the reader will be connected.

Connecting Your DataMan to the Network Wirelessly

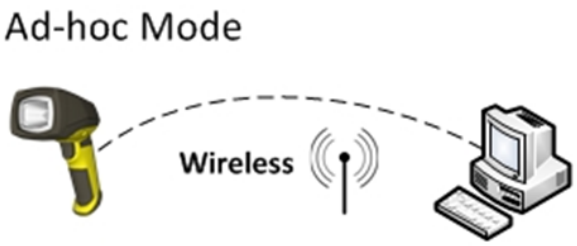
You can connect to your DataMan reader via the wireless network as well. For this, you need to use the WiFi slide-in with the device.

Ad-hoc Connection

The default factory settings for the wireless configuration of the device are:

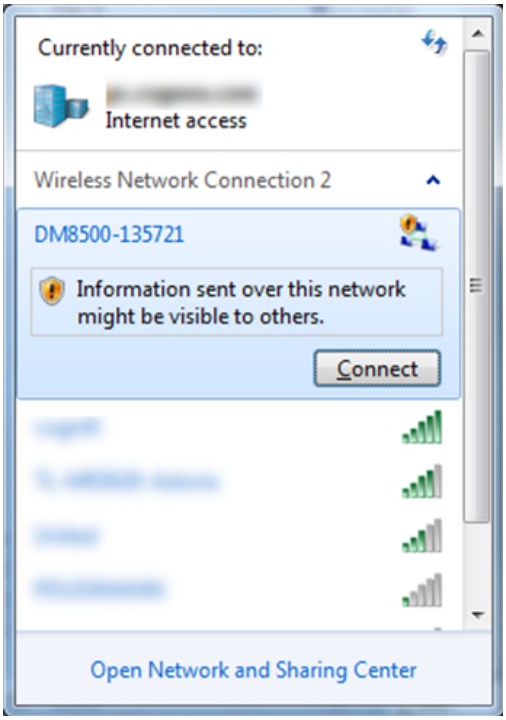
- ad-hoc connection
- no encryption and no authentication
- SSID: the name of the device

This means that you can connect to your DataMan without the base station or a router.



To connect to your DataMan reader in ad-hoc mode, perform the following steps:

1. Make sure that the DataMan device is (re)set to factory settings.
2. Search for the DataMan device among the available wi-fi connections and connect to it.

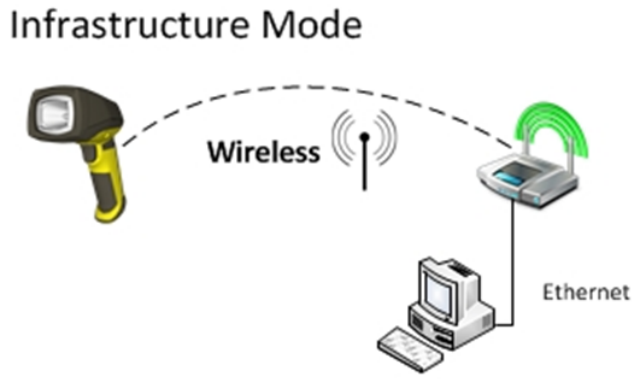


3. Open the DataMan Setup Tool.
4. Search for the device and connect to it.
5. Once you are connected to your DataMan device in the DataMan Setup Tool, you can configure the wireless connection.
    a. Authentication: only Open Mode can be selected.
    b. Encryption method: WEP-40 and WEP-104. You can enter a passphrase for these methods.

Infrastructure Mode

You can set Wireless Infrastructure mode in the DataMan Setup Tool as well.

1. Connect to your device in the DataMan Setup Tool.
2. In the **WiFi** tab of **Communication Settings**, select **Infrastructure mode** from the **Network Type** combo box. A warning appears if the SSID name is identical to the device name, as this designates misconfiguration of the device.



3. Select from the following authentication modes:

Authentication mode	Encryption mode	Requirements
Open System	WEP-40, WEP-104	passphrase
WPA-PSK, WPA2-PSK	TKIP, AES, TKIP/AES	passphrase
EAP-TLS (see the section below)	TKIP, AES, TKIP/AES	• Client's certificate • CA's certificate • Client's private key • Client's username
PEAP-MSCHAPV2 (see the section below)	TKIP, AES, TKIP/AES	• CA's certificate • Client's username • Client's password

EAP-TLS Authentication Mode

Encryption methods: TKIP, AES, TKIP/AES are supported. All of these methods require specifying several PEM files, which are created by the user's local system administrator, and contain certificate information.

These certificates will be used to encrypt the communication between the Wi-Fi Access Point and the reader.

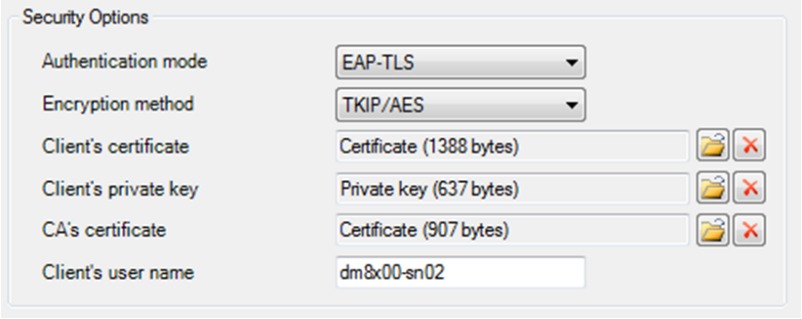
The following certificates are required:

- Client's certificate. This must be different for each reader. It may be publicly accessible (e.g. on a company webpage).
- CA's certificate (CA = Certificate Authority). One such file will be created for each authentication server within the company. It may be publicly accessible.
- Client's private key. This must be different for each reader. It must not be publicly accessible; must be stored and handled confidentially.

Uploading a Certificate File to DataMan

You can upload these files in the DataMan Setup Tool one by one: click the folder button beside the fields, select the proper file and the file content will be uploaded to the device.

A short message shows if a certificate is specified. The text "<not set>" appears in the field if there is no key or certificate specified.

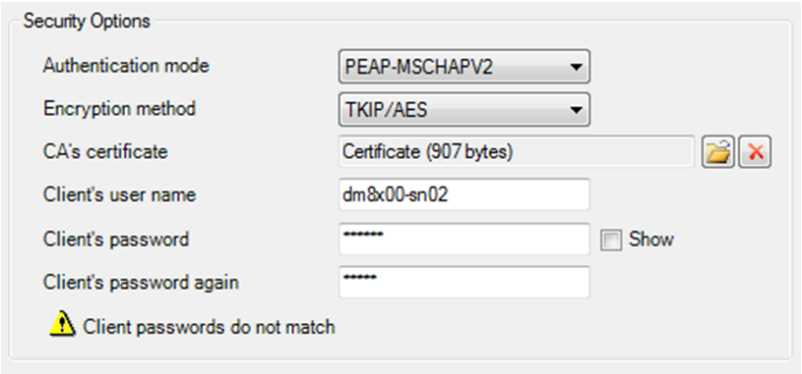


Removing a Certificate File from DataMan

Click the red X button beside the corresponding field to delete an existing certificate from your device. The certificates are saved into device backups, and may be completely restored.

PEAP-MSCHAPV2 Authentication Mode

Encryption methods: TKIP, AES, TKIP/AES are supported. All these methods require a PEM file containing the CA's certificate, the client's user name, and a password.



Certificate Files

The DataMan Setup Tool currently applies the following restrictions about the PEM files:

- Their format must be the industry-standard PEM format (generated by OpenSSL toolkit).
- The PEM dialect may be either PKCS8 or SSLeay.
- Only unencrypted private key and certificate files are allowed.
- The Client's private key and certificate must contain exactly one section; the CA's certificate may contain one or more certificates.
- The user must know the user name stored within her/his own certificate file; and exactly the same name must be included in the **Client's user name** text box. In other words, Setup Tool does not look into the certificate files and does not extract this user name information.
  When the user leaves the Wireless tab page, a reboot confirmation window pops up, and the settings are saved to the device.

Troubleshooting a Network Connection

Based on your network configuration, the DataMan Setup Tool may not be able to communicate with the reader and it will not appear in the list of **Network** devices. If you know the IP address of the reader, use the **Add Network Device** option in the DataMan Setup Tool. This method allows your DataMan reader to appear in the list of Network devices so that you can connect to it through the DataMan Setup Tool and your USB connection.

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 device has the EtherNet/IP protocol disabled. The protocol can be enabled via DMCC, scanning a parameter code, or in the DataMan Setup Tool.

Note: If you have a wireless DataMan reader, read the section on Industrial Protocols for the Wireless DataMan.

DMCC

The following commands can be used to enable/disable EtherNet/IP on the DataMan reader. 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 that 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 on your corded reader.

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

Enable:

Disable:

Scanning the following reader configuration codes will enable/disable EtherNet/IP on your DataMan 8000 base station.

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

Enable:

Disable:

Setup Tool

EtherNet/IP can be enabled by checking **Enabled** on the EtherNet/IP tab of the Industrial Protocols. Make sure to save the new selection by choosing “Save Settings” before disconnecting from the reader.

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

Getting Started

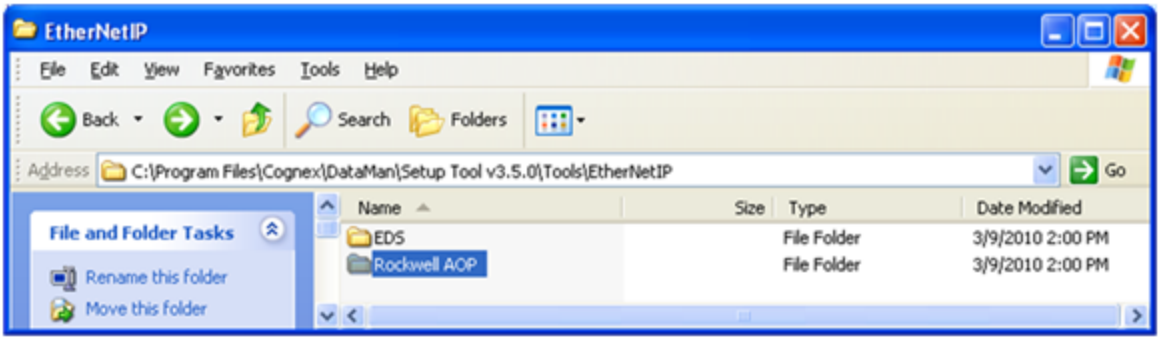
Preparing to use EtherNet/IP involves the following main steps:

- Make sure that 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.

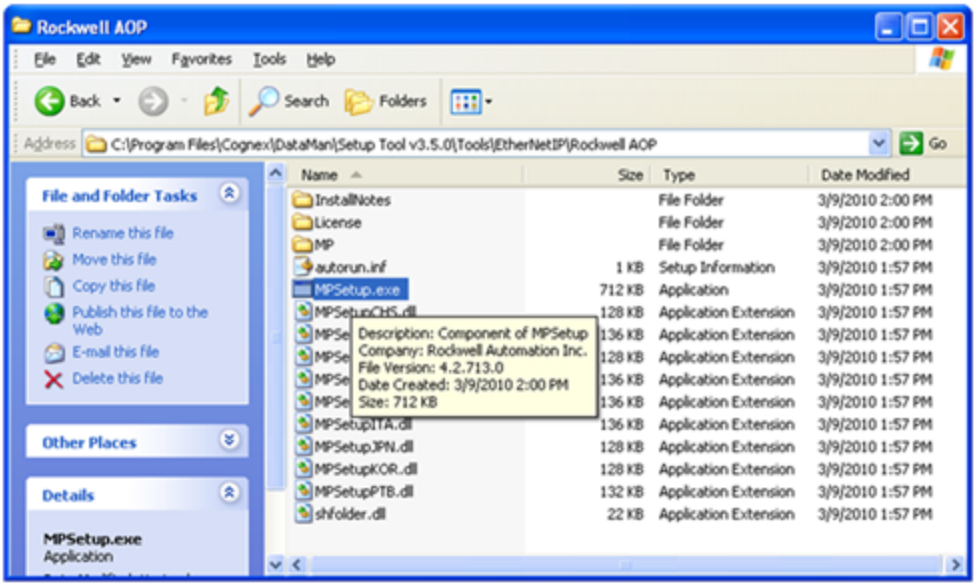
Perform the following steps to set up EtherNet/IP:

1. Verify that the Rockwell Software is on your PC.
2. Make sure that you select the **Add on Profile** installation and the **Samples** installation. The 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 DataMan Setup Tool version that you are using.



4. Select the Rockwell AOP directory:



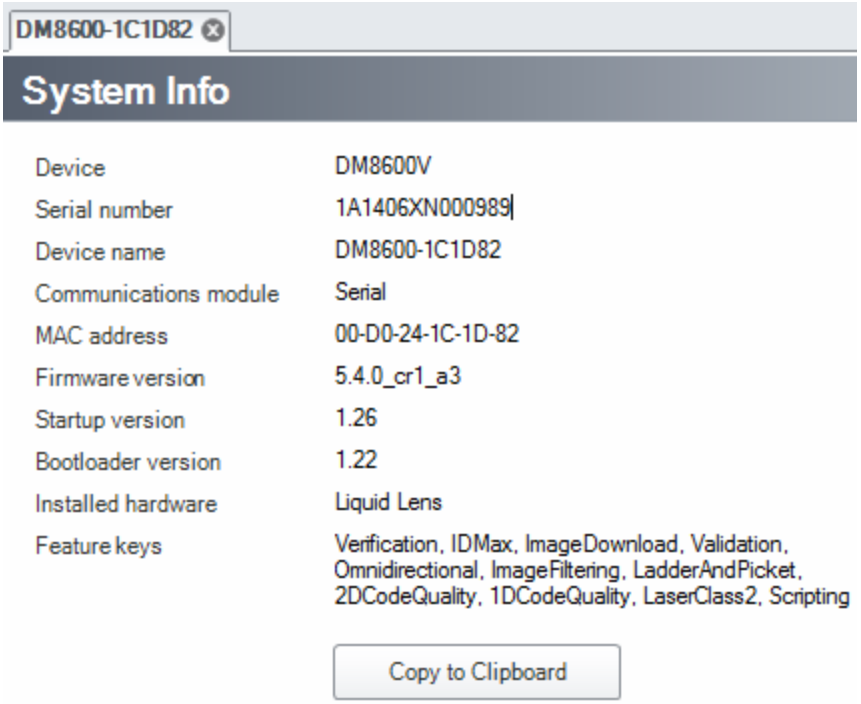
5. If you have not already installed the Rockwell AOP, now run MPSetup.exe.
6. From the Start menu, go to Programs -> Rockwell Software -> RSLinx -> Tools -> EDS Hardware Install Tool.

7. Run the EDS Install tool.

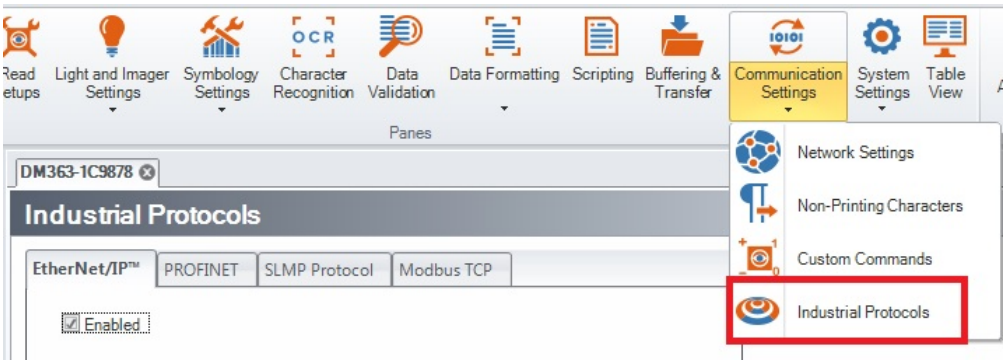
> **Note**: If you have an existing EDS file, uninstall it first, then install the latest version of the EDS.

8. Run the DataMan Setup Tool and update the DataMan firmware.

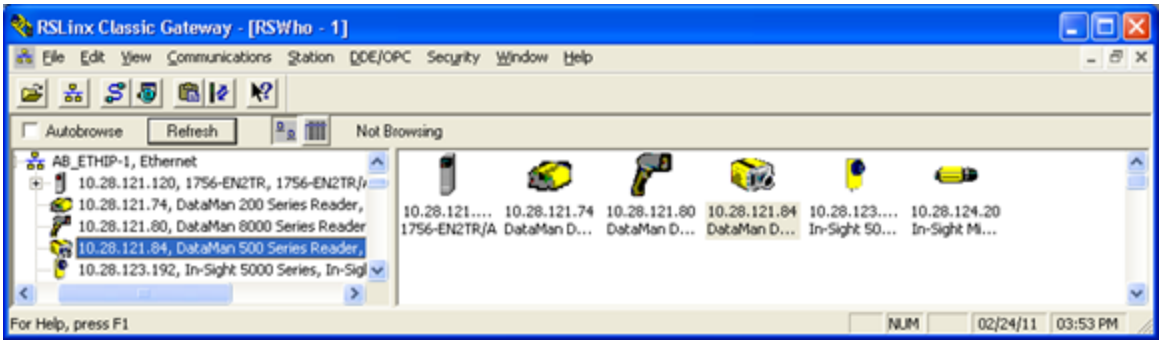
9. Check if the firmware has been loaded into the unit by clicking in the Setup Tool View -> System Info.



In the DataMan Setup Tool, go to the Industrial Protocols pane under **Settings -> Communication Settings** and check the **Enabled** checkbox on the EtherNet/IP™ tab.

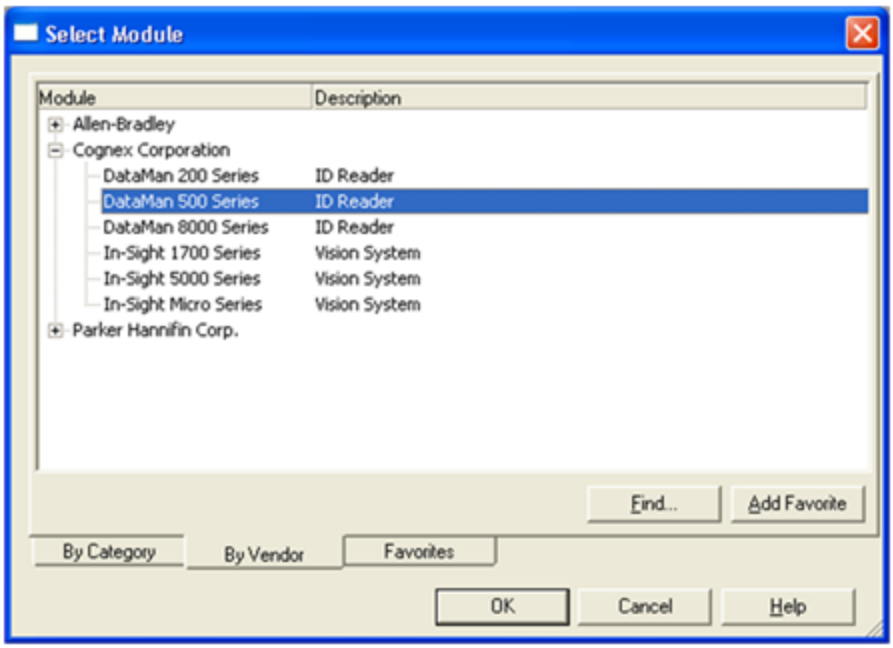


10. In order for the changes to take effect, you must save your settings and cycle power. Go to **System** and click **Save Settings**.
11. Reboot your reader.
12. Your DataMan is visible now in the RSWHO.



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

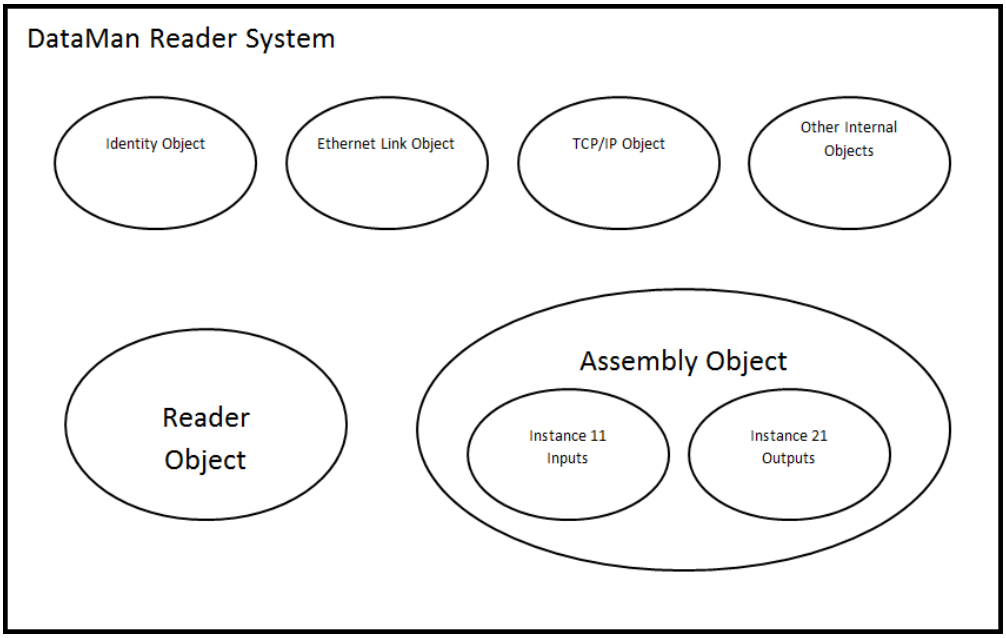
The ID Reader Object is a vendor specific object class. This means that it is not part of the CIP common (public) architecture but rather an extension. It is a custom object that Cognex has added to the EtherNet/IP architecture on the DataMan device. All the data and functionality of this object model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

The ID Reader Object is identified by its vender 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: Reserved 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	ResultsAck	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	UNIT	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
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.

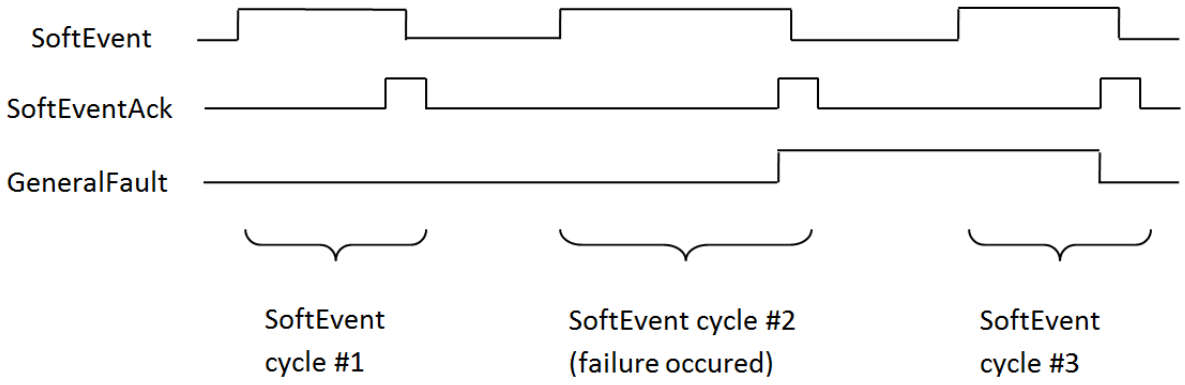
The SoftEvent and SoftEventAck form a logical handshake. After SoftEventAck changes to 1 the original SoftEvent should be set back to 0. When that occurs, SoftEventAck will automatically be set back to 0.

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.
DMCC commands transferred via the Industrial Ethernet protocols (EtherNet/IP, Profinet, etc) will automatically be routed to the wifi reader.
The commands cannot be executed while the wifi reader is powered down, hibernating, or out-of-range.

Service Return Code	Description	DMCC Return Code
0	Success – No error	0
1	Bad Command	-

Service Return Code	Description	DMCC Return Code
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/altered due to reader state	104

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.

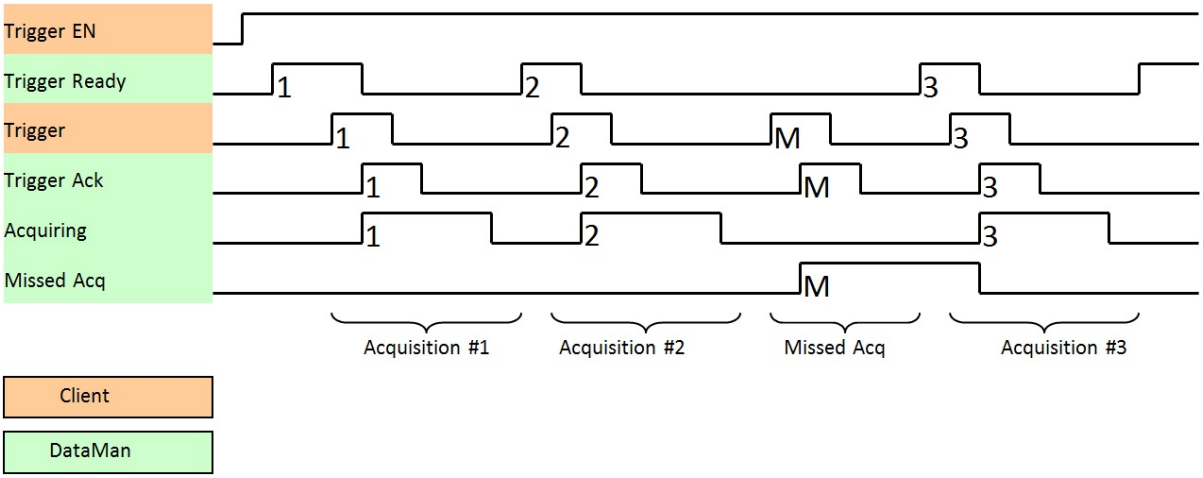
On startup, the AcqTriggerEnable attribute will be False. It must be set to True to enable triggering. When the device is ready to accept triggers, the Trigger Ready bit in the AcqStatusRegister will be set to True.

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

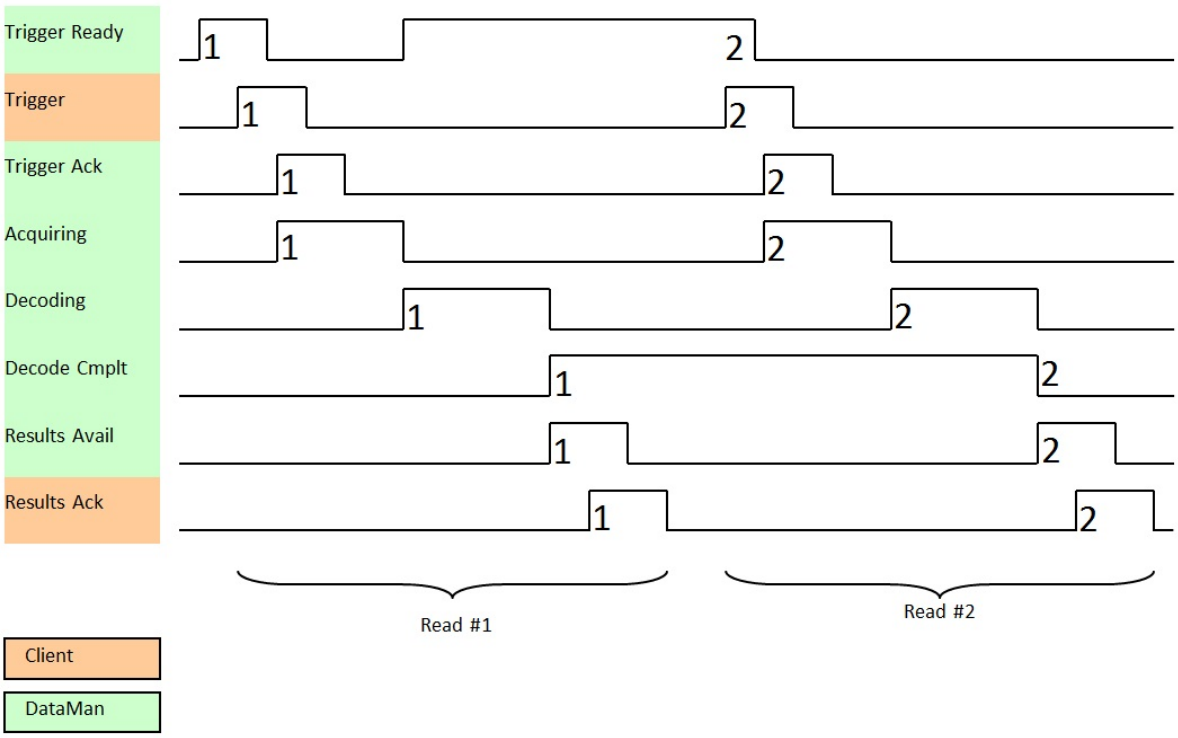
After an image is acquired it is decoded. While being decoded, the Decoding bit of the DecodeStatusRegister is set. When the decode is complete, the Decoding bit is cleared and the Decode Completed bit is toggled.

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 ResultsAck attribute to True. After the Results Available bit is cleared, the client should set the ResultsAck 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.
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 ResultsAck to true.	Set when new results are placed in the DecodeResults attribute. Stays set until the results are acknowledged by setting ResultsAck 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.

Note: If the queue has overflowed and then buffering is disabled, there will be a greater than 1 difference between the TriggerID and ResultID values. This difference represents the number of reads that had occurred but could not be queue because the queue was full (number of lost reads equals TriggerID - ResultID - 1). After the next read, the ResultID value will return to the typical operating value of TriggerID - 1.

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)							
	11								
	12	Result Extended (16-bit integer)							
	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

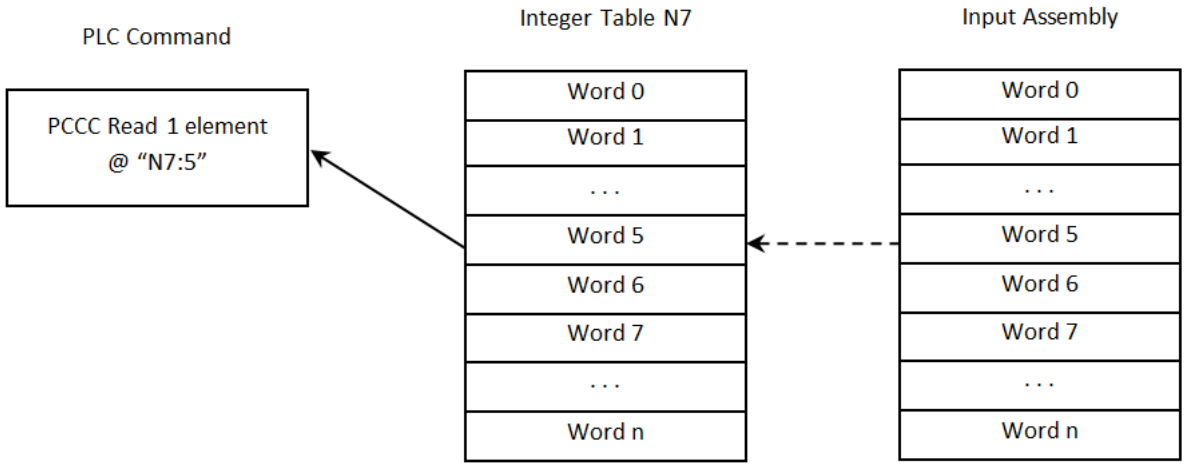
DataMan has limited support for the Rockwell PCCC object. This allows legacy PLC's (PLC-5, SLC, etc) to communicate with DataMan using their native PCCC command set and explicit messaging. The PCCC object allows DataMan to look like a Rockwell PLC-5 logic controller.

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).

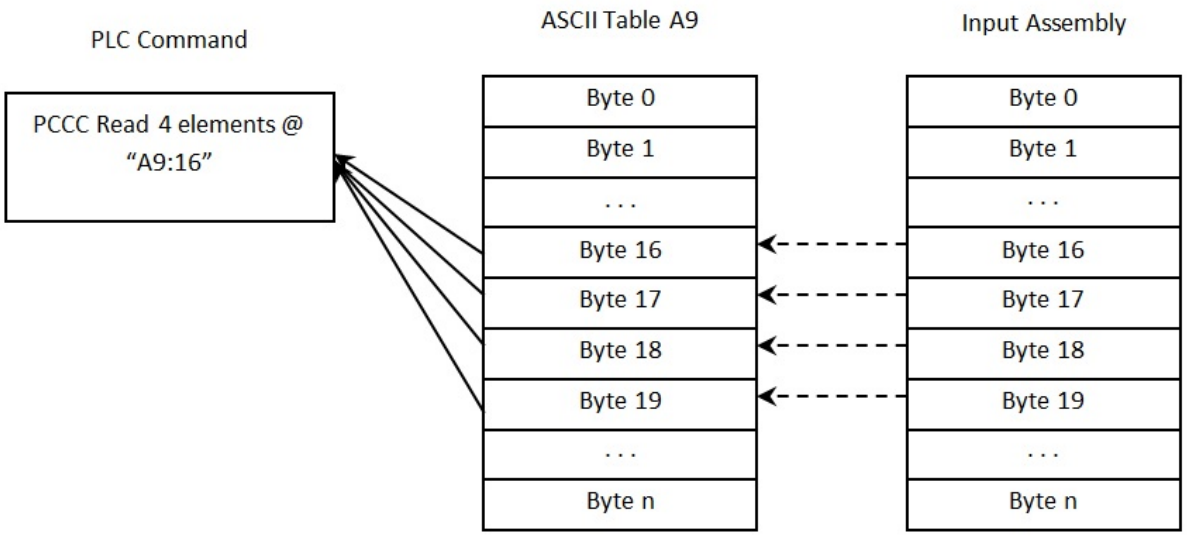
The PCCC object in DataMan maps the read and write requests to ID Reader assemblies (or in one special case to the DMCC service). Read commands return data from the Input assembly (instance 11). Write commands send data to the Output assembly (instance 21). In essence, the PCCC Object gives the outward 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

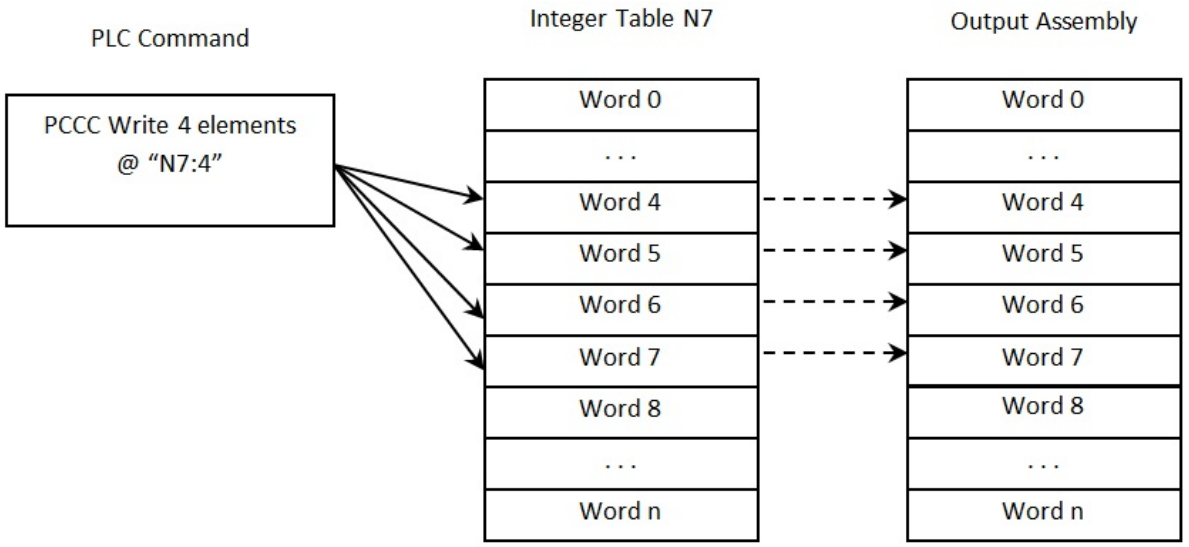
The ResultCode value is located at word offset 5 (counting from 0) of the Input Assembly. To access this value, you would issue the following PLC command:



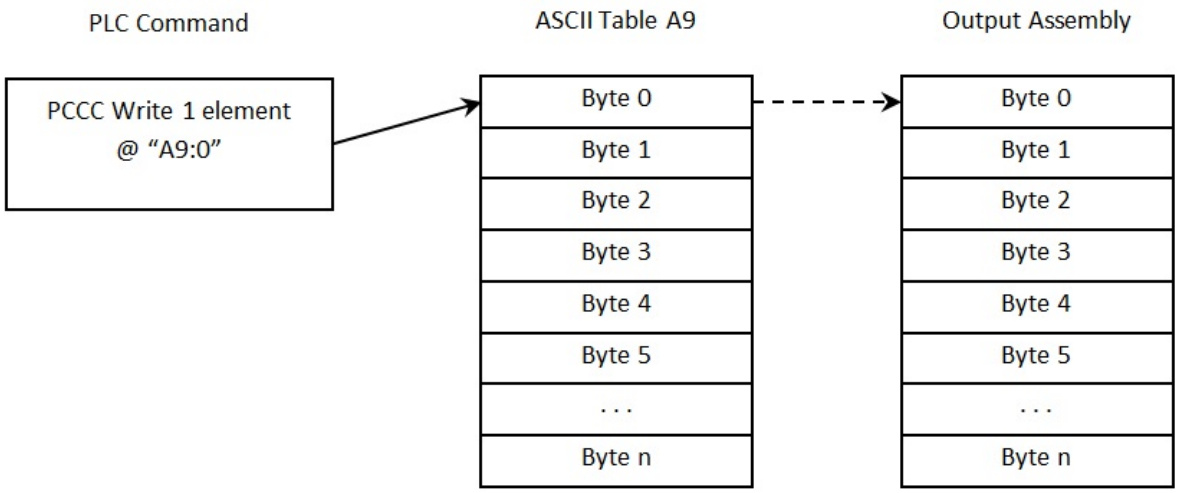
The decode ResultData begins at byte offset 16 (counting from 0) of the Input Assembly. To read the first 4 bytes of result data, you would issue the following PLC command:



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:

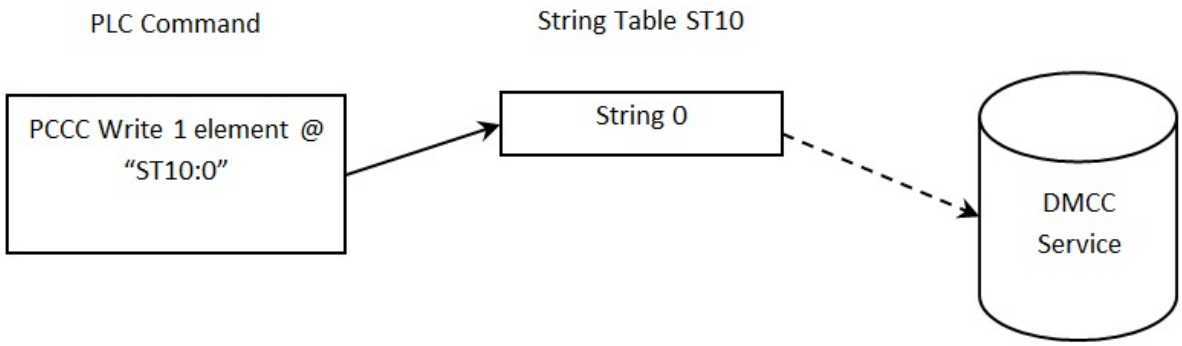


The bit to trigger an acquisition is in byte offset 0 of the Output Assembly. To write to this byte, you would issue the following PLC command:



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



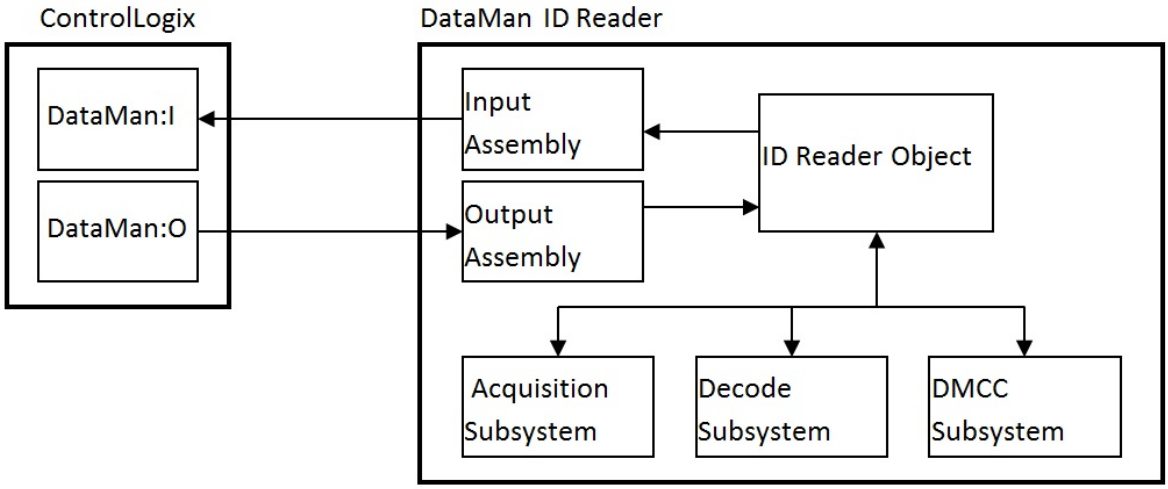
Rockwell ControlLogix Examples

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:



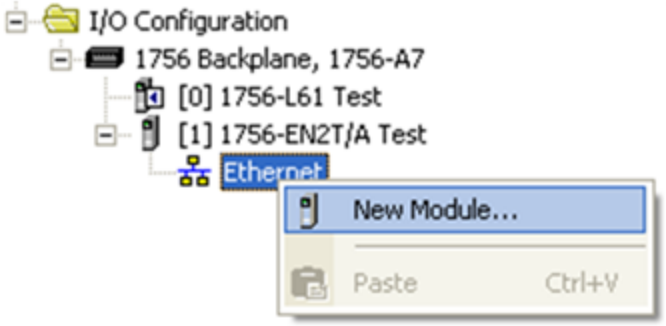
The *Input Assembly* and *Output Assembly* map various attributes (data) from the ID Reader object: The Input Assembly is the collection of DataMan reader data values sent to the PLC (PLC inputs); and the Output Assembly is the collection of data values received by the DataMan reader from the PLC (PLC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

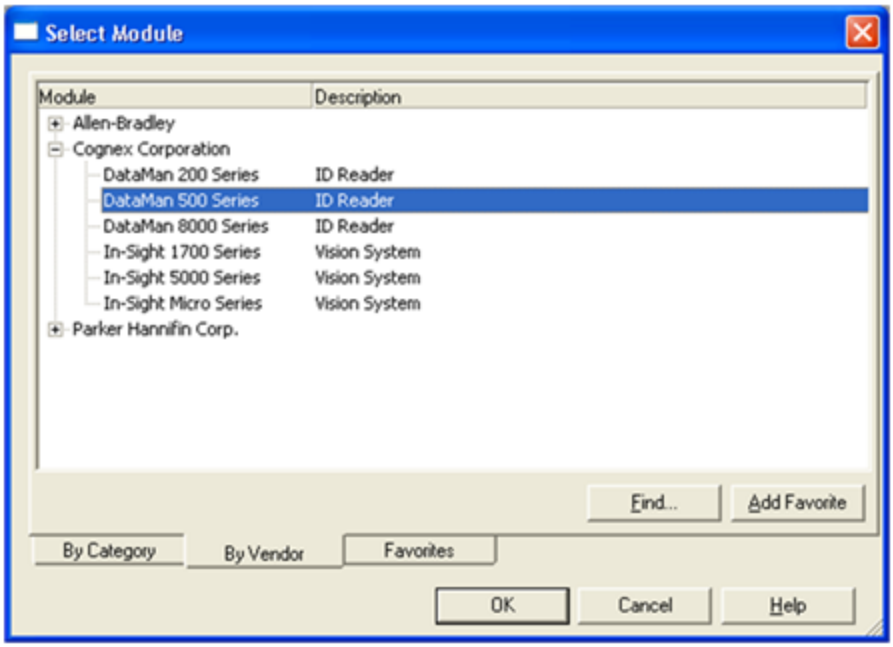
To establish an implicit messaging connection with a ControlLogix PLC:

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:



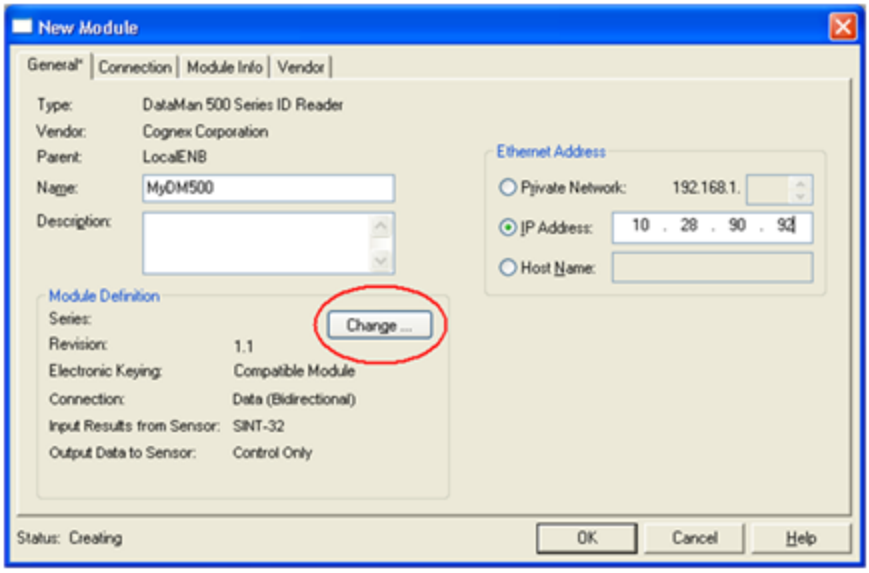
2. From the Select Module dialog, choose your model of DataMan ID Reader from the list.

> **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.

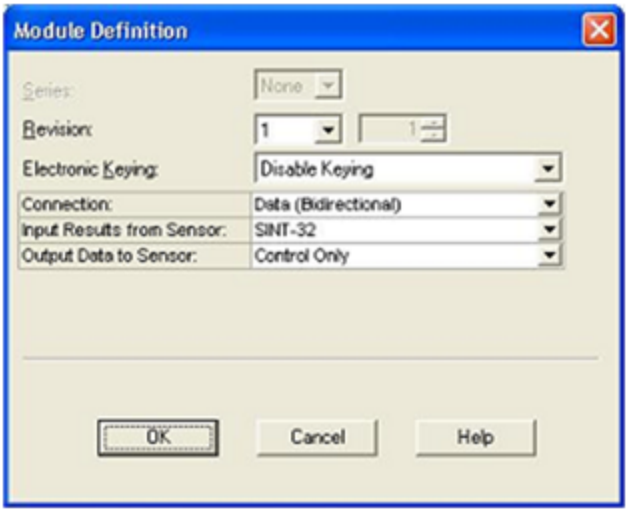
3. After the selection is made, the configuration dialog for the DataMan ID Reader system will be displayed. Give the module a name and enter the DataMan’s IP address. The default is a bidirectional (send/receive) connection consisting of control, status, and 32 bytes of result data with keying disabled. 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.

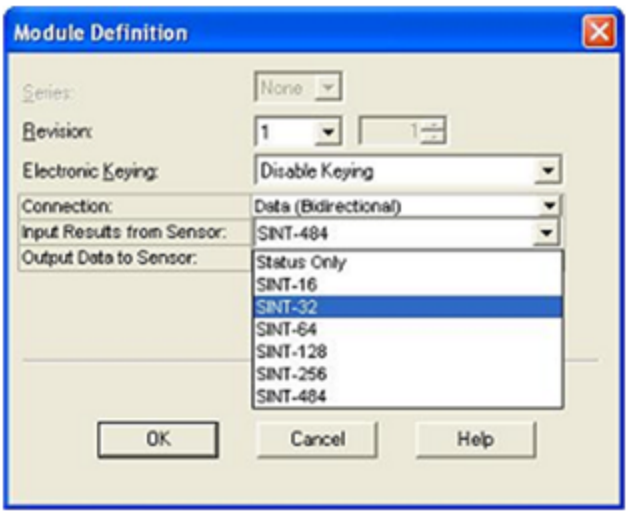
Disable Keying – The controller will not employ keying at all.

Connection: Defines the type of data flow.

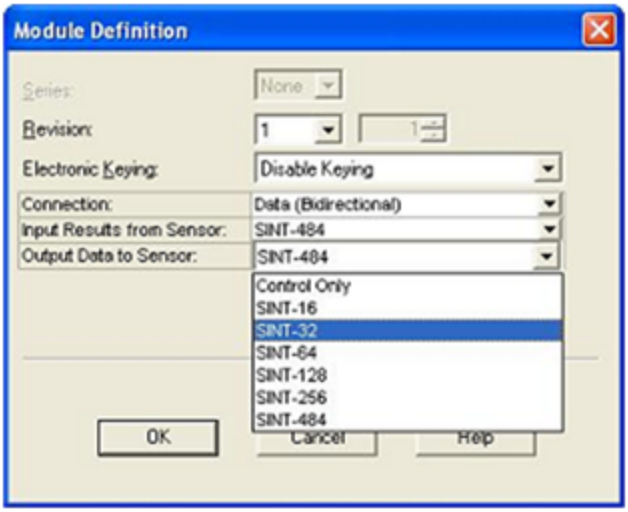
Data (Bidirectional) – The connection will send data (to the DataMan) and receive data (from the DataMan).

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.

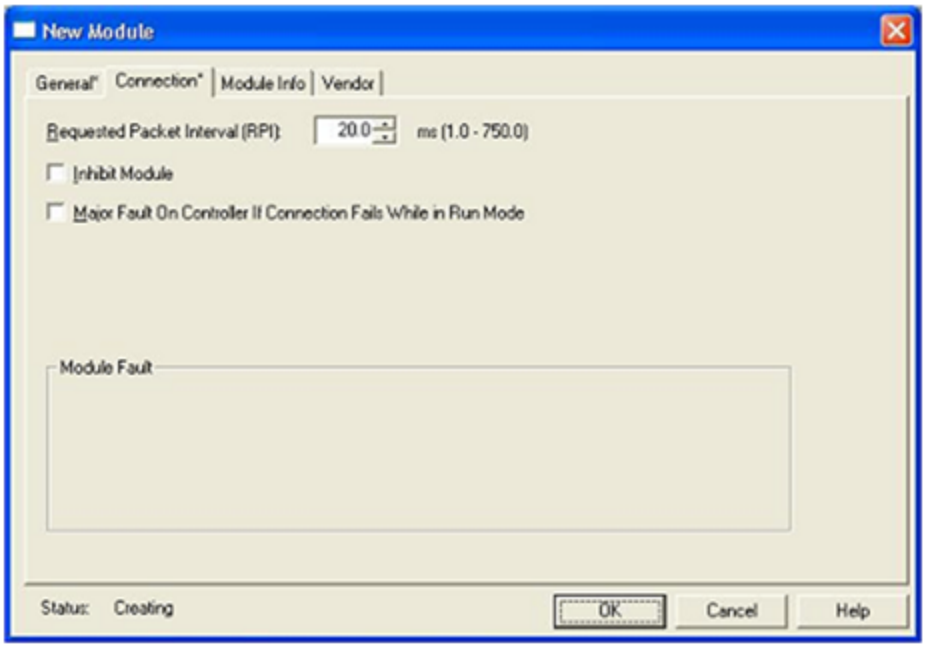
Input Results from Sensor: Defines the amount of data received on the connection (from the DataMan). The minimum amount is the Status data only. The connection can be configured to also receive read result data. The amount of result data received is defined in fixed increments (16 bytes, 32 bytes, 64 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.



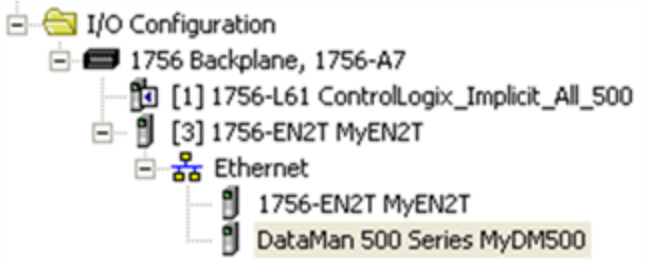
Output Data to Sensor: Defines the amount of data transmitted on the connection (to the DataMan). The minimum amount is the Control data only. The connection can be configured to also send user data. The amount of user data sent is defined in fixed increments (16 bytes, 32 bytes, 64 bytes etc). To enable User Data output, right-click the DataMan module and then go to Properties -> Change -> Output Data to Sensor.



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 ½ 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.

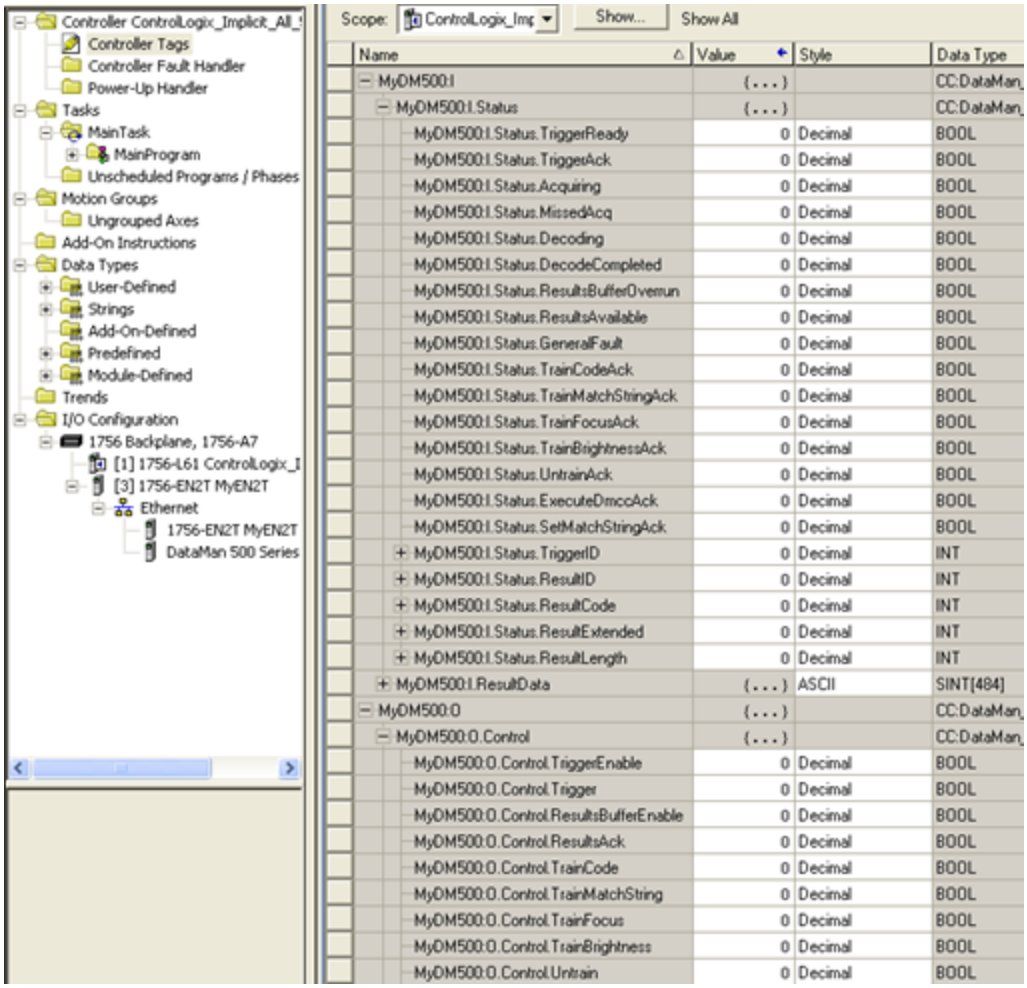


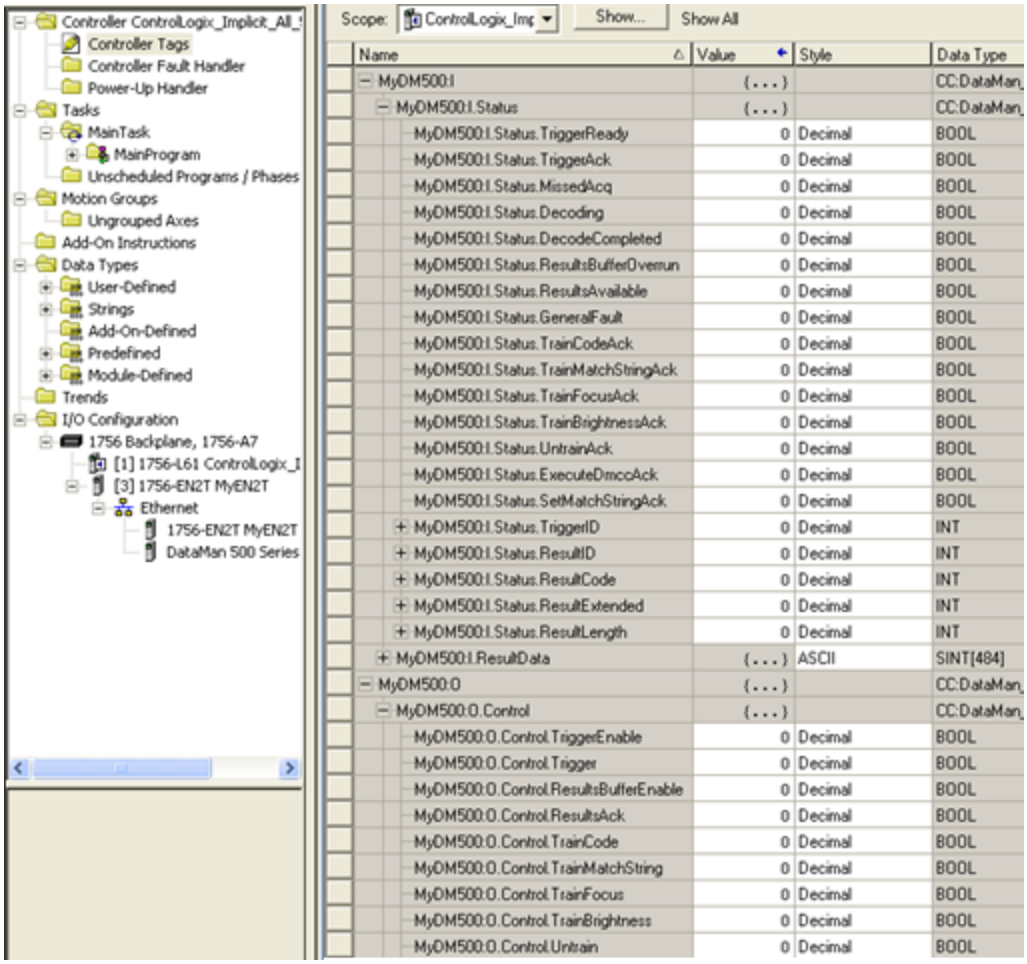
7. After adding the module to ControlLogix, the I/O tree should appear as follows:



8. 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.





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.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]

- **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:O	{...}		CC:DataMan...
MyDM200:O.Control	{...}		CC:DataMan...
MyDM200:O.Control.TriggerEnable	0	Decimal	BOOL
MyDM200:O.Control.Trigger	0	Decimal	BOOL
MyDM200:O.Control.ResultsBufferEnable	0	Decimal	BOOL
MyDM200:O.Control.ResultsAck	0	Decimal	BOOL
MyDM200:O.Control.TrainCode	0	Decimal	BOOL
MyDM200:O.Control.TrainMatchString	0	Decimal	BOOL
MyDM200:O.Control.TrainFocus	0	Decimal	BOOL
MyDM200:O.Control.TrainBrightness	0	Decimal	BOOL
MyDM200:O.Control.Untrain	0	Decimal	BOOL
MyDM200:O.Control.ExecuteDMCC	0	Decimal	BOOL
MyDM200:O.Control.SetMatchString	0	Decimal	BOOL
MyDM200:O.Control.UserDataOption	0	Decimal	INT
MyDM200:O.Control.UserDataLength	0	Decimal	INT
MyDM200:O.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 ResultsAck field has acknowledged them and been set. The DataMan reader will respond to the acknowledgement by clearing the ResultsValid bit. Once the ResultsAck 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.

Note:
You must manually add **UserData** to the output assembly by configuring the DataMan module in RSLogix 5000. Perform the following steps on a CompactLogix or ControlLogix PLC:

1. Right click the DataMan module and select **Properties**.
2. Under **Module Definition**, click **Change**.
3. The Module Definition window pops up. Under the **Output Data to Sensor** drop-down menu, select **SINT–484**.
4. Click OK. RSLogix 500 is now updating the Module Definition.

The output assembly controller tags will now list UserData as part of the output assembly.

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

3. The TriggerReady tag changes to 1.
4. 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.

5. 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:I	{...}		CC:DataMan...
− MyDM200:I.Status	{...}		CC:DataMan...
MyDM200:I.Status.TriggerReady	1	Decimal	BOOL
MyDM200:I.Status.TriggerAck	0	Decimal	BOOL
MyDM200:I.Status.Acquiring	0	Decimal	BOOL
MyDM200:I.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.

6. 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

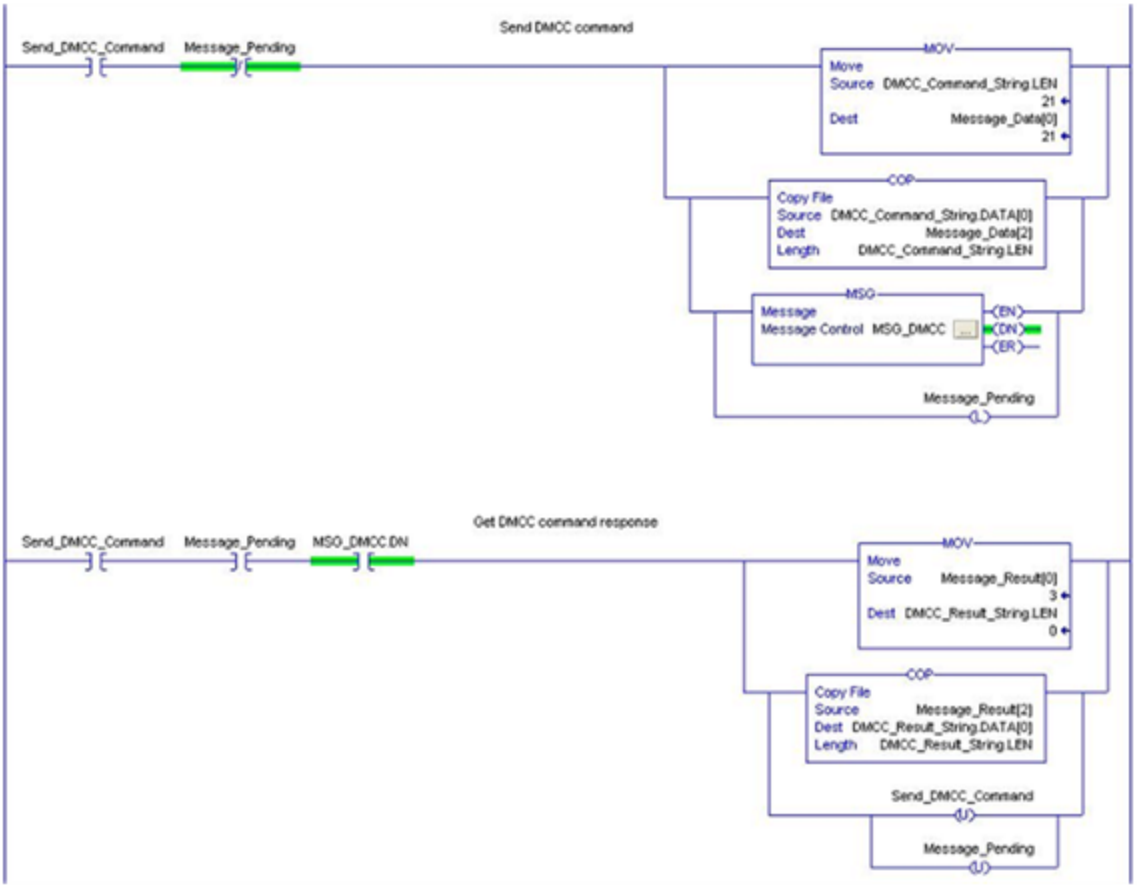
Message_Data: Temp buffer holding the data to send via the MSG instruction.

Message_Result: Temp buffer holding the data received via the MSG instruction.

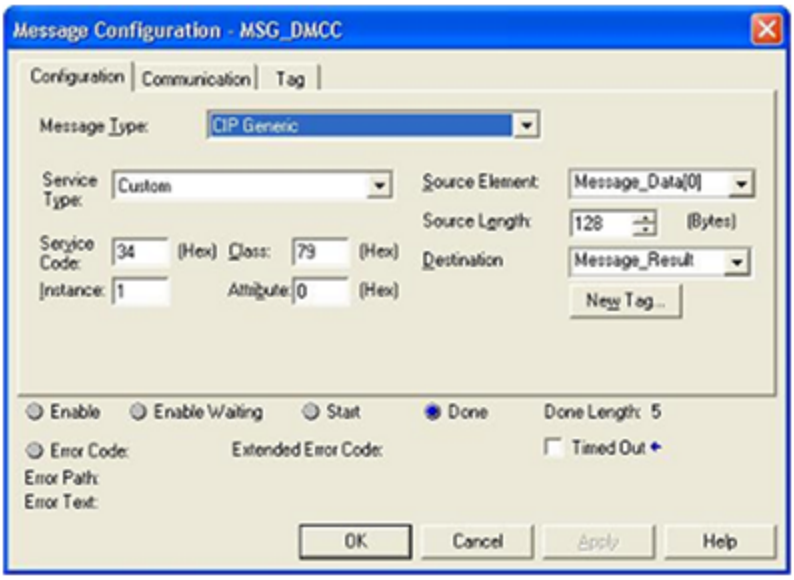
Message_Pending: Boolean flag used to indicate that a message is in process.

MSG_DMCC: Data structure required by the Logix MSG instruction.

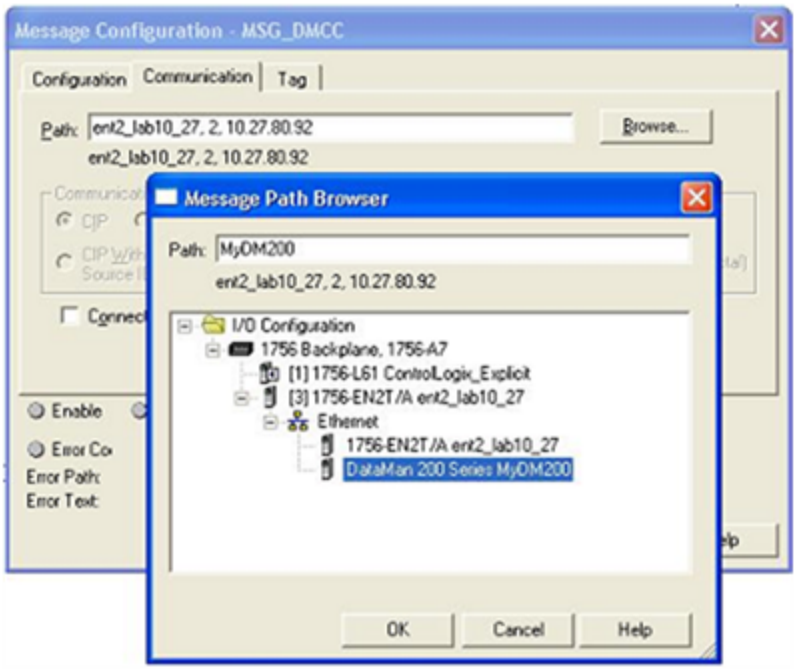
7. Add the following two rungs to the MainRoutine of your ControlLogix project:



8. 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”.



9. 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.



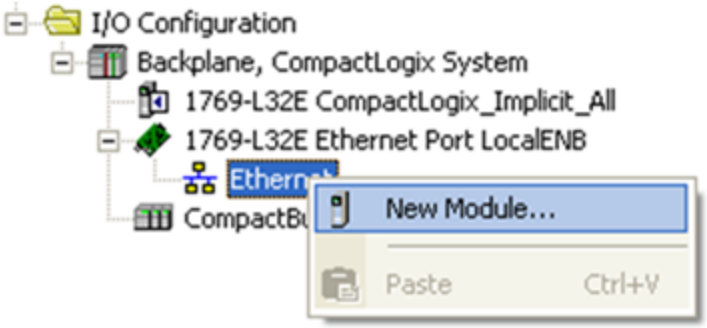
10. Download to the ControlLogix and place in “Run Mode”.
11. To operate:
    - Place a DMCC command in the “DMCC_Command_String” tag. For example “||>GET TRIGGER.TYPErl”. Note the rl at the end of the string. This is how Logix represents a CRLF.
    - Toggle the “Send_DMCC_Command” tag to 1.
    - When 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

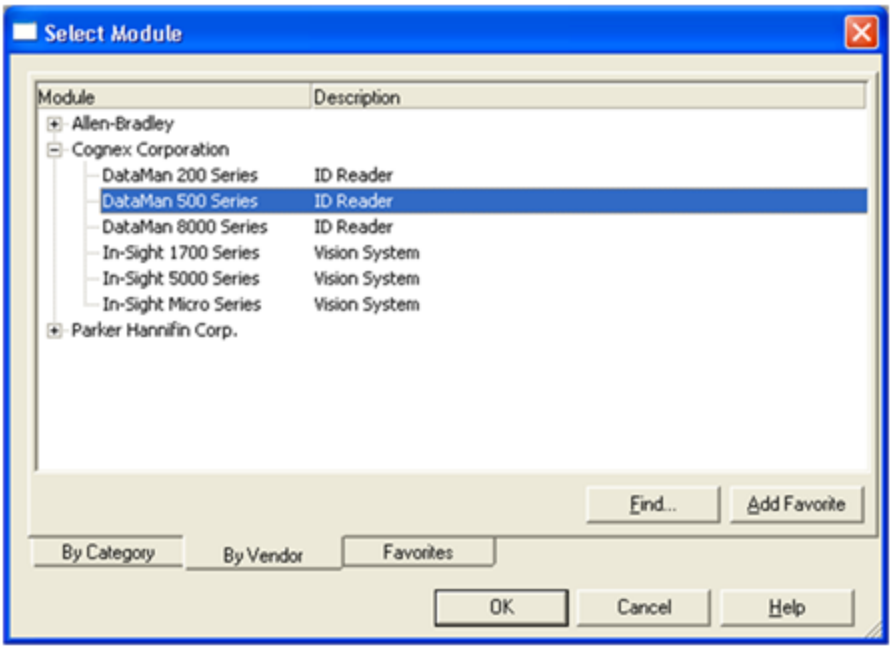
CompactLogix differs very little from ControlLogix in terms of programming. The ControlLogix examples apply equally to CompactLogix systems. There is only a slight difference in adding the DataMan device in the project I/O tree.

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.



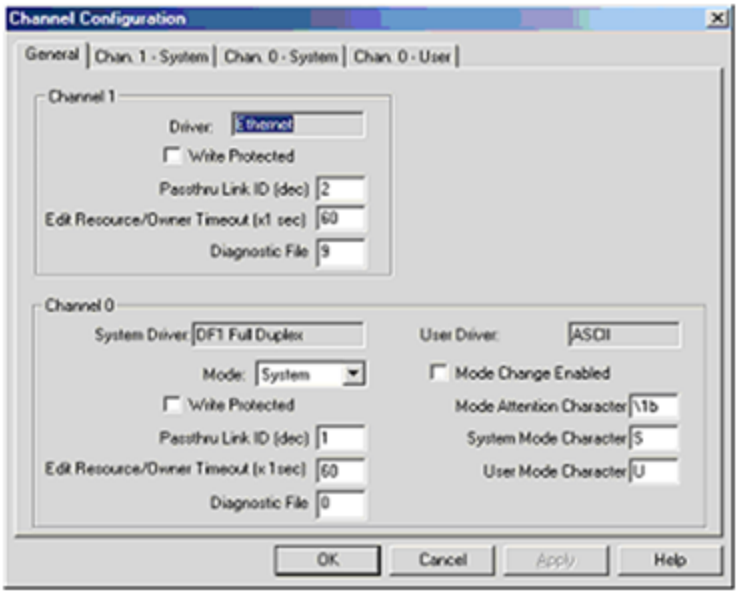
After the selection is made, the configuration dialog for the DataMan ID Reader system will be displayed. From this point on, configuration and programming are done exactly as shown in the ControlLogix section above.

Rockwell SLC 5/05 Examples

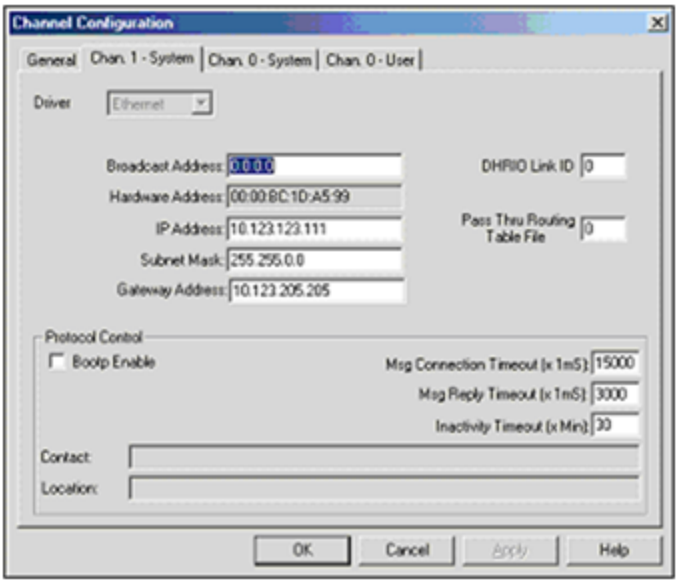
This section outlines a PCCC (PC^3) Communications configuration between a DataMan reader and the PLC. This example uses the Allen-Bradley SLC5/05 and Rockwell 500 software.

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.

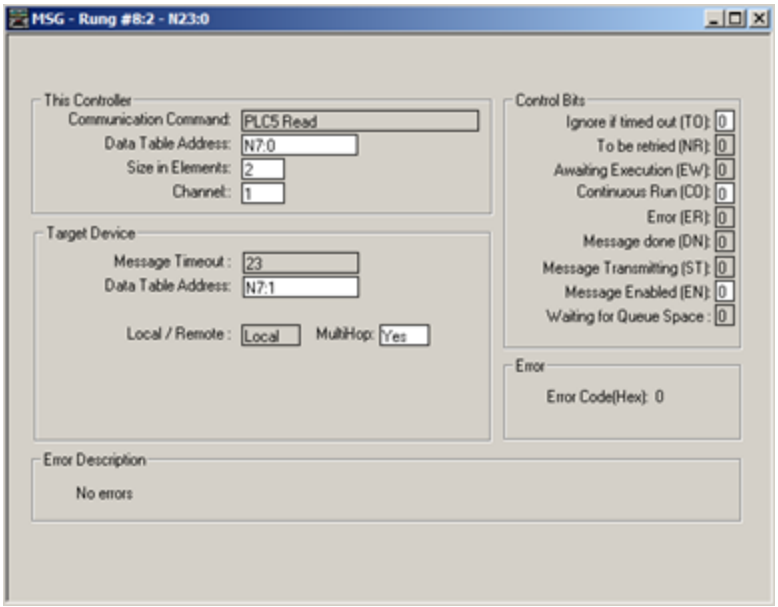
Message Instruction (MSG)

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 the ***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, 'N7:0', 'N' indicates the integer file, '7' indicates the file number 7, and '0' indicates the offset into that file (in this case, start at the 0th element). The figure below shows an example of the Integer Table accessed from the RSLogix 500 main screen.
- **Size in Elements**: This is the number of elements (or individual data) to read. In this example, two elements are being read.
- **Channel**: Depends on the configuration of the SLC. In the SLC, Channel 1 is the Ethernet port.
- **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 from 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.

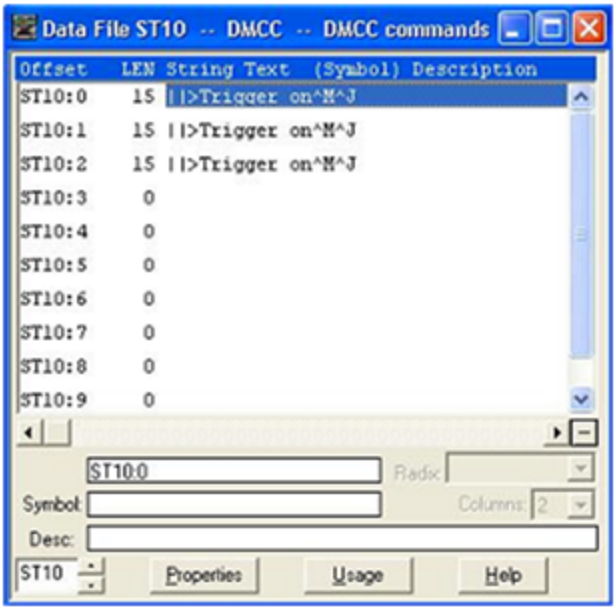
Note: The ST10:0 destination address is a special case used for sending DMCC commands to a DataMan reader. Any string sent to ST10:0 will be interpreted as a DMCC command.

- **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 5/0

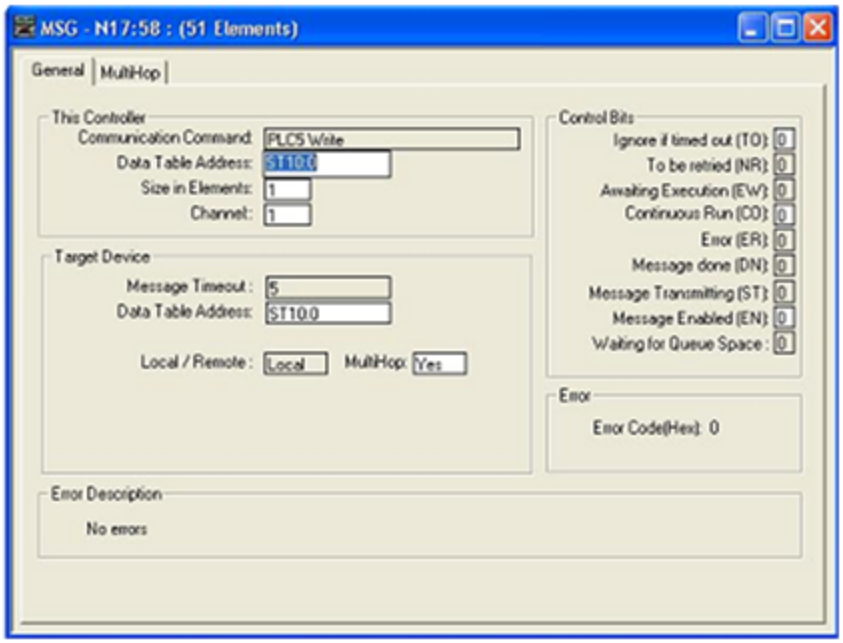
1. Configure the SLC5/05 as necessary.
2. Create a String Table that will hold your DMCC commands.

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.

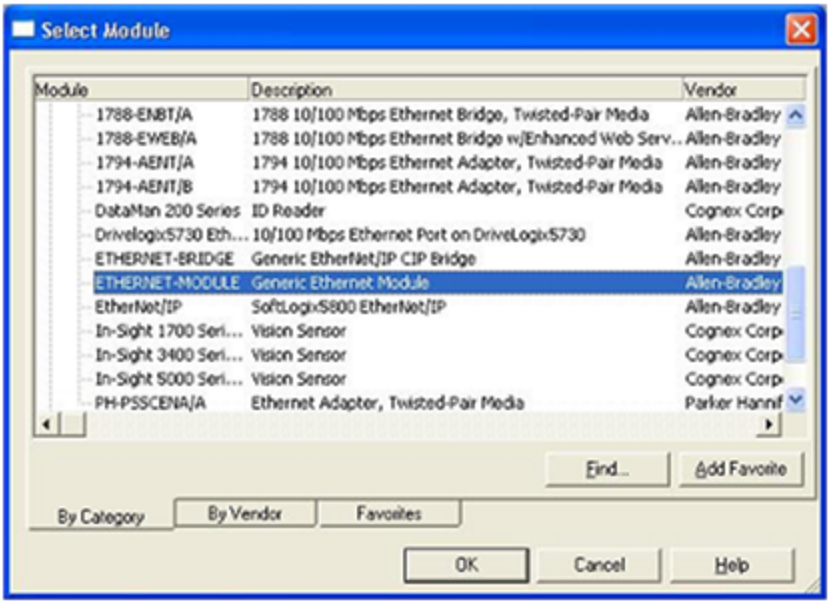
Establishing a Generic Implicit Messaging Connection

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.

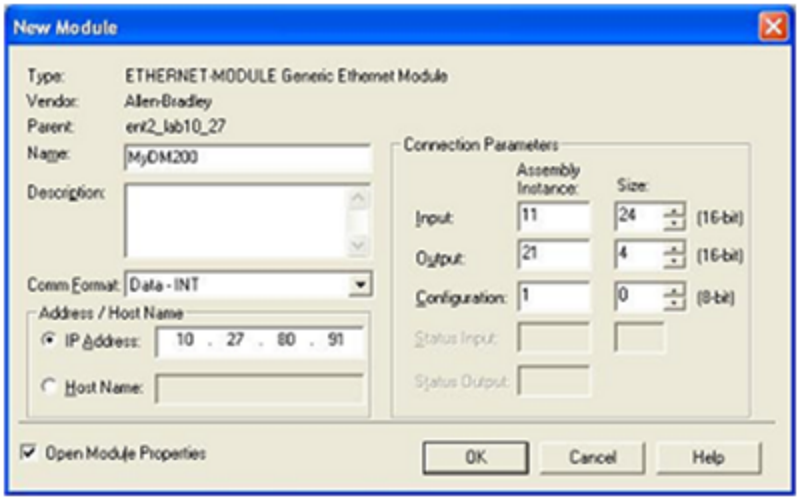
To establish a generic implicit messaging connection with a ControlLogix PLC:

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:

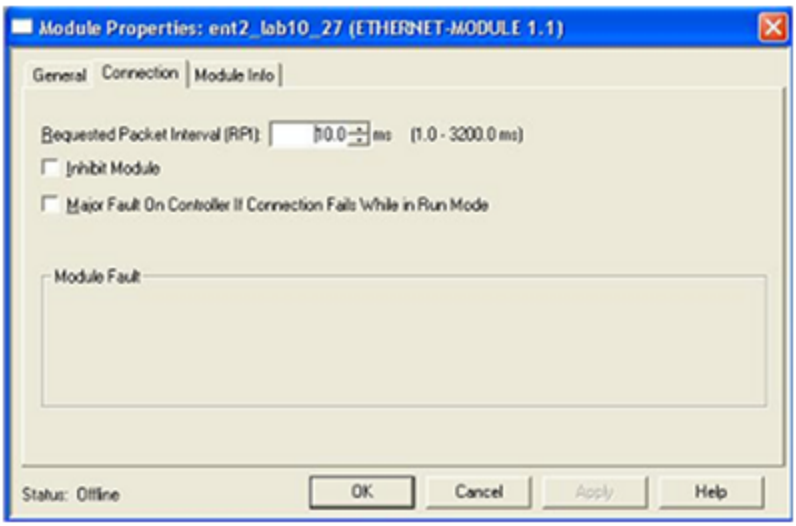
3. From the Select Module dialog, choose the Allen-Bradley Generic Ethernet Module.



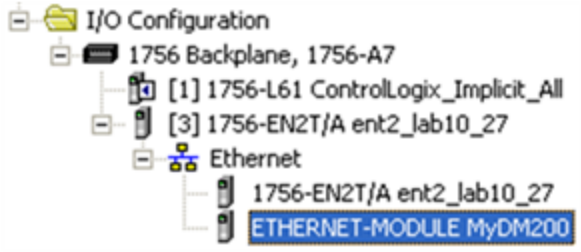
4. After the selection is made, the configuration dialog for the Generic Ethernet Module will be displayed. Configure the following:
    - Give the module a name.
    - Enter your DataMan's IP address.
    - Set the Comm Format to "Data – INT". This tells the module to treat the data as an array of 16-bit 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).



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 no case should it be set to lower than ½ the median scan rate of the PLC ladder program. Setting it lower wastes bandwidth and does not improve processing performance.

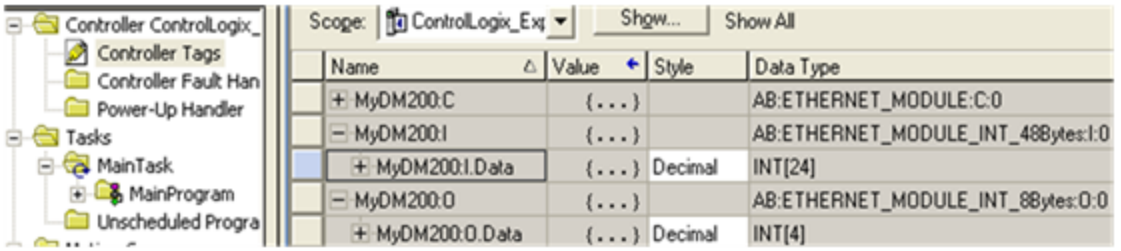


6. After adding the generic module to ControlLogix, the I/O tree should appear as follows.



7. 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.



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 Ouput 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	1	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.

SLMP Protocol

The SLMP 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 SLMP Protocol on Ethernet only.

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

Note: If you have a wireless DataMan reader, read the section Industrial Protocols for the Wireless DataMan.

Note: The SLMP Protocol can be used at the same time as iQss Protocol and CC-Link Protocol.

DMCC

The following commands can be used to enable/disable SLMP 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 SLMP-PROTOCOL.ENABLED ON
||>CONFIG.SAVE
||>REBOOT
```

Disable:

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

Reader Configuration Code

Scanning the following reader configuration codes will enable/disable SLMP Protocol for your corded reader.

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

Enable:

Disable:

Scanning the following reader configuration codes will enable/disable SLMP protocol on your DataMan 8000 base station.

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

Enable: 

Disable: 

Setup Tool

SLMP Protocol can be enabled by checking Enabled on the Industrial Protocols pane's SLMP Protocol tab. Make sure to save the new selection by choosing "Save Settings" before disconnecting from the reader.

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

SLMP Protocol Scanner

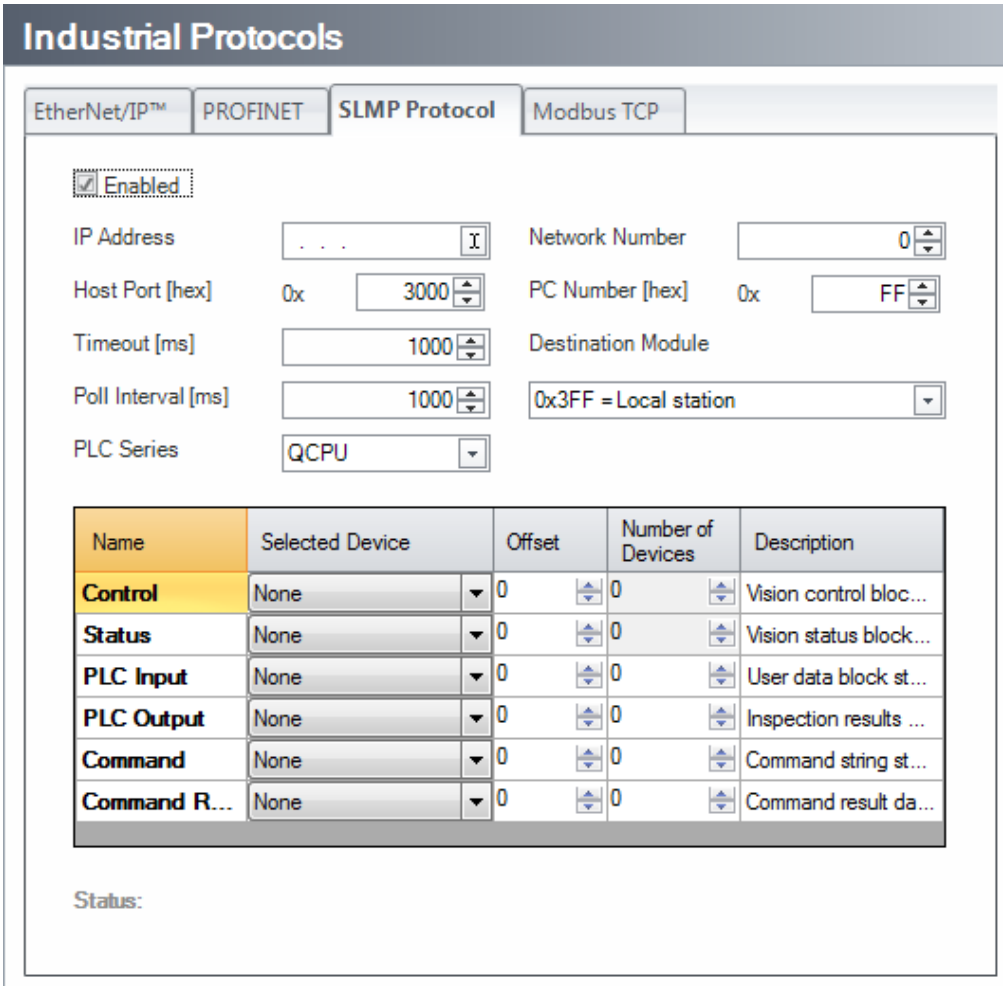
SLMP Protocol on DataMan is implemented as a client type device also referred to as a scanner. All communication is initiated by the DataMan reader in the form of read and write requests. The PLC acts as a passive server reacting to the read and write requests. Since the PLC cannot initiate communication, 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, SLMP 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 DataMan Setup Tool.
2. Under **Settings** -> **Communication Settings**, click **Industrial Protocols**.

3. Select the SLMP Protocol tab.



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

SLMP Protocol configuration consists of two aspects; defining the network information and defining the data to be exchanged. All configuration parameters are accessed via the SLMP 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 that you select **System -> Save Settings** to save any changes made to the SLMP 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

Name	Default	Range	Description
Host Port (Hex)	3000	Any Hex port number 1000-FFFF	Port number of the SLMP Protocol channel on the PLC
Timeout (ms)	1000	5 - 30000	Time to in milliseconds for a response from the PLC to an SLMP 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	SLMP 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.
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)

Block Name	Supported Device Types	Offset	Number of Devices
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 SLMP Protocol. The interface model consists of 6 data blocks grouped in 3 logical pairs:

- Control and Status
- PLC Input and PLC Output
- Command and Command 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.Control Block

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 *Trigger* bit. Clear this field to disable the network triggering mechanism.
1	Trigger	Setting this bit triggers an acquisition. **Note**: The *Trigger Ready* bit must be set high before triggering an acquisition.
2	Buffer Results Enable	When this bit is set, each read result set *(ResultID*, *ResultCode*, *ResultLength* and *ResultData* fields) will be held in the *Output Block* 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 (*ResultID*, *ResultCode*, *ResultLength* and *ResultData* fields). When the reader sees this bit transition from 0->1 it clears the *ResultsAvailable* 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 *UserData* is available. After reading the new *UserData* the reader sets Set *UserDataAck* 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 *StringCommand* is available. After processing the command the reader sets *StringCmdAck* 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 *SoftEventAck* 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
Soft Event Ack 7	Soft Event Ack 6	Soft Event Ack 5	Soft Event Ack 4	Soft Event Ack3	Soft Event Ack 2	Soft Event Ack 1	Soft Event Ack 0

Status Block Field Descriptions

Bit	Name	Description
0	Trigger Ready	Indicates when the reader is ready to accept a new *Trigger*. The reader sets this bit when *TriggerEnable* has been set and the reader is ready to accept a new trigger.
1	Trigger Ack	Indicates when the reader recognizes that *Trigger* has been set. This bit will remain set until the *Trigger* 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 *ResultsAck* 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 *TriggerEnable* is set low and then high again.
16	Set User Data Ack	Set to indicate that the reader has received new *UserData*. Bit will remain set until the corresponding *SetUserData* 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 *InitiateStringCmd* 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 *UserData* 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 ResultID of the corresponding read.
2	Result ID	Result set identifier. This is the value of TriggerID 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 ResultData 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 *StringCommand* 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 *Command Reference*, available through the Windows **Start** menu or the DataMan Setup Tool Help menu, for specific values.
1	Length	Number of bytes of valid data in the *StringCommand* field.
2..N	String Command Result	ASCII text string containing the command to execute. No null termination required.

Operation

SLMP 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.

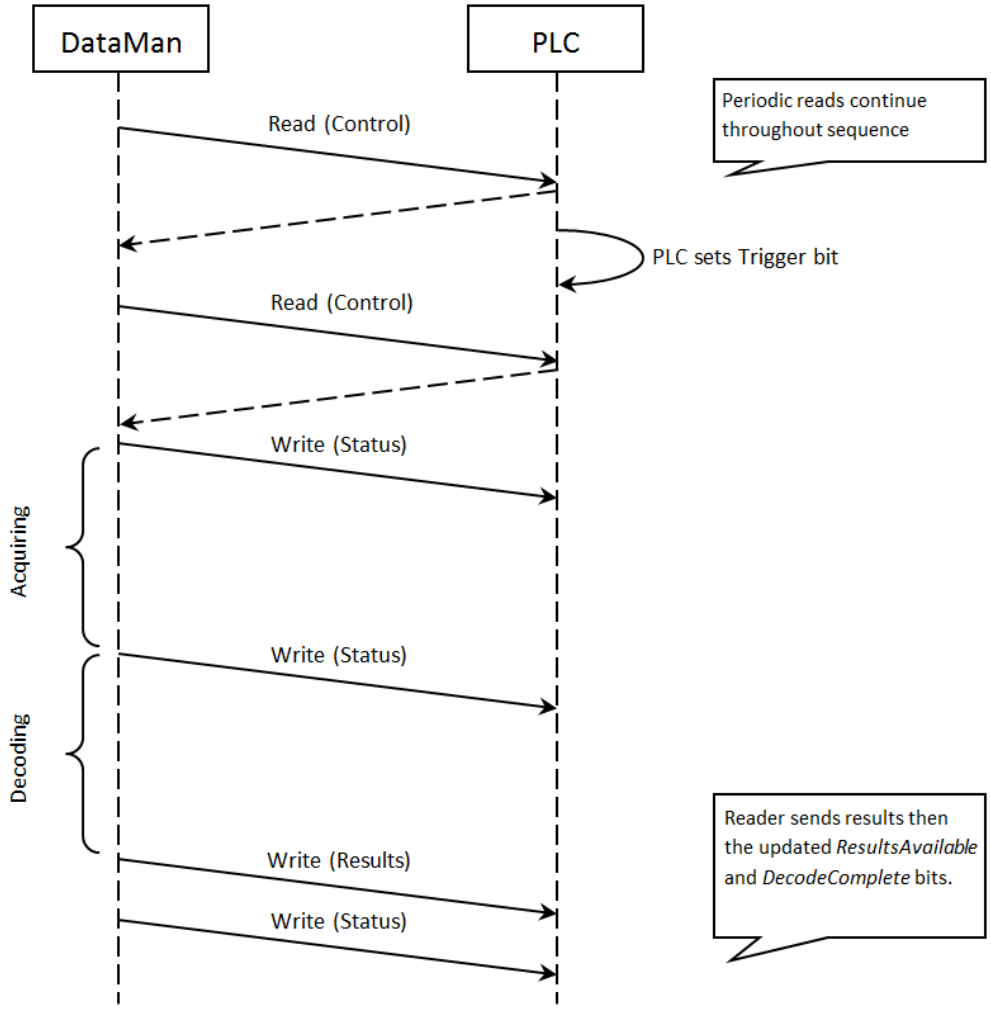
Scanning

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

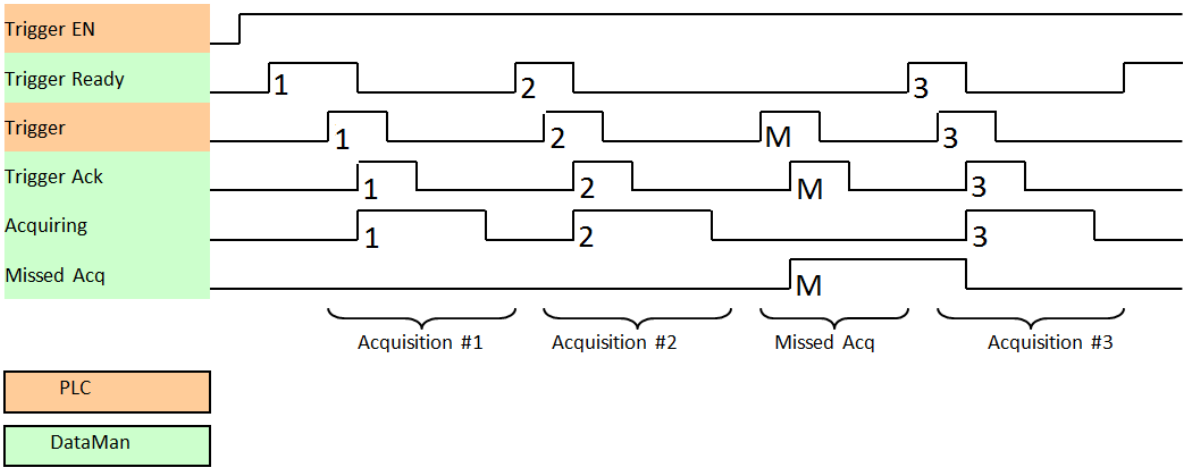
DataMan can be triggered to acquire images by several methods. It can be done via the SLMP Protocol by setting the *Trigger* bit or issuing a trigger String Command. It can also be done via DMCC command (Telnet) or hardwired trigger signal. The *Trigger* bit method will be discussed here.

On startup the *TriggerEnable* will be False. It must be set to True to enable triggering via the SLMP 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.

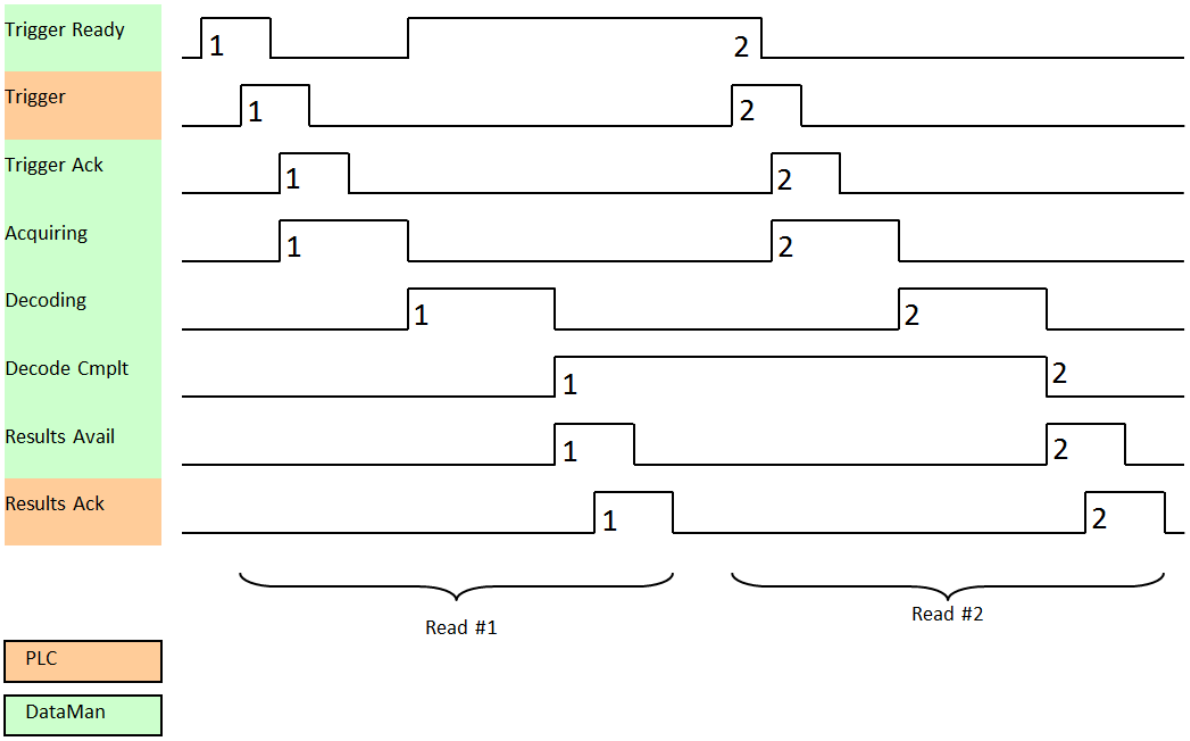
As a special case, an acquisition can be cancelled by clearing the *Trigger* signal before the read operation has completed. This allows for the cancellation of reads in Presentation and Manual mode if no code is in the field of view. To ensure that a read is not unintentionally cancelled, it is advised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

Note: If the queue has overflowed and then buffering is disabled, there will be a greater than 1 difference between the TriggerID and ResultID values. This difference represents the number of reads that had occurred but could not be queue because the queue was full (number of lost reads equals TriggerID - ResultID - 1). After the next read, the ResultID value will return to the typical operating value of TriggerID - 1.

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.

WARNING: 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.

Note: 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

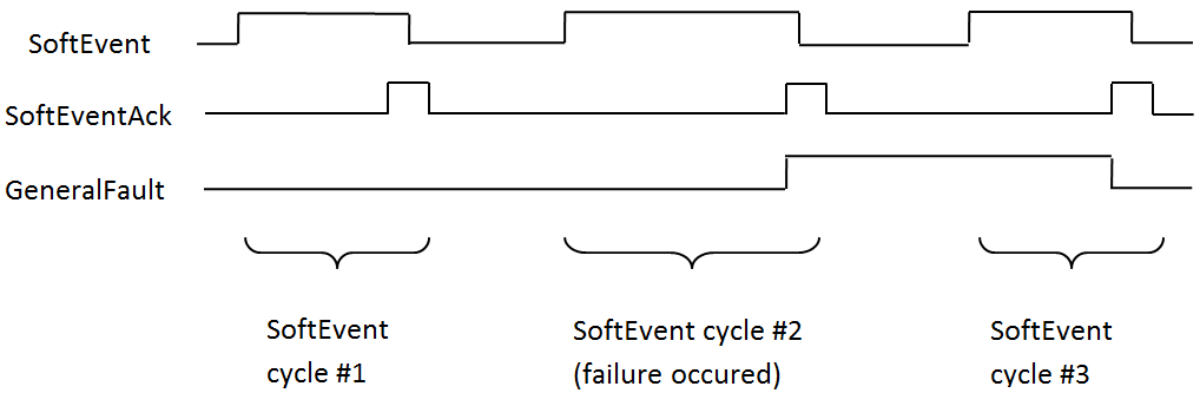
The DataMan SLMP Protocol implementation includes a String Command feature. This feature allows you to execute string-based DMCC commands over the SLMP protocol connection. The DMCC command is sent to the reader via the String Command block. The DMCC command result is returned via the String Command 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 SLMP 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 DataMan 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)

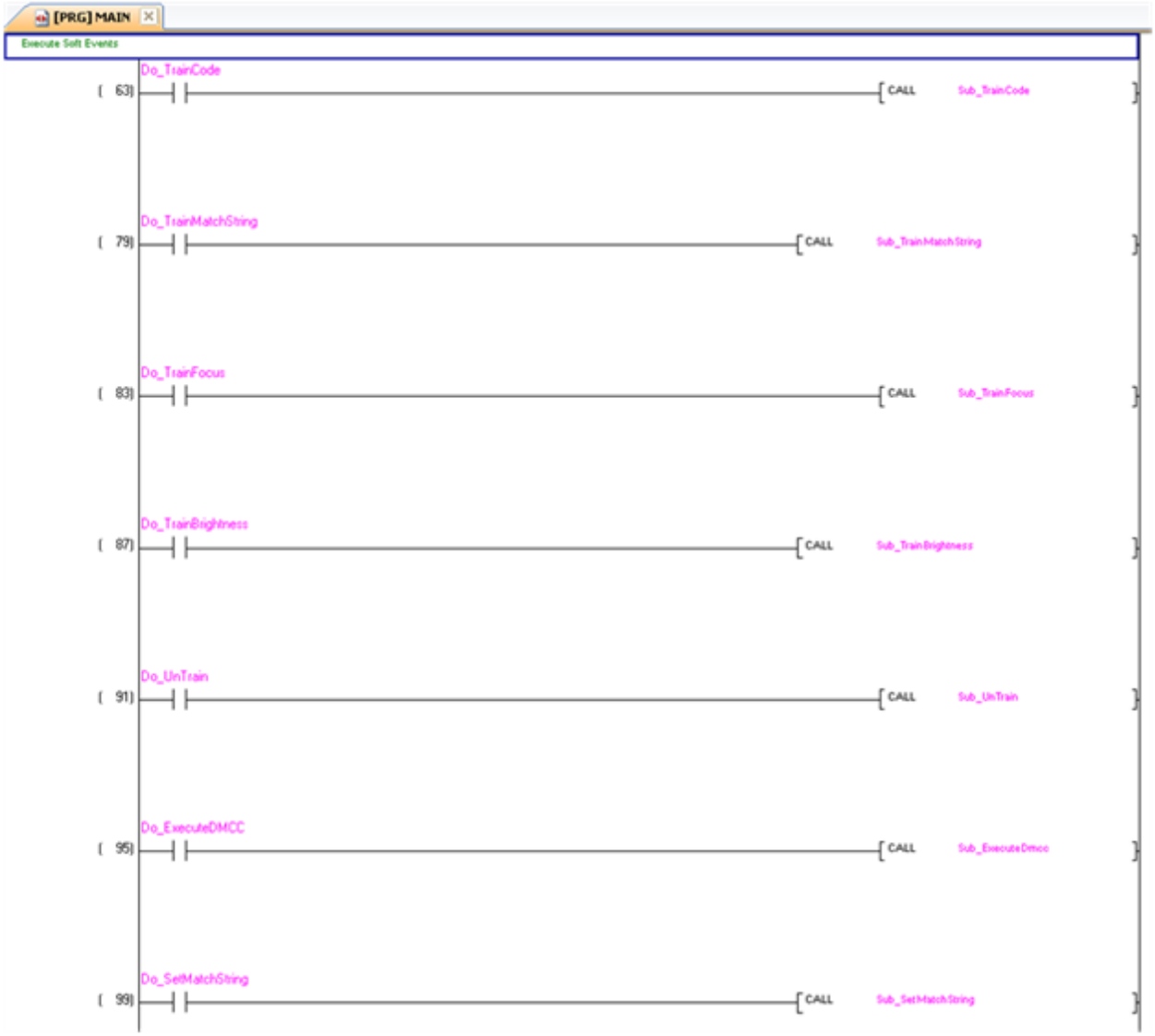
The string command feature provides a simple way to invoke DMCC commands from the PLC. The command format and command result format is exactly identical to that used for serial or Telnet DMCC operation.

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 or the DataMan Setup Tool **Help** 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”.

ModbusTCP

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.

Note: If you have a wireless DataMan reader, read the section Industrial Protocols for the Wireless DataMan.

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 for your corded reader.

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

Enable:

Disable:

Scanning the following reader configuration codes will enable/disable Modbus TCP for your DataMan 8000 base station.

Note: You must reboot the device for the change to take effect

Enable: 

Disable: 

Setup Tool

Modbus TCP can be enabled by checking **Enabled** on the Industrial Protocols pane's Modbus TCP tab. Make sure to save the new selection by choosing "Save Settings" before disconnecting from the reader.

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

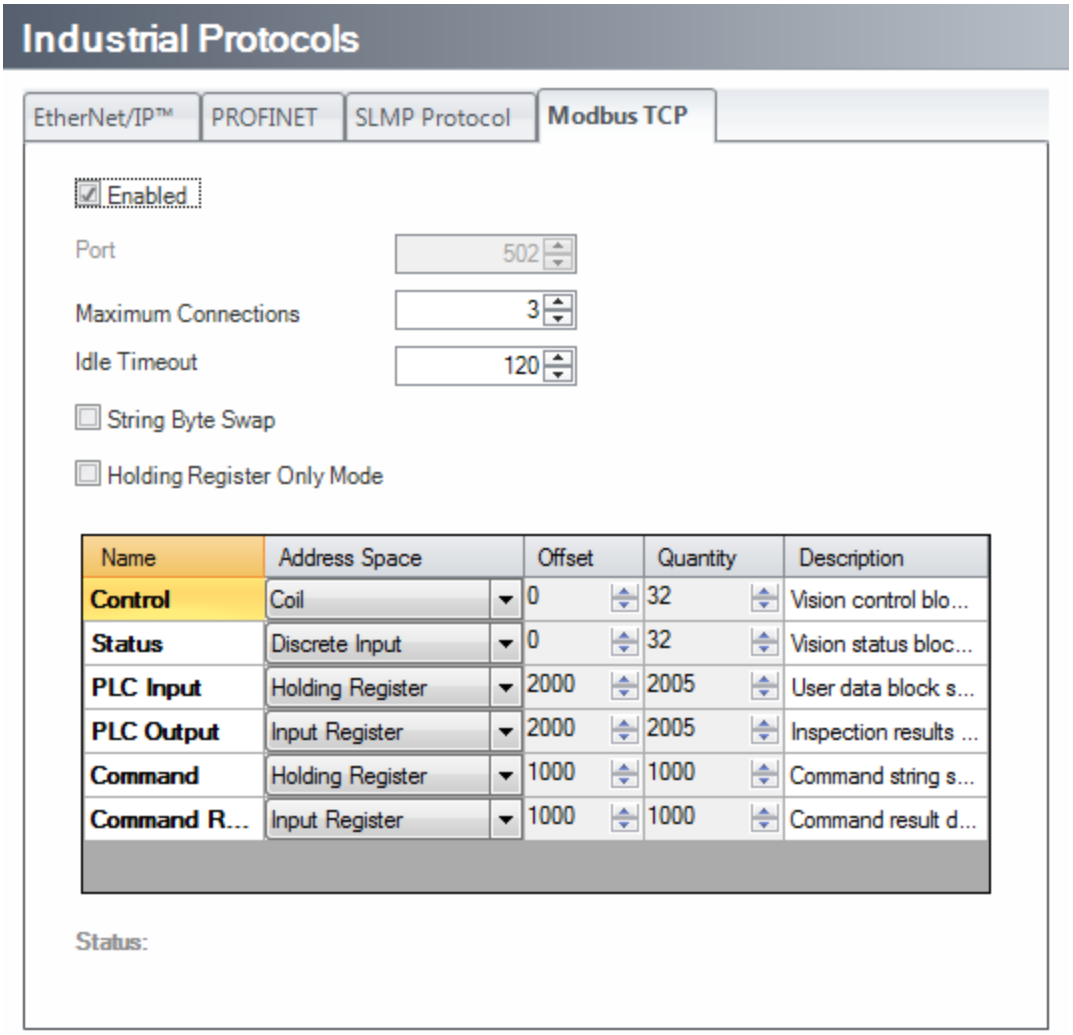
Modbus TCP Handler

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 DataMan 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 that 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.
Holding Register Only Mode	False	True / False	Holding Register Only Mode enable. If set to True, all data blocks will be mapped to the Holding Register space.

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 standard 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.

Standard Block Configuration

Block Name	Address Space	Offset	Schneider Form Addressing	Quantity
Control	Coil or Holding Register	0	000000 – 000031 400000 – 400001	32, if coil 2, if holding register
Status	Discrete Input or Input Register	0	100000 – 100031 300000 – 300001	32, if discrete input 2, if input register
PLC Input	Holding Register	2000	402000 – 404004	1 – 2005
PLC Output	Input Register	2000	302000 – 304004	1 - 2005
Command String	Holding Register	1000	401000 – 401999	1 - 1000
Command String Result	Input Register	1000	301000 – 301999	1 - 1000

Note: The 6 digit 0-based Schneider representation is used here.

DataMan also supports an alternate block configuration. A number of PLCs can only access the ModbusTCP 'Holding Register' address space. The alternate 'Holding Register Only' configuration exists to accommodate these PLCs. In this alternate configuration, all reader input and output data is mapped to the ModbusTCP 'Holding Register' address space. To enable this alternate mode, perform one of the following options:

- check the "Holding Register Only" box on the ModbusTCP configuration screen
- use the DMCC command, which switches on or off the alternate mode (Holding Register Only mode)

Enable:

```
||>SET MODBUSTCP-HOLDING-ONLY ON
```

Disable:

```
||>SET MODBUSTCP-HOLDING-ONLY OFF
```

- scan the Reader Programming Codes
    - for the wireless reader

Enable: 

Disable: 

- for the corded reader

Enable: 

Disable: 

Holding Register Only" Block Configuration

Block Name	Address Space	Offset	Schneider Form Addressing	Quantity
Control	Holding Register	0	400000 – 400001	2
Status	Holding Register	5000	405000 – 405001	2
PLC Input	Holding Register	2000	402000 – 404004	1 – 2005
PLC Output	Holding Register	7000	407000 – 409004	1 – 2005
Command String	Holding Register	1000	401000 – 401999	1 – 1000
Command String Result	Holding Register	6000	406000 – 406999	1 – 1000

Note: The 6 digit 0-based Schneider representation is used here.

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

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 *Trigger* bit. Clear this field to disable the network triggering mechanism.
1	Trigger	Setting this bit triggers an acquisition. Note that the *Trigger Ready* bit must be set high before triggering an acquisition.
2	Buffer Results Enable	When this bit is set, each read result (*ResultID*, *ResultCode*, *ResultLength* and *ResultData* fields) will be held in the *Output Block* 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 (*ResultID*, *ResultCode*, *ResultLength* and *ResultData* fields). When the reader sees this bit transition from 0à1 it clears the *ResultsAvailable* 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 *UserData* is available. After reading the new *UserData* the reader sets *SetUserDataAck* 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 *StringCommand* is available. After processing the command, the reader sets *StringCmdAck* 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 *SoftEventAck* 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. 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**
Soft Event Ack 7	Soft Event Ack 6	Soft Event Ack 5	Soft Event Ack 4	Soft Event Ack3	Soft Event Ack 2	Soft Event Ack 1	Soft Event Ack 0

Status Block Field Descriptions

Bit	Name	Description
0	Trigger Ready	Indicates when the reader is ready to accept a new *Trigger*. The reader sets this bit when *TriggerEnable* has been set and the reader is ready to accept a new trigger.
1	TriggerAck	Indicates when the reader recognizes that *Trigger* has been set. This bit will remain set until the *Trigger* 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 *ResultsAck* 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 *TriggerEnable* is set low and then high again.
16	Set User Data Ack	Set to indicate that the reader has received new *UserData*. Bit will remain set until the corresponding *SetUserData* 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 *InitiateStringCmd* 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 *UserData* 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 *ResultID* of the corresponding read.
2	Result ID	Result identifier. This is the value of *TriggerID* 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 *ResultData* field.
5...n	Result Data	Result data from this acquisition/decode. Formatted as ASCII text with two characters per 16-bit register. No terminating null character.

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 *StringCommand* 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 ***Command Reference***, available through the Windows **Start** menu or the DataMan Setup Tool **Help** menu, for specific values.
1	Length	Number of bytes of valid data in the *StringCommand* field.
2..N	String Command Result	ASCII text string containing the command to execute. No null termination required.

Operation

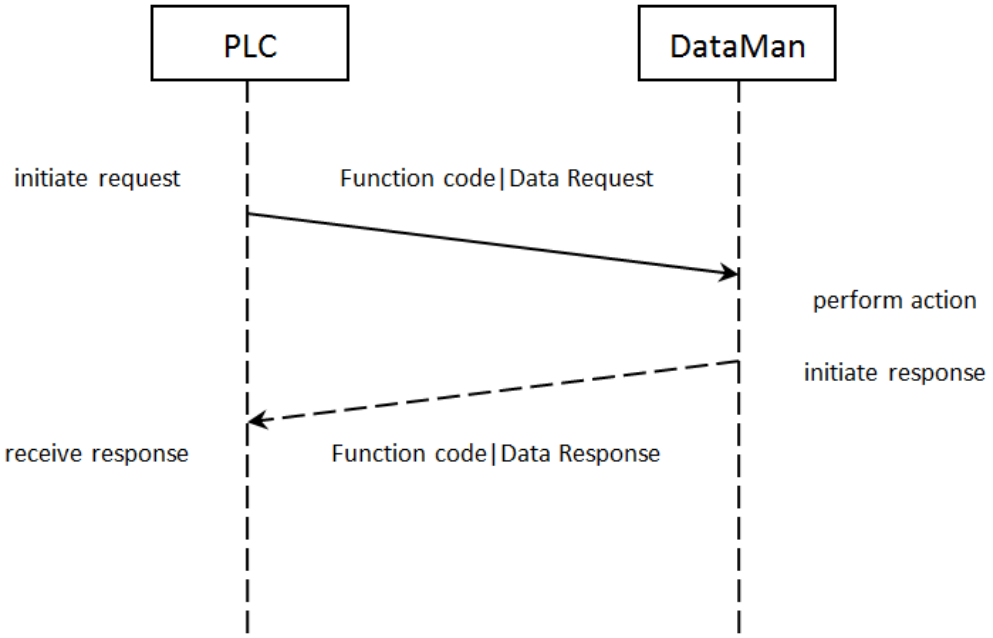
Modbus TCP is a request/response based protocol. All communications are originated from the PLC. The reader acts as server.

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

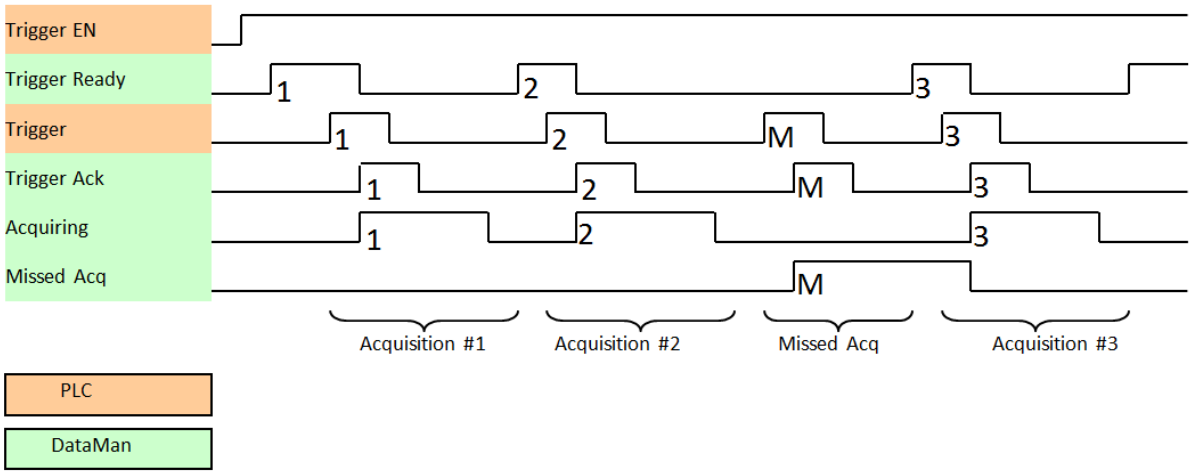
DataMan can be triggered to acquire images by several methods. It can be done by setting the *Trigger* bit or issuing a trigger String Command. It can also be done via DMCC command (Telnet) or hardwired trigger signal. The *Trigger* bit method will be discussed here.

On startup, *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).

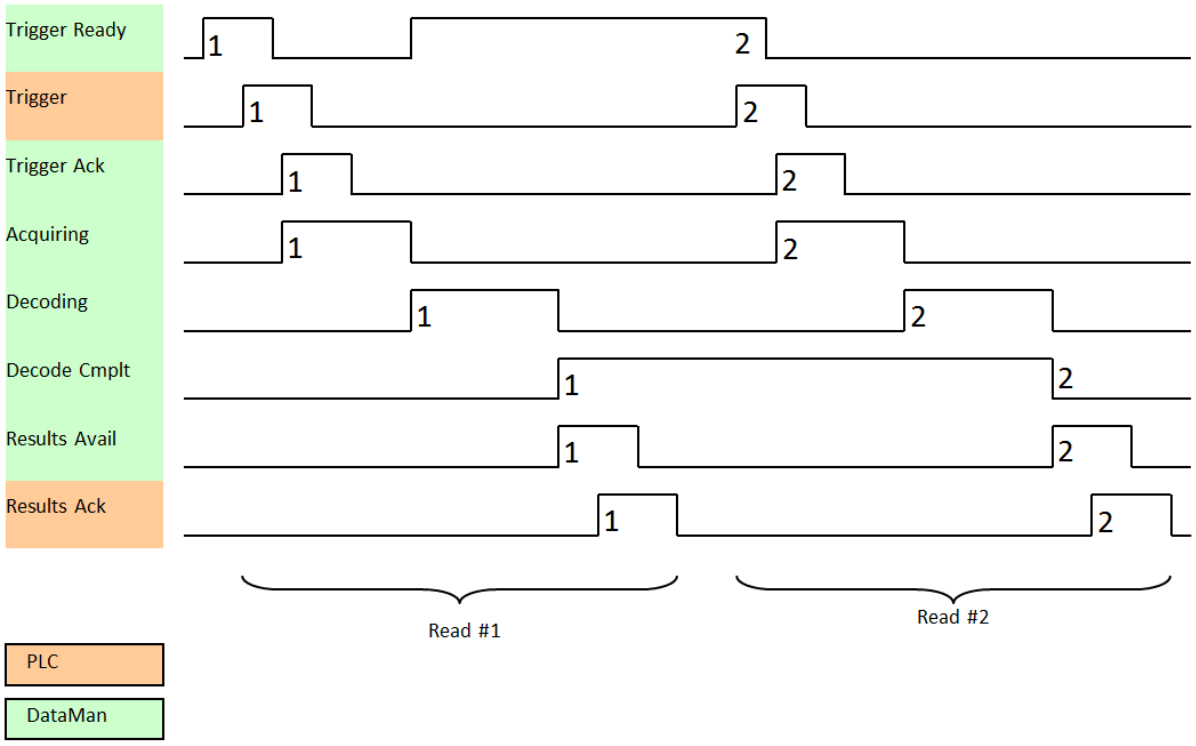
Decode / Result Sequence

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.

Note: If the queue has overflowed and then buffering is disabled, there will be a greater than 1 difference between the TriggerID and ResultID values. This difference represents the number of reads that had occurred but could not be queue because the queue was full (number of lost reads equals TriggerID - ResultID - 1). After the next read, the ResultID value will return to the typical operating value of TriggerID - 1.

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

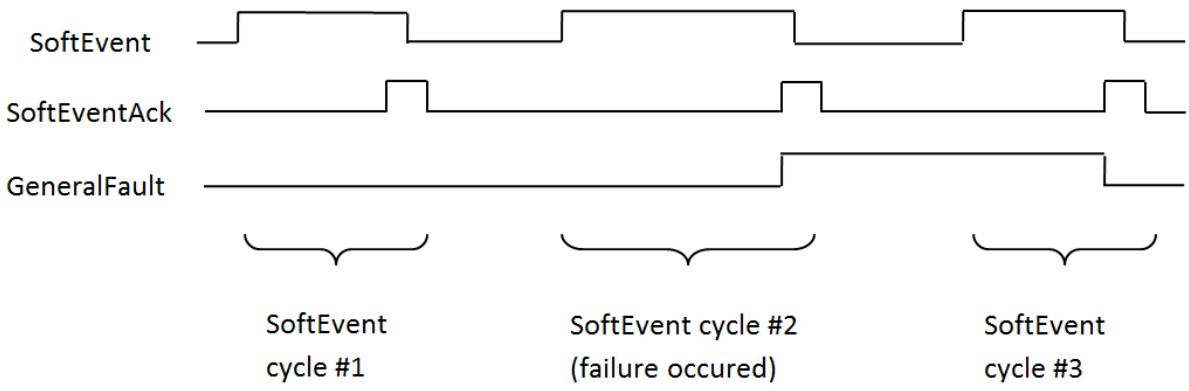
The DataMan Modbus TCP Protocol implementation includes a String Command feature. This feature allows you to execute string-based DMCC commands. The DMCC command is sent to the reader via the String Command block. The DMCC command result is returned via the String Command Result block. Initiating a command 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 DataMan 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 Contol 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 or the Setup Tool **Help** 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 DataMan Setup Tool.

Note: If you have a wireless DataMan reader, read the section Industrial Protocols for the Wireless DataMan.

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 that 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 for your corded reader.

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

Enable:

Disable:

Scanning the following reader configuration codes will enable/disable PROFINET for your DataMan 8000 base station.

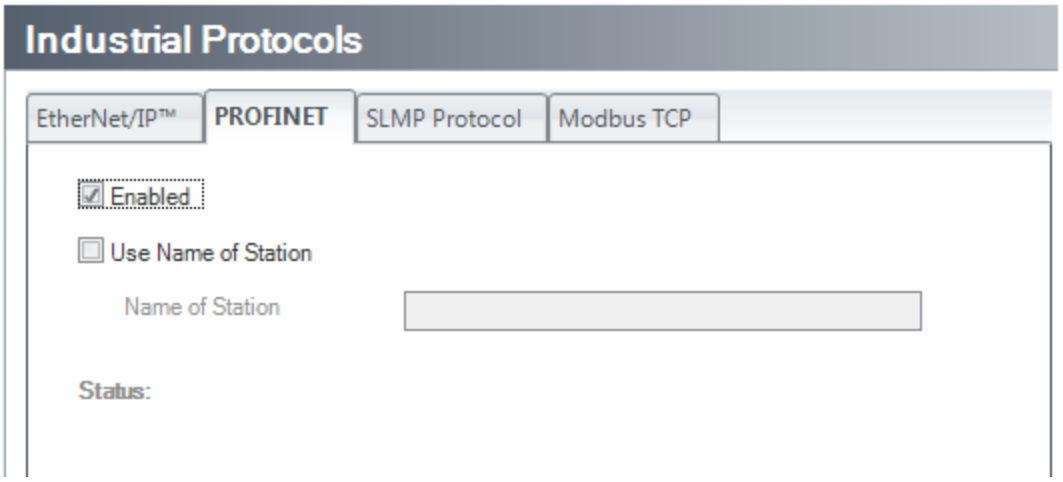
Note: You must reboot the device for the change to take effect

Enable:

Disable:

Setup Tool

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



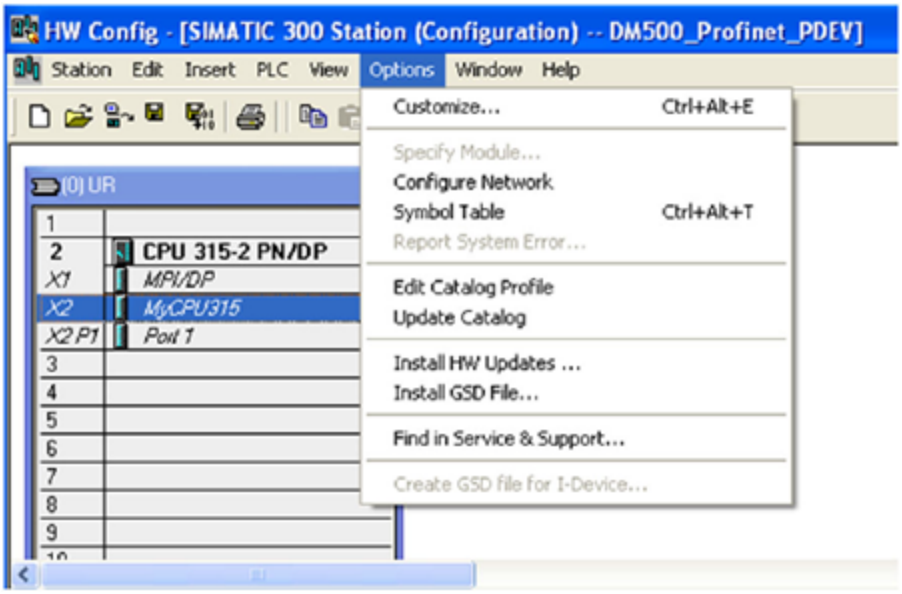
Getting Started

Preparing to use PROFINET involves the following main steps:

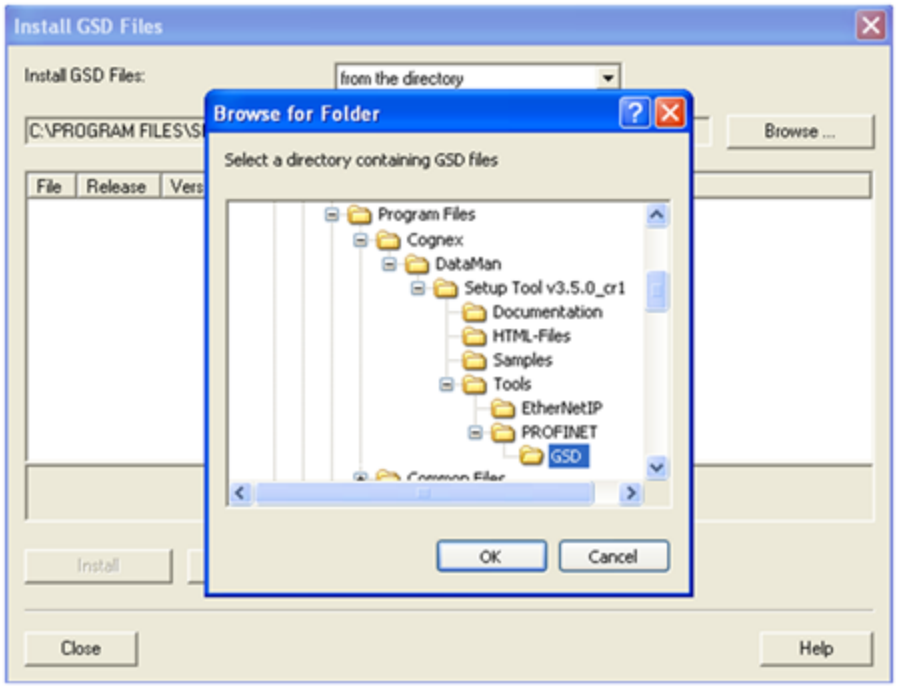
- Make sure that you have the Siemens Step 7 programming software (SIMATIC) installed.
- Set up the Siemens Software tool so that it recognizes your DataMan device.
- Install the Generic Station Description (GSD) file.

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…".

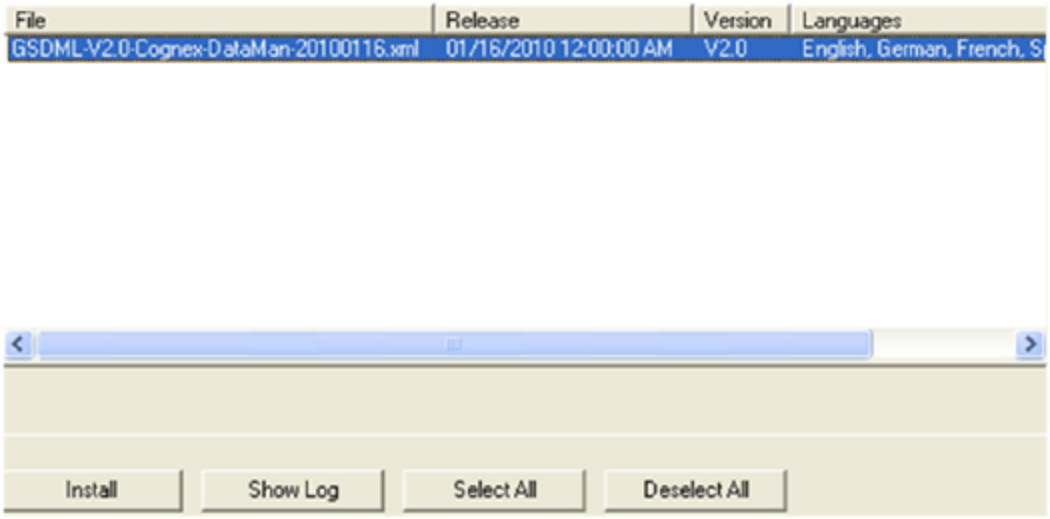


5. Browse to the location where the GSD file was installed (or the location where you saved the GSD file if it was downloaded from the web).

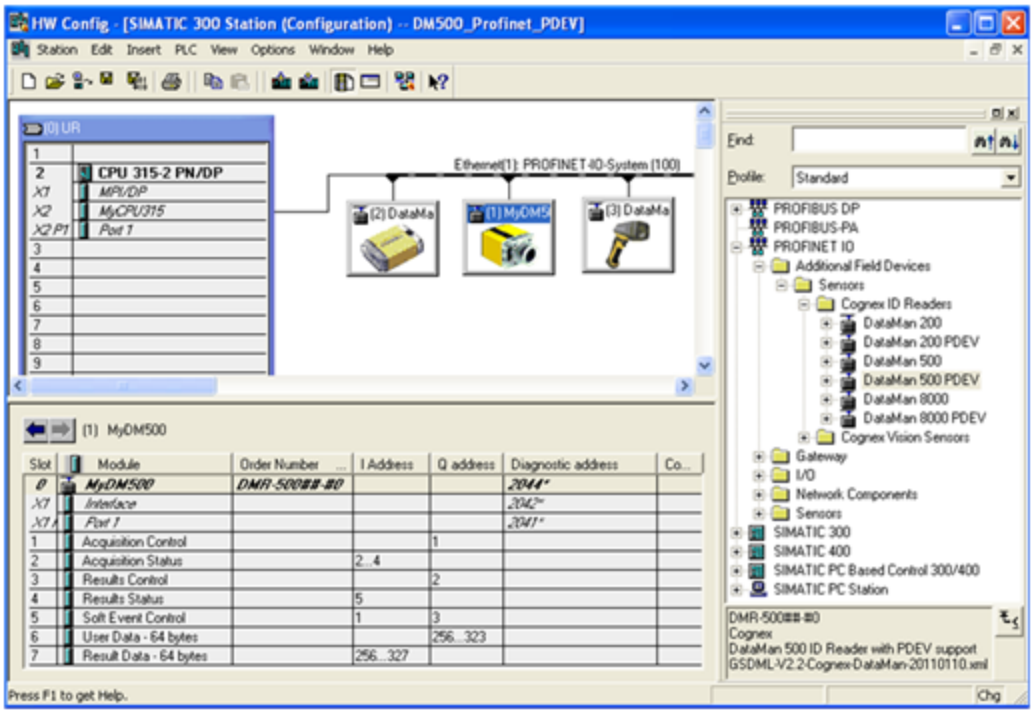


6. Select the GSD file you wish to install and follow the displayed instructions to complete the installation.

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.



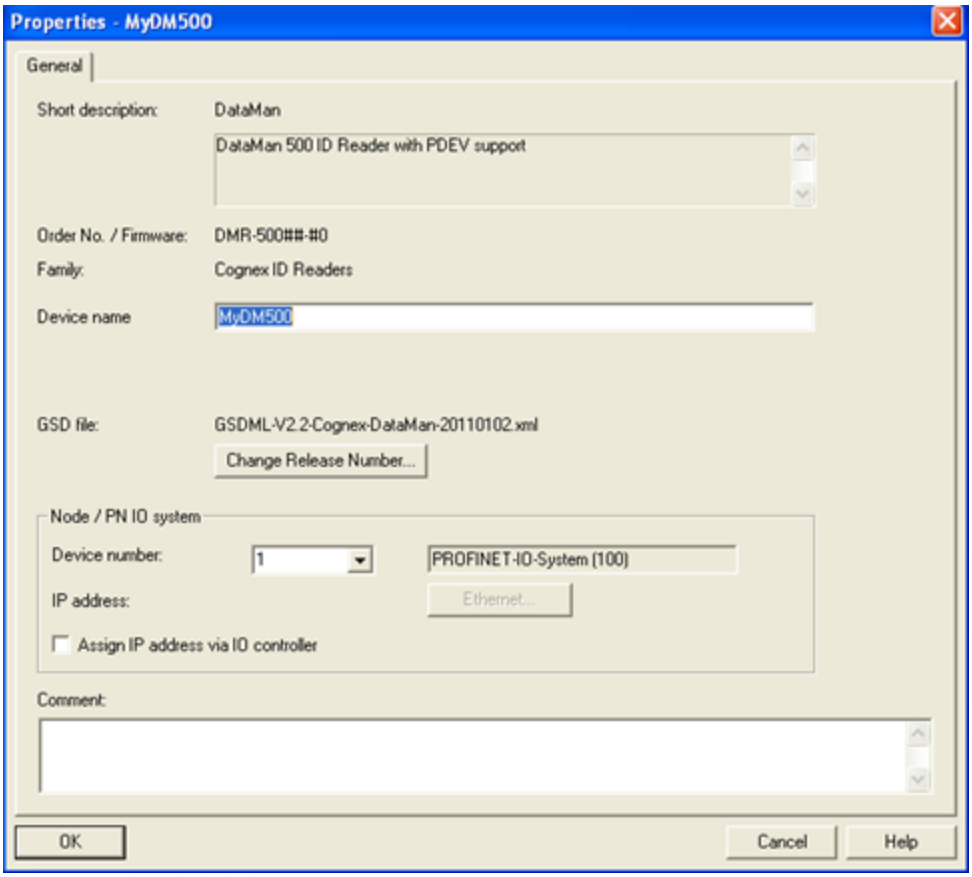
7. Add your DataMan device to your project. This makes the DataMan available in the Hardware Catalog. Launch the SIMATIC Hardware Config tool.
8. In the main menu, select View -> Catalog.
9. The catalog is displayed. Expand the “PROFINET IO” tree to the “Cognex ID Readers” node.
10. With the left mouse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



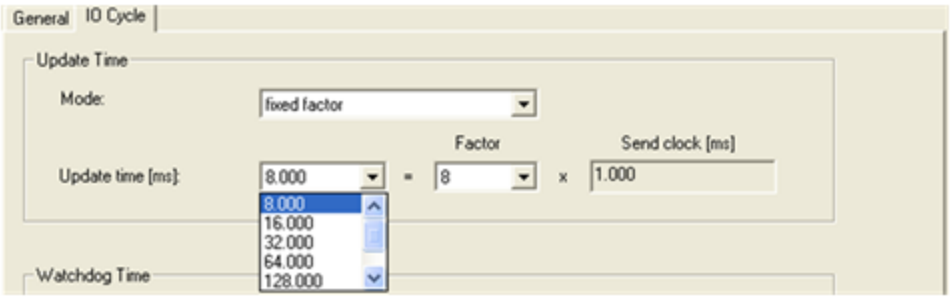
The HW Config tool automatically maps the DataMan I/O modules into the memory space.

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.

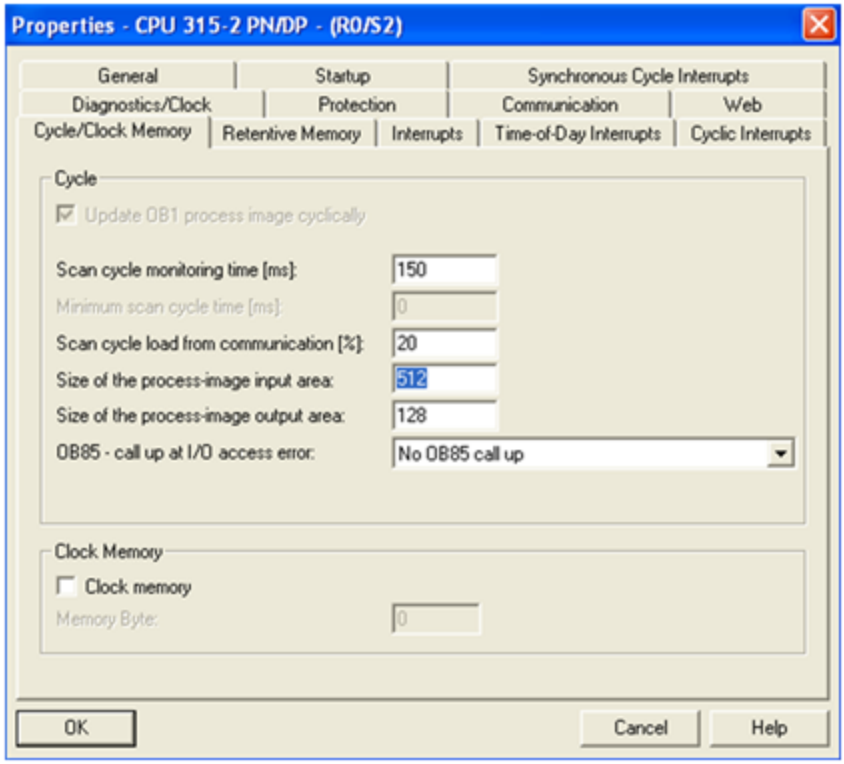


14. In the “IO Cycle” tab, select the appropriate cyclic update rate for your application.



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.

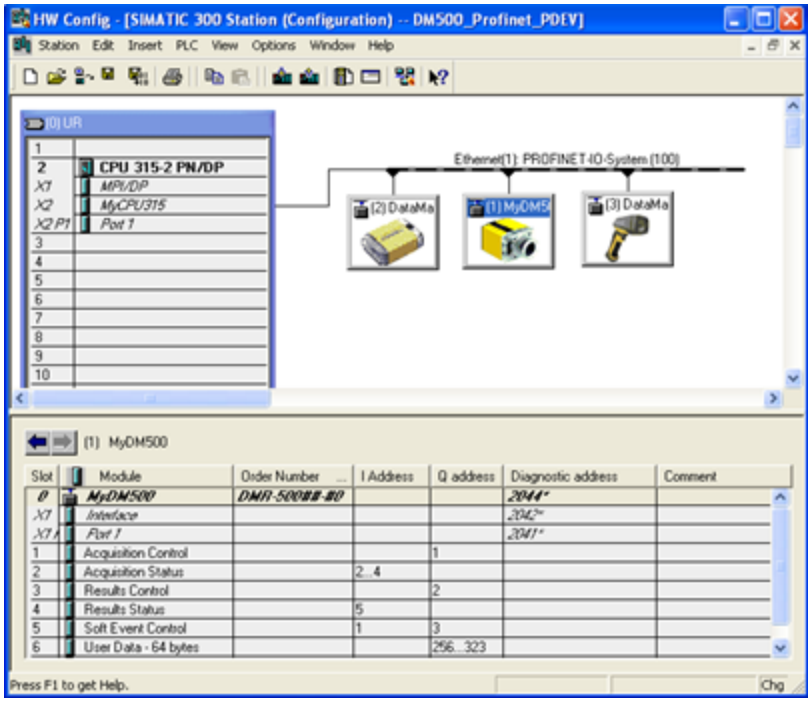


Note: Expanding the process image can have a performance impact on the PLC scan cycle time. If your scan time is critical, use the minimal acceptable module sizes and manually remap them down lower in the process image.

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: • 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	Reserved	.

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.

Note: The Missed Ack bit in the Acquisition Status Register will only be set if an acquisition triggered from the Acquisition Control Module could not get executed.

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.
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.
3	Train Brightness	Bit transition from 0 -> 1will 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.

Look into the Result Data Module for non-PROFINET related data if there was an acquisition problem on the reader. The "Result Code" part contains information about trigger overruns or buffer overflows that have occurred on the reader.

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).
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

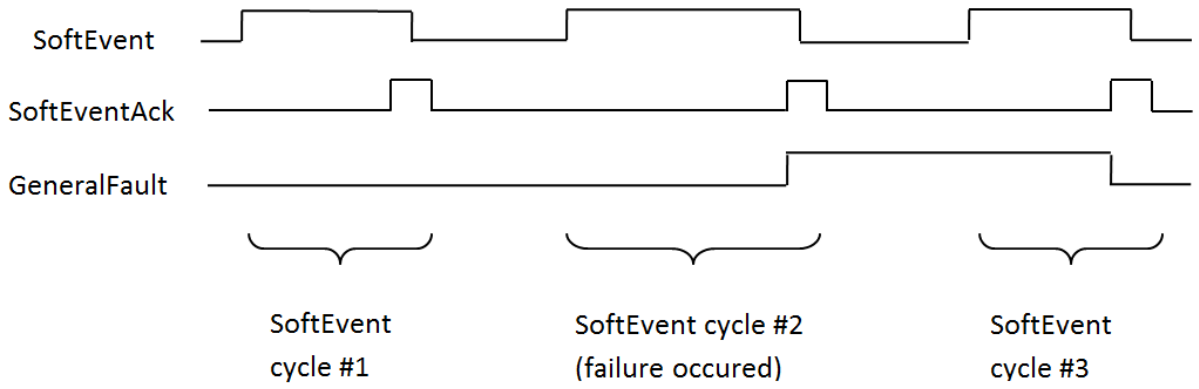
SoftEvents

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.

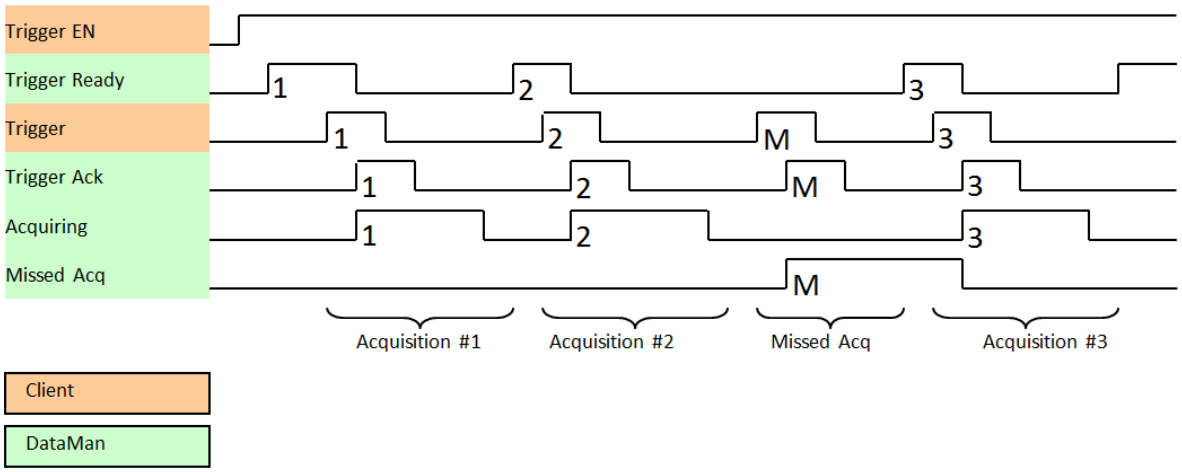
While the Trigger Ready bit is True, each time the reader sees the “Trigger” bit change from 0 to 1, it will initiate an image acquisition. The client (PLC) should hold the bit in the new state until that same state value is seen back in the Trigger Ack bit (this is a necessary handshake to guarantee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

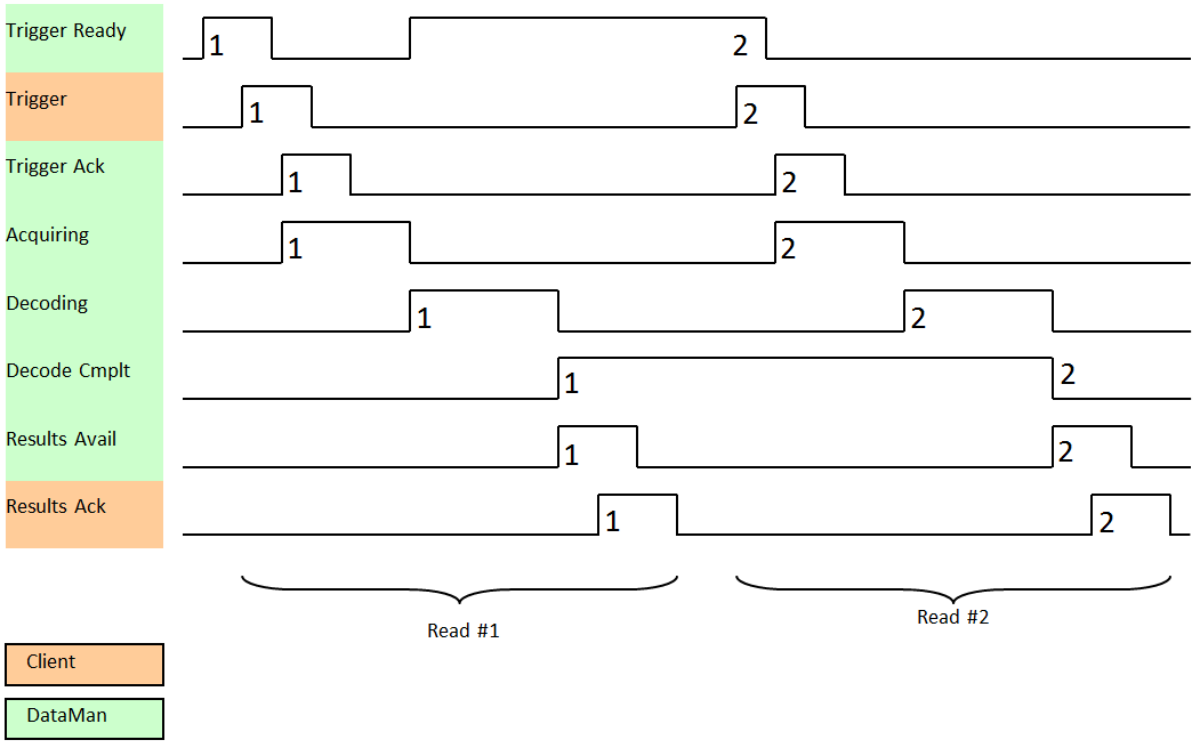
After an image is acquired it is decoded. While being decoded, the “Decoding” bit of the Result Status Module is set. When decode is complete, the Decoding bit is cleared and the “Decode Complete” bit is toggled.

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
0	Decoding	Set when decoding an image.	Set when decoding an image.
1	Decode Complete	Toggled on completion of an image decode.	Toggled on completion of an image decode.
2	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.
3	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.

Note: If the queue has overflowed and then buffering is disabled, there will be a greater than 1 difference between the TriggerID and ResultID values. This difference represents the number of reads that had occurred but could not be queue because the queue was full (number of lost reads equals TriggerID - ResultID - 1). After the next read, the ResultID value will return to the typical operating value of TriggerID - 1.

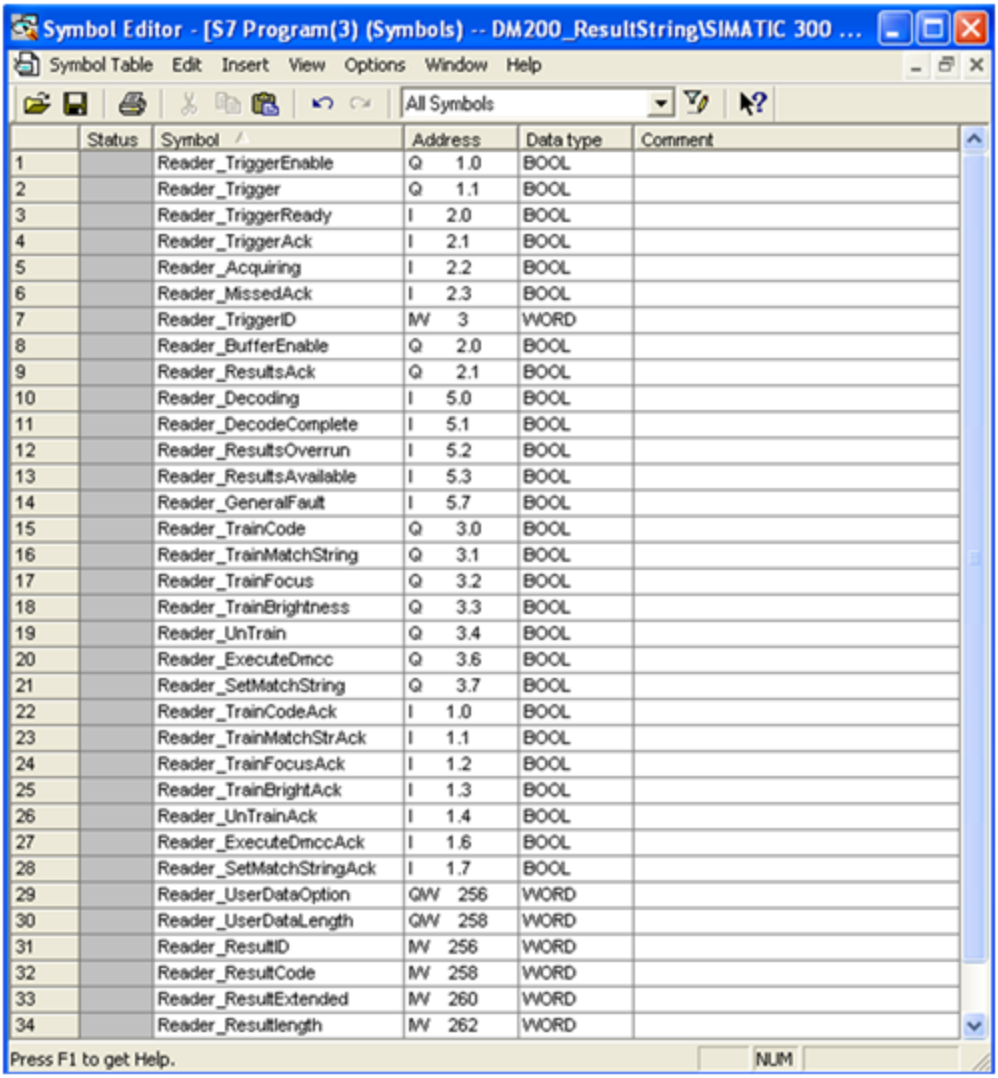
Siemens Examples

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	MW 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	MW 256	WORD	
32		Reader_ResultCode	MW 258	WORD	
33		Reader_ResultExtended	MW 260	WORD	
34		Reader_Resultlength	MW 262	WORD	

Trigger and Get Results

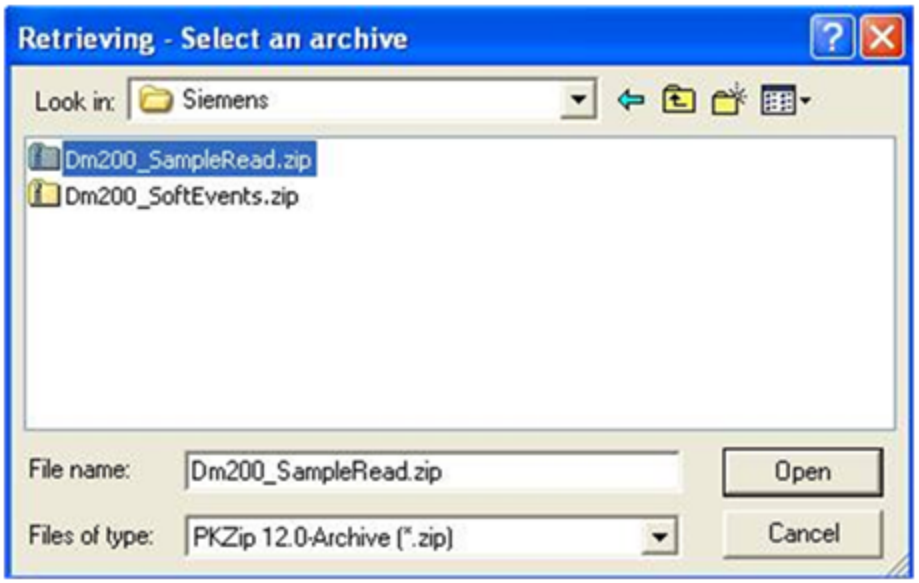
Run the sample program “DM200_SampleRead” for the complete example program.

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

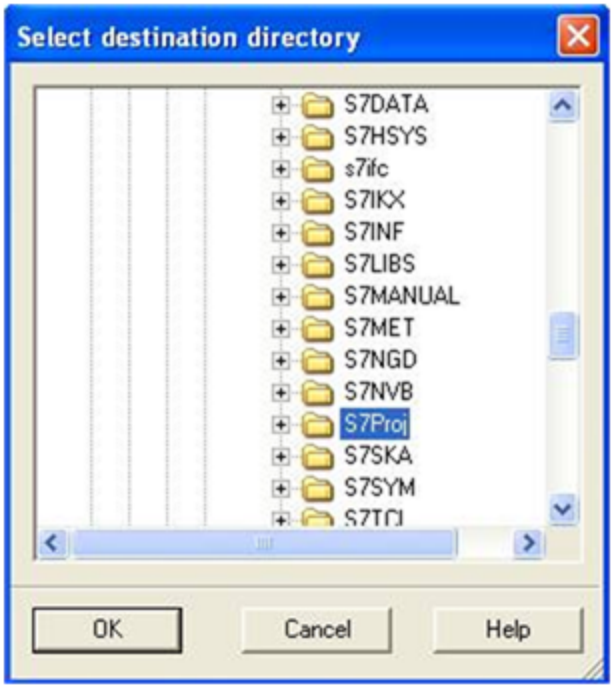
Perform the following steps to install the program:

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.

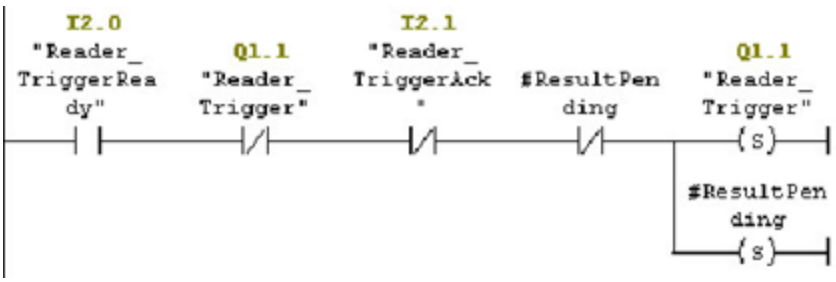


7. The Siemens software extracts the sample archive and makes it available.
8. Reduced to the basics, the process of reading and retrieving results consists of the following:
9. Define an area in your application to save read results. There are many options regarding how and where result data can be 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.

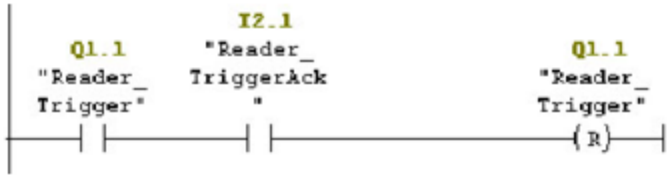
Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	ID	INT	0	ID of this read result
+2.0	Flags	INT	0	Flags indicating success or failure
+4.0	Length	INT	0	Number of data bytes in the array Value[]
+6.0	Value	ARRAY[1..63]		Code value read
*1.0		CHAR		
=70.0		END_STRUCT		

10. Enable the reader

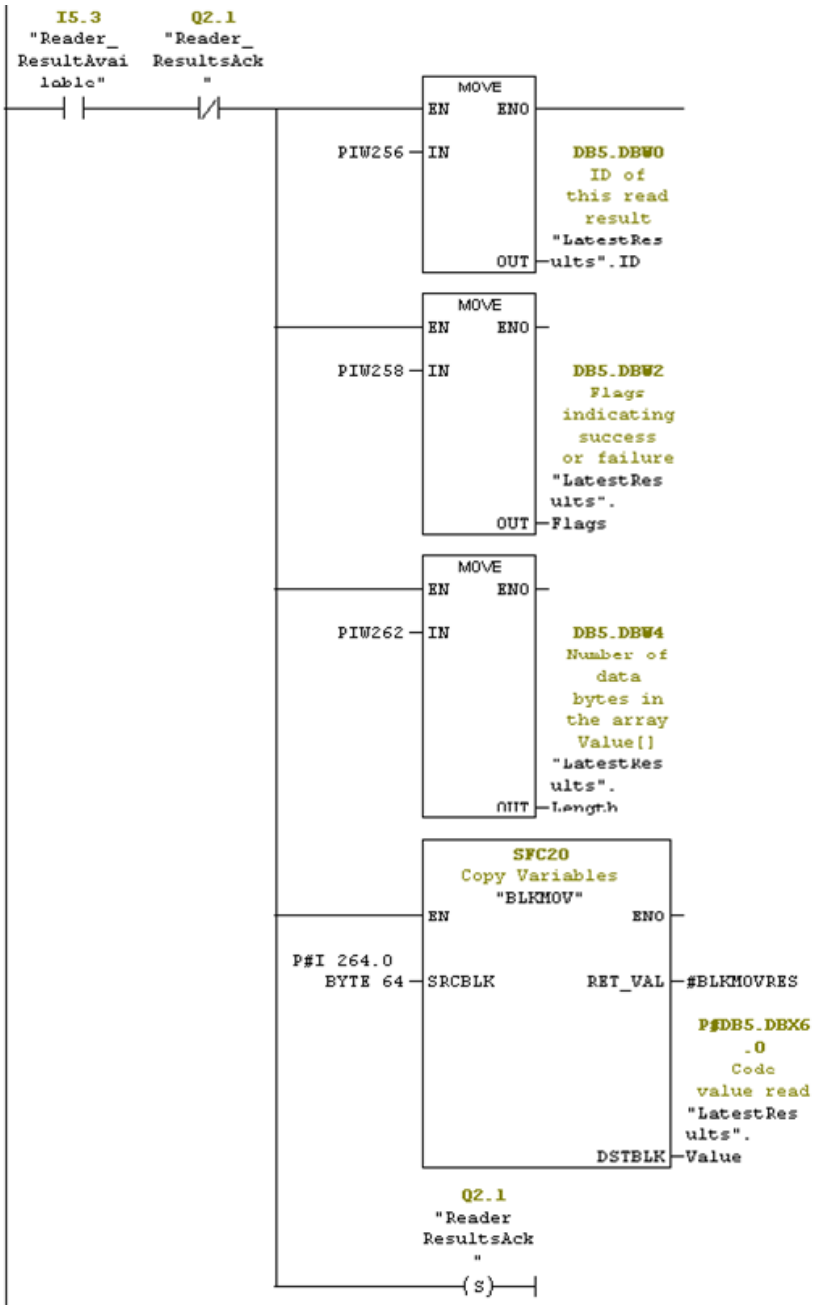
11. Set the trigger signal and set semaphore to indicate a read is pending.



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

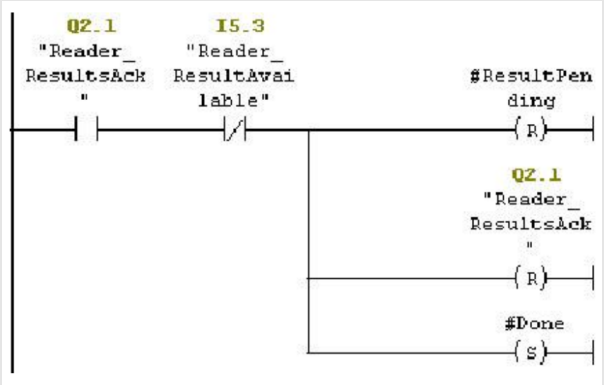


13. As soon as the results are available save a copy of the result data and set the results acknowledge signal.



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

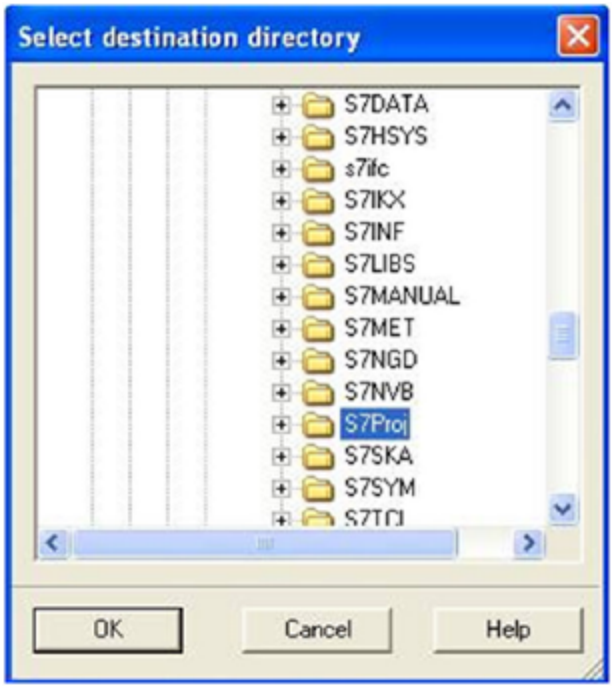
Run the sample program “DM200_SoftEvents” for the complete example program.

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.

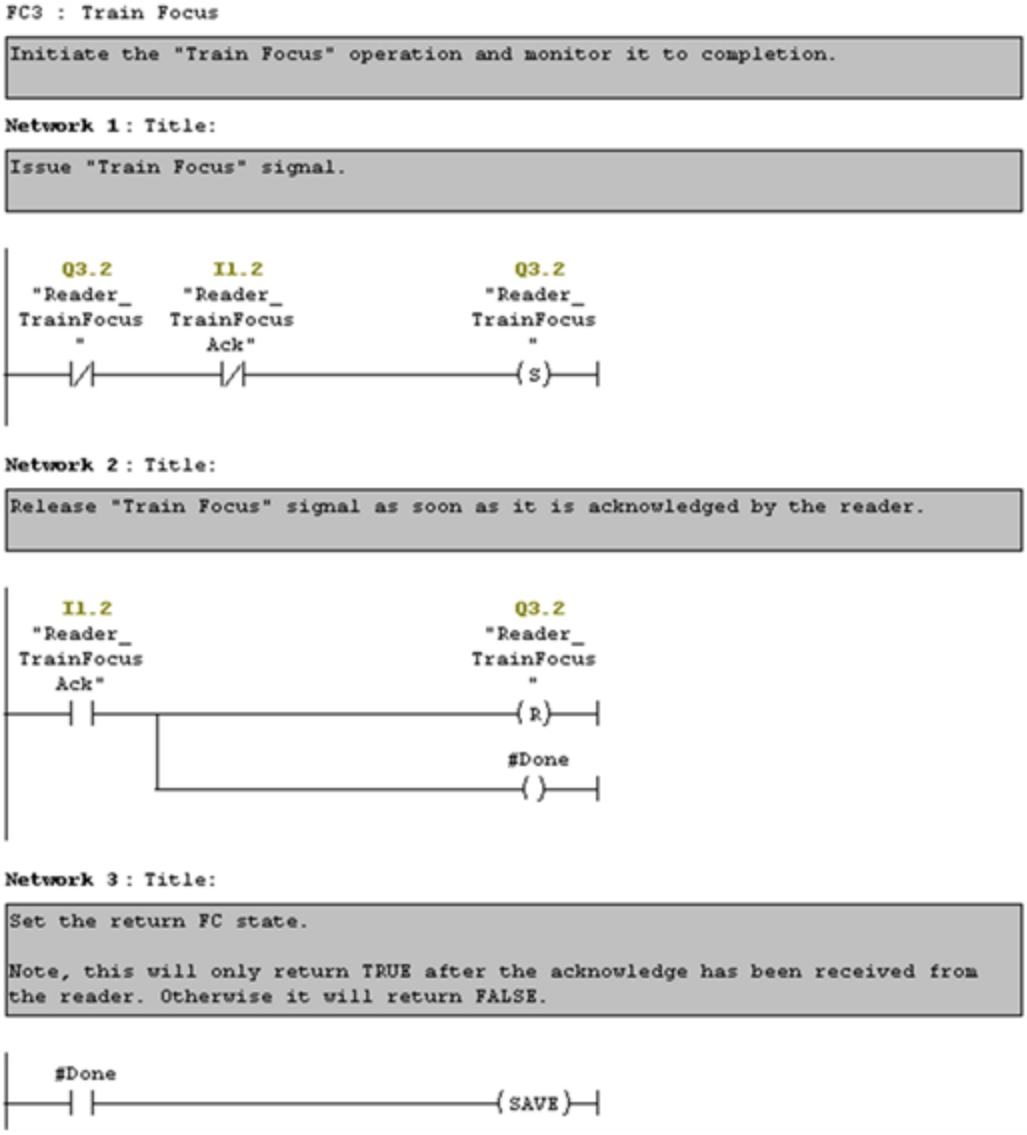
5. Look for the Siemens folder and select Dm200_SoftEvents.zip.
6. Select a destination directory to save the project on your PC.



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:



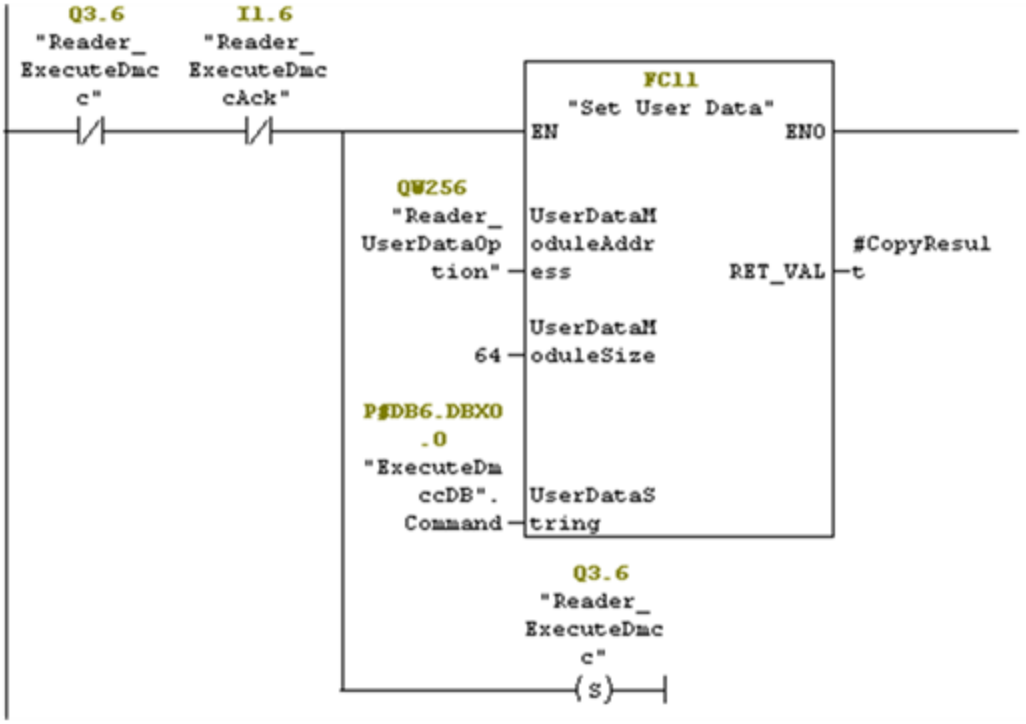
Executing DMCC Commands

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.

Industrial Protocols for the Wireless DataMan

Special considerations must be taken into account when using any of the Industrial Ethernet protocols with DataMan wireless readers. Industrial protocol connections are established not between the PLC and the wireless device, but between the PLC and the DataMan base station.

The DataMan wireless reader may be, at times:

- powered down
- in hibernation mode
- physically out of range

Automatic power down and hibernation are features that preserve power and extend the time period between recharging.

The following sections detail the necessary configurations you have to set to communicate with the DataMan base station (through which information is routed to the DataMan wireless reader).

Protocol Operation

Because the wireless reader can become unavailable at times, the Industrial Ethernet protocols are actually run from the base station. The base station is directly wired to the Ethernet network and is online continuously (even when the reader is unavailable). In this way the Ethernet protocol operation is not disrupted when the reader is unavailable. This results in a few special considerations.

Ethernet Address

Because the Ethernet protocols are run from the base station, you must use the base station Ethernet address when configuring the protocol. For example, when configuring EtherNet/IP using the RSLogix5000 software package, you select the DM8000 reader but you must enter the IP address of the base station (not the reader). The PLC using the protocol must communicate with the base station.

PLC Triggering

Because the wireless reader can become unavailable at times, triggering from the PLC is not supported on wireless readers. The PLC will receive all read results triggered from the reader. Also, Soft Events and DMCC commands are supported (when the reader is available).

Soft Events

Soft Events triggered from the PLC are supported. However, these events will fail if the reader is unavailable. Such a failure will be signaled by the ‘General Fault’ bit.

Note: Some operations assigned to soft events are in general not supported on any handheld DataMan readers (for example, ‘Train Code’, ‘Train Focus’, ‘Train Brightness’). Unsupported operations will also result in the ‘General Fault’ signal being set.

DMCC

DMCC commands issued from the PLC are supported. However, these events will fail if the reader is unavailable. Depending on which Industrial Ethernet protocol is in use, the failure will be signaled either with the ‘General Fault’

signal or an error status return code. The new DMCC error return code ‘105’ has been established to signal that the reader is unavailable.

Note: When issuing DMCC commands from the PLC to a wireless reader, do not change the DMCC response format (that is, do not issue the command ‘COM.DMCC-RESPONSE’). Doing this may disable DMCC operation on the wireless reader. This command can be used on non-Industrial Ethernet connections such as Telnet.

Offline Buffering

When the wireless reader is physically out of range of the base station, it will automatically buffer all reads. Buffer capacity varies according to a number of factors. However, typically several hundred reads can be stored. These buffered reads are also preserved if the reader enters hibernation (sleep mode). When the reader is brought back into range of the base station the buffered reads will automatically be downloaded.

If this offline buffering situation is anticipated in the user application, then Industrial Ethernet protocol buffering should also be enabled. Protocol buffering will allow a PLC to control the rate that read results are returned to the PLC. This prevents the PLC from becoming overwhelmed by a rapid flow of read results. Refer to the specific Industrial Protocol section of this document for details of enabling and utilizing protocol buffering.

Note: The base station can buffer up to 500 results.

Status of Industrial Protocols

The Status field of the industrial protocols (in Setup Tool, under Industrial Protocols tab) displays the last logged message of the status dialog. The message will remain displayed until another message is logged. If no industrial protocol is enabled, the status field remains blank.

Status messages vary by protocol, and some protocols do not log any messages. See the table below for a summary of messages for protocols that display them:

Protocol/Reader	Message	Description
Wifi readers	Industrial ethernet protocols shutdown	Displayed if no protocol is running
	Proxy %s connection opening	Opening connection to wifi reader, s == connection type ("Command", "Result", or "User DMCC")
	Proxy %s open	Connection open, s == connection type ("Command", "Result", or "User DMCC")
	Proxy %s not open	failed to open connection, s == connection type ("Command", "Result", or "User DMCC")
	Proxy %s closed	connection has been closed, s == connection type ("Command", "Result", or "User DMCC")
Modbus TCP	Modbus TCP starting...	Protocol starting
	Modbus TCP: Server Setup Failed	Failed to start server
	Modbus TCP running	Protocol running
	Modbus TCP: Add block %s failed	Data block configuration error
	Modbus TCP control block %s	Status of ModbusTCP data block, s == “info” or “error”

Protocol/Reader	Message	Description
SLMP Protocol	Opening SLMP scanner connection %s:%x	starting SLMP connection, s==IP address, x==port number
	SLMP scanner connection established %s:%x	SLMP connection established, s==IP address, x==port number
	SLMP scanner connection %s:%x rejected	PLC has rejected the connection, s==IP address, x==port number
	SLMP scanner connection %s:%x faulted	Miscellaneous connection fault has occurred, s==IP address, x==port number
	SLMP scanner poll rate set too low	Configured SLMP poll rate is set too low
	SLMP scanner connection normal %s:%x	SLMP connection has been restored, s==IP address, x==port number
	SLMP Scanner control block %s	Status of SLMP data block, s == “info” or “error”
	SLMP Scanner failed transfering '%s' block	Comm error, failed to transfer a data block, s==block name

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. Alternatively, you can open the Command Reference through the Setup Tool Help menu.

DMCC Overview

DataMan Control Commands (DMCC) are a method of configuring and controlling a DataMan reader from a COM port, either directly or programatically through a custom application. Depending on the DataMan reader you are using, the COM port connection be either RS232, USB, or the Telnet protocol in the case of Ethernet capable readers. By default, Ethernet capable readers are configured to communicate over TCP port number 23, but you can use the DataMan Setup Tool to assign a different port number as necessary.

Cognex recommends using HyperTerminal, a built-in Windows communications application, to test your ability to connect to an available reader.

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 *command* and *data*.

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 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

101: command invalid

102: parameter invalid

103: checksum incorrect

104: parameter rejected/altered due to reader state

105: reader unavailable (offline)

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	*no response*	||[0]\r\n	Enable the DataMatrix symbology.
||>TRIGGER ON\r\n	*decoded data or no-read response*	||[0]\r\n ||[1]*decoded data or no-read response* in base64\r\n	Trigger Command

DMCC Application Development

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

You can also use the DataMan SDK (hereafter referred to as SDK). The following sections give detailed information about installing the SDK, its contents, building the SDK sample application, and about the utility source codes provided with the SDK.

Note: If you want to create your own application from scratch and 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.

DataMan SDK Contents

The DataMan SDK comprises the SDK binary files and their documentation, along with code sources of some helper utilities and a sample application.

The binary files are available for two platforms: one for Microsoft .Net (PC) and one for Microsoft .Net Compact Framework (CF). The name of each file corresponds to the platform it belongs to (PC/CF). There are two components for each platform, one is the DataMan SDK core itself (Cognex.DataMan.SDK), the other is for discovering available devices to be used with the SDK (Cognex.DataMan.Discovery).

The source codes are provided in the form of complete Microsoft Visual Studio projects. In order to build the SDK sample application, open the sample code's solution in Microsoft Visual Studio and choose *Build solution*.

Using the SDK

Usual steps in a typical DataMan SDK application

1. Discover the device (may be omitted if the device address is known in advance).
2. Subscribe to the events you are interested in (e.g. result string arrived event).
3. Connect to the device.
4. Send DMCC commands to the device (e.g. trigger).
5. Process the incoming result data (e.g. show result string).

Accessing the DataMan SDK library

To use the SDK for your own purposes, perform the following steps:

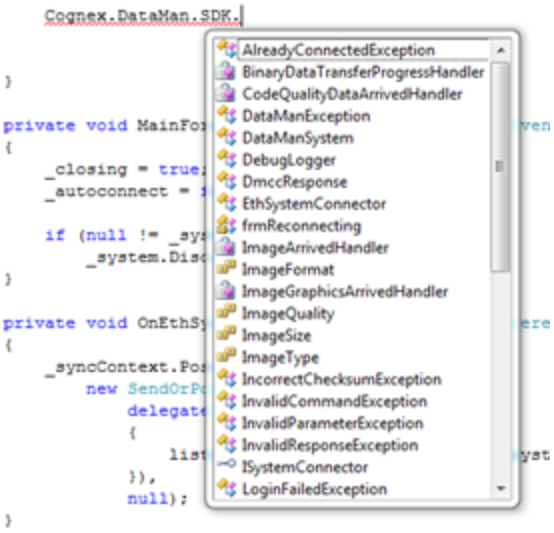
1. In Microsoft Visual Studio, click **Create Solution/Project**.
2. Under **Project**, right-click **References** and choose **Add Reference…**
3. In the pop-up window, click the Browse tab and look for the *Cognex.DataMan.SDK.*.dll* file (where * refers to the

platform you are working on, either PC or CF) in the directory where you installed or copied the binary files.

4. You can add the following line to the beginning of your code:

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

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



Enumerating DataMan Devices

In your project, which already uses the SDK, you'll need the following additional steps:

1. Under **Project**, right-click **References** and choose **Add Reference…**
2. In the pop-up window, click the Browse tab and look for the *Cognex.DataMan.Discovery.*.dll* file (where * refers to the platform you are working on, either PC or CF) in the directory where you installed or copied the binary files.
3. Add the following line to the beginning of your code:

```
using Cognex.DataMan.Discovery;
```

to find the different elements belonging to the SDK in these namespaces. They will appear in the intellisense.

From this point on, you can choose to discover devices either via Ethernet or via serial communication (RS232/USB), or you can choose to do both.

4. Discovering devices via Ethernet:

a. Create a new EthSystemDiscoverer.

```
EthSystemDiscoverer ethSystemDiscoverer = new
EthSystemDiscoverer();
```

b. Subscribe to its SystemDiscovered event.

```
ethSystemDiscoverer.SystemDiscovered += new
EthSystemDiscoverer.SystemDiscoveredHandler(OnEthSystemDiscovered);
```

c. Create event handler of type EthSystemDiscoverer.SystemDiscoveredHandler.

d. The event handler argument is an EthSystemDiscoverer.SystemInfo. These SystemInfo objects contain information required for connecting to a reader. You can store these SystemInfo objects in your own collection.

e. To start device discovery, call the ethSystemDiscoverer.Discover() method.

5. Discovering devices via serial communication (RS232/USB):

a. Create a new SerSystemDiscoverer.

```
SerSystemDiscoverer serSystemDiscoverer = new
SerSystemDiscoverer();
```

b. Subscribe to its SystemDiscovered event.

```
serSystemDiscoverer.SystemDiscovered += new
SerSystemDiscoverer.SystemDiscoveredHandler(OnSerSystemDiscovered);
```

c. Create event handler of type SerSystemDiscoverer.SystemDiscoveredHandler.

d. The event handler argument is a SerSystemDiscoverer.SystemInfo. These SystemInfo objects contain information required for connecting to a reader. You can store these SystemInfo objects in your own collection.

e. To start device discovery, call the serSystemDiscoverer.Discover() method.

Note: The SystemDiscovered event will be fired every time a device is detected (either the device announced itself after booting up or it responded to the Discover() method).

Subscribing to Events

If you want to react to result-like events in your application, you have to subscribe to the related events. There are also some events related to connection state changes.

Here is an example where you subscribe for the events of read string and image arrival:

```
mySystem.XmlResultArrived += new XmlResultArrivedHandler(OnXmlResultArrived);
mySystem.ImageArrived += new ImageArrivedHandler(OnImageArrived);
```

Note: The order of the result components may not always be the same, so if it is important to synchronize them, use the ResultCollector utility class provided via the DataManUtils component. (See details in section Helper Utilities).

Connecting to a DataMan Device

Your Ethernet device

Connect to your Ethernet device by performing the following steps:

1. Create a connector to your device:

```
EthSystemConnector myConn = new EthSystemConnector(deviceIP);
```

where *deviceIp* is either a known IP address or one that was discovered by an EthSystemDiscoverer.

2. Specify user name and password:

```
myConn.UserName = "admin";
myConn.Password = "password or empty string";
```

3. Create a new DataManSystem instance with the created connector:

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

4. Call the Connect() method of your DataManSystem instance:

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

5. (Optional) Verify if you are connected:

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

6. To disconnect, call

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

Note: Currently all devices use the user name admin. If no password is required, an empty string can be used.

Your Serial device

Connect to your serial device by performing the following steps:

1. Create a connector to your device:

```
SerSystemConnector myConn = new SerSystemConnector(PortName, Baudrate);
```

where *PortName* and *Baudrate* are either known serial connection parameters or come from a SerSystemDiscoverer.

2. Create a new DataManSystem instance with the created connector:

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

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

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

4. (Optional) Verify if you are connected:

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

5. To disconnect, call

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

Sending DMCC Commands to DataMan Devices

Use SendCommand () for sending different commands to the reader. For information about available commands, refer to the ***DMCC Command Reference***.

There is one mandatory parameter for SendCommand () which is the command string itself. There are also two optional parameters: one for overriding the default timeout for the command and another for passing additional bytes with the command.

The following is an example for sending a DMCC command.

```
DmccResponse response = mySystem.SendCommand("GET DEVICE.TYPE");
```

Note: The response's content resides in the response object's PayLoad property. Also note that no DMCC header or footer is specified in the command string.

Some functions like SendCommand() or GetLiveImage() also have asynchronous implementations. If you wish to use these, look for the desired function name with *Begin/End* prefix. These functions go in pairs; the function with the Begin prefix returns an IAsyncResult which can be used by the one with the End prefix.

Displaying Static and Live Images from a DataMan Device

To have static images displayed, use DataManSystem.GetLastReadImage () or subscribe for the event ImageArrived to get images.

To have live images displayed, perform the following steps:

1. Set the reader to live display mode:

```
mySystem.SendCommand("SET LIVEIMG.MODE 2");
```

2. Periodically poll the device for images by using

```
mySystem.GetLiveImage(ImageFormat, ImageSize, ImageQuality);
```

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.

To turn off live display mode, use

```
mySystem.SendCommand("SET LIVEIMG.MODE 0")
```

Helper Utilities

Some helper functions are provided as source codes with the SDK in the project called DataManUtils. Some of the main features are described below.

Gui

Provides functions for image manipulation like fitting a result image into a specified control, converting bitmap data to/from a byte array, etc.

Additional classes provide SVG helpers for image parsing and SVG rendering. SVG formatted result component is used by the reader to mark the area of the image where the code was detected.

ResultCollector

The order of result components may not always be the same. For example sometimes the XML result arrives first, sometimes the image. This issue can be overcome by using the ResultCollector.

The user needs to specify what makes a result complete (e.g. it consists of an image, an SVG graphic and an xml read result) and subscribe to ResultCollector's ComplexResultArrived event.

The ResultCollector waits for the result components. If a result is complete, a ComplexResultArrived event is fired. If a result is not complete but it times out (time out value can be set via the *ResultTimeOut* property) or the ResultCollector's buffer is full (buffer length can be set via the *ResultCacheLength* property), then a PartialResultDropped event is fired. Both events provide the available result components in their event argument, which can be used to process the complex result (e.g. maintain result history, show the image, graphic and result string, etc.)

DmccEscaper

Can be used to escape or un-escape a DMCC command string.

FileLogger

Simple logger class can be used during development. This, like all other utilities provided here, works both on PC and CF platforms.

Using the Helper Utilities

The helper utilities are contained in two projects. Both projects refer to the same source codes, but one is created for Microsoft .Net Compact Framework (DataManUtilsCF) and the other of for the PC's Microsoft .Net Framework (DataManUtilsPC). To use the features provided in these utilities, include the proper DataManUtils project in your solution and reference it in the project in which you wish to use it.

Scripting

Script-Based Data Formatting

The DataMan Setup Tool allows you to have different data formatting combinations, and to have the reader perform different actions on the output channel, for example, beep, or have the LEDs blink, or pull output 1 up.

The script-based formatting has two main advantages:

- flexible result configuration
- configuring reader events before the result returns

Note: Script-based formatting limits the user to performing two custom events and overwriting the system event.

Global JavaScript Functions

The DMCC functions fall to three categories:

- Commands, e.g. to issue a beep or a re-boot
- Setter functions for properties
- Getter functions for properties

The functions make use of the variable arguments feature of the script engine. The types of the function arguments are compared to the expected types defined by the DMCC commands. If the number of arguments or an argument type is incorrect an error status is returned.

The functions return an object with a property for the status. If a command returns a response it can be accessed by the response property. The status codes are the same as for the DMCC commands.

If a function executes successfully, a zero status value is returned. Script exceptions are not used.

To simplify the integration of DMCC commands in scripting, it is now possible to evaluate a DMCC command line as full command string. It is not required to split the DMCC command into the type correct command arguments.

Note:

- The data formatting script function is executed after the output delay time or distance elapsed.
- All scripting functions run in a separate thread and the execution is mutual exclusive. It is not possible that a script function is interrupted by another.
- Use [0,1] or [true,false] instead of [ON|OFF] when scripting.

DMCC	Description
dmccGet	Based on the DMCC implementation the response is always returned as a single string even for multi-value responses.
dmccSet	It supports multiple and type correct parameters.
dmccCommand	N/A
dmccSend	The functions evaluates a DMCC command. The return value contains the dmcc response type containing status and response string. The function requires one string argument.

Example

```
var foo = dmccGet("DECODER.ROI");
```

The set command supports multiple and type correct parameters, for example:

```
dmccSet("DECODER.ROI", 16, 1280, 16, 1024);
```

Example

The following example uses the dmccSet functions to issue a beep signal, set the ftp server IP for image storage and adds the MAC to the output response:

```
function onResult (decodeResults, readerProperties, output)
{
      var myoutput;
      var result_tmp = dmccCommand(”BEEP”, 1, 1);


      result_tmp = dmccSet(”FTP-IMAGE.IP-ADDRESS”, ”192.168.23.42”);
      if(result_tmp.status !=0)
{
      throw(”FATAL: failed to set the ftp server address”);
}
var mac = dmccGet(”DEVICE.MAC-ADDRESS”);

myoutput = ’Result=”’ + decodeResults[0].content + ’”, MAC=’+mac.response;
output.content = myoutput;
}
```

In case the DMCC set command for the IP address fails, a non-zero status will be returned, and a script exception will be thrown that is reported by the DataMan Setup Tool.

Note: If you use the Throw() command, like in the example above, to report the occurrence of an anomalous situation (exception), the error will appear in the Setup Tool’s error log. To access the error log, in the Setup Tool’s menu bar, click System and then click Show Device Log.

Example

To get the device name using the dmccGet function the correct string argument is required:

```
var res = dmccGet(”DEVICE.NAME”);
```

The dmccSend function can be used in a similar way, but without splitting the command and type correct arguments:

```
var res = dmccSend(”GET DEVICE.NAME”);
```

The return value is the same.

DMCC Support

The following DMCC commands are available for Script-Based Formatting:

Command	Range	Description
GET/SET FORMAT.MODE	[0..1]	Select formatting mode: • 0 = basic formatting • 1 = script-based formatting
SCRIPT.LOAD	length	Load the formatting script from the host to the reader.
SCRIPT.SEND	-	Send the formatting script from the reader to the host.

Auxiliary Functions

The following auxiliary global functions are also available:

- function for decoding escape sequences
- function to encode a string argument into base64 encoding

Function decode_sequences

This global function is used to decode escape sequences. The function returns the string that contains the decoded escape sequence. The return value can be used to add keyboard control commands to a result transmitted over a HID connection.

Parameter	Type	Description
`encodedString`	string	A string value that contains keyboard escape sequences.

To simulate Alt-key, Ctrl-key, or Shift-key combinations, the following four escape sequences are available:

- **\ALT-** for <ALT-key> sequences
- **\CTRL-** for <CTRL-key> sequences
- **\SHIFT-** for <SHIFT-key> sequences
- **\K** for special keys

Note: The key after the backslash needs to be a capital letter, otherwise no special key combination is recognized.

Supported Key Sequences and Keys

The following list contains the currently supported keys/key combinations:

- ALT-A to ALT-Z
- CTRL-A to CTRL-Z
- CTRL-F1 to CTRL-F12
- SHIFT-F1 to SHIFT-F12
- F1 to F12
- ALT-F1 to ALT-F12
- PageUp, PageDown, Home, End, Arrow (up, down, left, right), , Insert, Delete, Backspace, Tab, Esc, Print Screen, GUI (left, right) keys.

- The escape sequences for these are the following:
  - PageUp -> \KPup;
  - PageDown -> \KPdn;
  - Home -> \KHome;
  - End -> \KEnd;
  - Up Arrow -> \KUar;
  - Down Arrow -> \KDar;
  - Left Arrow -> \KLar;
  - Right Arrow -> \KRar;
  - Insert -> \KIns;
  - Delete -> \KDel;
  - Backspace -> \KBksp;
  - Tab -> \KTab;
  - Esc -> \KEsc;
  - Print Screen -> \KPrtScr;
  - Left GUI -> \KLGui;
  - Right GUI -> \KRGui;

Example

To pre- or post-pend a Ctrl-B keyboard control command, the following code example can be used:

```
var ctrl_b = decode_sequences("\\Ctrl-B;");

function onResult (decodeResults, readerProperties, output)
{
      if (decodeResults[0].decoded)
      {
            output.content = ctrl_b+decodeResults[0].content+ctrl_b;
      }
}
```

Note: The backslash for initiating the escape sequence must also be escaped in the input string. The terminating semicolon is necessary to be able to distinguish between sequences with the same prefix, otherwise key sequences could be interpreted arbitrarily, e.g. there would be no means to detect if \KF11 means "press F11" or "Press F1 followed by a one". If a wrong or incomplete sequence is used, the two characters which mark the escape sequence are ignored. In this case, the first two letters of the escape sequence are skipped and the remaining characters will be sent. For example, the sequence "\ALT-M11;" is invalid and will result in displaying "LT-M11;".

Function encode_base64

This global function is used to encode a string argument into base64 encoding. The encoded result is returned as a string object.

Parameter	Type	Description
`inputString`	string	Input string to encode into base64.

Error Management

Scripting errors may occur when the script is loaded or the code parser function is called. These errors are shown in the following locations:

- device log
- error box in Script-Based Formatting window

Formatting Script

When script-based formatting is enabled, a user-defined JavaScript module is responsible for data formatting. The parsing function, which defaults to onResult, is called with three objects as arguments holding the array of DecodeResult objects, ReaderProperties objects such as trigger mode or statistics, and the output object. There is only one entry point for both single and multicode results.

Class hierarchy is structured in the following way:

```
function onResult [decodeResults, readerProperties, output]
```

```
DecodeResult
SymbologyProperties
  Point
ValidationResult
  GS1Validation
  DoDValidation
QualityMetrics
  Metric
ReaderProperties
  Trigger
  Statistics
```

```
Output
  Event
```

See the detailed description of the related objects below.

Function onResult

This is the event handler for decode events, with zero, one or more decoded results.

Property	Type	Description
`decodeResults`	DecodeResult[]	Input, an array of DecodeResult objects. One decode result will hold all information related to that decode attempt.
`readerProperties`	ReaderProperties	Input, the reader properties not tied to the actual decode result.
`output`	Output	Output, the object which needs to be updated to modify the output string or raise events.

Function onGenerateFTPFilename

The name of the file to be sent to the FTP server can be generated with this function.

Property	Type	Description
`decodeResults`	DecodeResult[]	Input, an array of DecodeResult objects. One decode result will hold all information related to that decode attempt.
`readerProperties`	ReaderProperties	Input, the reader properties not tied to the actual decode result.
`output`	Output	Output, the object which needs to be updated to modify the output string or raise events.

The file name of the image to be uploaded is taken from the string return value of the script. For example:

```
function onGenerateFTPFilename(decodeResults, readerProperties, output)
{

      var ftp_filename = readerPRoperties.name + "-";
      ftp_filename +=readerProperties.trigger.index + "-" + decodeResults
      [0].image.index;
      return ftp_filename;


}
function onGenerateFTPPCMReportFilename(decodeResults, readerProperties, output)
{

      var ftp_filename = readerPRoperties.name + "-";
      ftp_filename +=readerProperties.trigger.index + "-" + decodeResults
      [0].image.index;
      return ftp_filename;


}
```

DecodeResult Object

The following tables list the details of the DecodeResult object, its types and properties.

Decode Result

Describes the details of one decoded result.

Property	Type	Description
decoded	boolean	True if decoding was successful.
content	string	The (raw) read result.
decodeTime	integer	The decoding time in milliseconds.
triggerTime	integer	The trigger time in milliseconds.
timeout	string	The trigger timeout in milliseconds.
symbology	SymbologyProperties	The values of this property are listed in the Symbology Properties table below.
image	ImageProperties	The values of this property, also known as capture attributes, are listed in the Image Properties table below.
validation	ValidationResult	The values of this property are listed in the Validation Result table below.
metrics	QualityMetrics	The values of this property are listed in the Quality Metrics table below.
readSetup	integer	Used read setup index token.
source	string	The name of the device that decoded the image.
annotation	string	Result annotation for Multi-Reader Sync triggering.
label	string	Symbol label.

Symbology Properties

Symbology properties for a decoded result.

Property	Type	Description
`name`	string	The name of the symbology.
`id`	string	The symbology identifier (by ISO15424).
`quality`	integer	The overall quality metrics for the code, ranging in [0, 100]. For symbologies that use Reed-Solomon error correction (e.g. DataMatrix, QR Code, AztecCode, MaxiCode, DotCode, PDF417, certain 4-state postal codes), it reports the UEC (unused error correction). For linear symbologies, it indicates the overall quality of the code. The higher the number, the better the quality.
`moduleSize`	float	The module size. (The unit is pixel per module, ppm.)
`size`	point	The size of the symbol in columns x rows. If not applicable for the symbol, the values will be set to -1.
`corners`	array of Point	This specifies the coordinates of the four corners. The details of the Point property type are listed in the Point table below. The corner coordinates are returned in the following order: For non-mirrored symbols, • corner 0: upper left corner of the symbol, • corner 1: upper right corner of the symbol, • corner 2: lower right corner of the symbol, • corner 3: lower left corner of the symbol, except for non-mirrored DataMatrix and Vericode, where corner 0 is where the two solid lines of cells along two sides meet, and corners 1-3 follow counter clockwise. For mirrored symbols, the corners are mirrored correspondingly.
`center`	Point	This specifies the coordinates of the center. The details of the Point property type are listed in the Point table below.
`angle`	float	The code orientation in degrees.

PtpTimeStamp

Peer to peer timestamp value on image acquisition.

Property	Type	Description
s	integer	Image acquisition timestamp sec property.
ns	integer	Image acquisition timestamp nanosec property.

Point

Point is the ordered pair of integer x- and y-coordinates that defines a point in a two-dimensional plane.

Property	Type	Description
x	integer	This value specifies the x coordinate.
y	integer	This value specifies the y coordinate.

ImageProperties Object

The following tables list the details of the ImageProperties object, its types and properties.

Image Properties

Properties of a captured image.

Property	Type	Description
index	integer	The index of the image within the trigger.
FoV	Rect	The Field of View, the area of image sensor used relative to the top left sensor corner. The details of the Rect property type are listed in the Rect table below.
RoI	Rect	The Region of Interest, the part of the FoV that is actually used, relative to the sensor. The details of the Rect property type are listed in the Rect table below.
exposureTime	integer	The exposure time in microseconds.
gain	integer	The camera gain.
autoExposure	boolean	True if automatic exposure is used.
illEnabled	boolean	True if internal illumination is enabled.
illIntensity	integer	The internal illumination intensity.
extillEnabled	boolean	True if external illumination is enabled.
extillIntensity	integer	The external illumination intensity.
targetBrightness	integer	The target brightness in case of automatic exposure.
focusLength	integer	The focus value in millimeters. It is 0 if NA.
setupIndex	integer	The current index of read setup.
inputStates	array of boolean	The state of the input lines when the trigger was started.
filterTime	integer	The duration of filtering in milliseconds.
creationTime	integer	Creation time.
creationTicks	integer	Encoder ticks corresponding to image creation time.
ptpTimeStamp	ptpTimeStamp	PtP image acquisition timestamp.
id	integer	The numerical identifier of this image.

Rect

Rect describes the width, height, and location of a rectangle.

Property	Type	Description
top	integer	This specifies the top value relative to the top left sensor corner.
bottom	integer	This specifies the bottom value relative to the top left sensor corner.
left	integer	This specifies the left value relative to the top left sensor corner.
right	integer	This specifies the right value relative to the top left sensor corner.

ValidationResult Object

The following tables list the details of the ValidationResult object, its types and properties.

Validation Result

Describes all details of the validation.

Property	Type	Description
state	integer	These are the validation states: • notTried • fail • pass The format of this property is “validation.state.notTried”.
method	integer	These are the validation methods: • none • gs1 • iso • dod_uid • pattern • matchString The format of this property is “validation.method.none”.
matchString	string	This property returns with the previously configured match string. Match string validation should be enabled for this.
failurePos	integer	The position of validation failure.
failureCode	integer	The validation failure code.
failureMsg	string	The error message describing the cause of validation failure.
gs1	GS1 Validation	The details of the GS1 Validation property type are listed in the GS1 Validation table below.
dod_uid	DoD Validation	The details of the DoD Validation property type are listed in the DoD Validation table below.

GS1Validation

GS1 validation details.

Property	Type	Description
AI00	string	Identification of a logistic unit (Serial Shipping Container Code)
AI01 AI01 AI01 AI01	string string string string	Identification of a fixed measure trade item (Global Trade Item Number) Identification of a variable measure trade item (GTIN) Identification of a variable measure trade item (GTIN) scanned at POS Identification of a variable measure trade item (GTIN) not scanned at POS
AI02 AI02	string string	Identification of fixed measure trade items contained in a logistic unit Identification of variable measure trade items contained in a logistic unit
AI10	string	Batch or lot number
AI11	string	Production date
AI12	string	Due date for amount on payment slip
AI13	string	Packaging date
AI15	string	Best before date
AI16	string	Sell by date
AI17	string	Expiration date
AI20	string	Product variant

AI21	string	Serial number
AI240	string	Additional product identification assigned by the manufacturer
AI241	string	Customer part number
AI242	string	Made-to-Order variation number
AI243	string	Packaging component number
AI250	string	Secondary serial number
AI251	string	Reference to source entity
AI253	string	Global Document Type Identifier
AI254	string	GLN extension component
AI255	string	Global Coupon Number (GCN)
AI30	string	Variable count
AI31nn AI32nn AI35nn AI36nn	string	Trade measures
AI33nn AI34nn AI35nn AI36nn	string	Logistic measures
AI337n	string	Kilograms per square metre
AI37	string	Count of trade items contained in a logistic unit
AI390n	string	Amount payable or coupon value - Single monetary area
AI391n	string	Amount payable and ISO currency code
AI392n	string	Amount payable for a variable measure trade item – Single monetary area
AI393n	string	Amount payable for a variable measure trade item and ISO currency code
AI394n	string	Percentage discount of a coupon
AI400	string	Customer’s purchase order number
AI401	string	Global Identification Number for Consignment (GINC)
AI402	string	Global Shipment Identification Number (GSIN)
AI403	string	Routing code
AI410	string	Ship to - Deliver to Global Location Number
AI411	string	Bill to - Invoice to Global Location Number
AI412	string	Purchased from Global Location Number
AI413	string	Ship for - Deliver for - Forward to Global Location Number
AI414	string	Identification of a physical location - Global Location Number
AI415	string	Global Location Number of the invoicing party
AI420	string	Ship to - Deliver to postal code within a single postal authority
AI421	string	Ship to - Deliver to postal code with three-digit ISO country code
AI422	string	Country of origin of a trade item
AI423	string	Country of initial processing
AI424	string	Country of processing
AI425	string	Country of disassembly
AI426	string	Country covering full process chain
AI427	string	Country subdivision of origin code for a trade item

AI7001	string	NATO Stock Number (NSN)
AI7002	string	UN/ECE meat carcasses and cuts classification
AI7003	string	Expiration date and time
AI7004	string	Active potency
AI7005	string	Catch area
AI7006	string	First freeze date
AI7007	string	Harvest date
AI7008	string	Species for fishery purposes
AI7009	string	Fishing gear type
AI7010	string	Production method
AI703s	string	Number of processor with three-digit ISO country code
AI710 AI711 AI712 AI713	string	National Healthcare Reimbursement Number (NHRN):
AI8001	string	Roll products - width, length, core diameter, direction, splices
AI8002	string	Cellular mobile telephone identifier
AI8003	string	Global Returnable Asset Identifier (GRAI)
AI8004	string	Global Individual Asset Identifier (GIAI)
AI8005	string	Price per unit of measure
AI8006	string	Identification of the components of a trade item
AI8007	string	International Bank Account Number (IBAN)
AI8008	string	Date and time of production
AI8010	string	Component / Part Identifier (CPID)
AI8011	string	Component / Part Identifier serial number
AI8012	string	Software version
AI8017 AI8018	string	Global Service Relation Number (GSRN)
AI8019	string	Service Relation Instance Number (SRIN)
AI8020	string	Payment slip reference number
AI8110	string	Coupon code identification for use in North America
AI8111	string	Loyalty points of a coupon
AI8200	string	Extended packaging URL
AI90	string	Information mutually agreed between trading partners
AI91-99	string	Company internal information

DoD Validation

DoD validation details.

Property	Type	Description
enterpriseID	string	The enterprise identifier.
serialNum	string	The serial number.
partNum	string	The part number.

uniqueItemID	string	The unique item identifier.
batchNum	string	The batch number.

QualityMetrics Object

The following tables list the details of the QualityMetrics object, its types and properties. The details of the Metric property type are listed in the Metric table below. All the metrics listed are available for all the standards available under the Symbology Settings pane in the DataMan Setup Tool.

Quality Metrics

Describes the quality of all measured parameters.

Property	Type	1D Standards	2D Standards	Description
singleScanInt	Metric	1D Readability		The single-scan integrity, raw member is set to -1. Single-scan integrity is a general measure of the ease of decoding a barcode using only a single scan across it. This is meant to represent the way that simple decoders work. In general, such algorithms are not advanced and the decodability is lower if a symbol has damage in multiple locations in the barcode. A low singleScanInt metric may indicate many different problems, as it is a general measure of code quality.

Property	Type	1D Standards	2D Standards	Description
symbolContrast	Metric	1D Readability, ISO/IEC 15416	ISO/IEC 15415 (DataMatrix, QR, DotCode), SEMI T10	The contrast of the symbol in ISO15415. Symbol contrast is a measure of the difference in grayscale value between the light and dark cells. A high contrast makes the code easier to decode, while a code with low contrast may not decode well due to difficulty separating the light and dark cells from each other. A poor contrast might indicate poor lighting, a code which is difficult to read due to similarities between the print and the background, or that a printer is performing poorly.
cellContrast	Metric		AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR)	The contrast of the cell. Cell contrast is a measure of the difference in grayscale value between the light and dark parts of the cell. A high contrast makes the code easier to decode, while a code with low contrast may not decode well due to difficulty separating the light and dark areas of the cells. A poor contrast might indicate poor lighting, a code which is difficult to read due to similarities between marked and unmarked areas.

Property	Type	1D Standards	2D Standards	Description
axialNonUniformity	Metric		ISO/IEC 15415 (DataMatrix, QR, DotCode), AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR)	The axial non-uniformity. Axial non-uniformity is a measure of the difference in spacing of grid cells along each axis. In the best case, this value will be zero, indicating that centers of the grid cells are evenly spaced in all directions. A poor axial non-uniformity might indicate problems in the printing process for the code, which causes the code to appear stretched out or compressed.

Property	Type	1D Standards	2D Standards	Description
printGrowth	Metric	1D Readability	ISO/IEC 15415 (DataMatrix, QR, DotCode), AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR)	The print growth. Print growth is a measure of how completely a light or dark patch fills the cell allocated to it. High print growth means that a cell exceeds the boundaries allocated to it, while a low print growth indicates that the cells are not taking up all the available space. Either of these may cause problems (either by making adjacent cells difficult to read in the case of high growth, or making the cell itself difficult to read in the case of low growth). As a result, a print growth close to zero is desirable. A high or low print growth usually indicates problems with the printing process for a code. For instance, a dot peen marker may be wearing out and making smaller marks, or a printer may be depositing too much ink on a label and making the marks too large.
UEC	Metric		ISO/IEC 15415 (DataMatrix, QR, DotCode), AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR), SEMI T10	The unused error correction. Unused Error Correction measures the amount of Error Checking and Correction data that was printed into the code, but was unused. A high UEC count is good, as it means that little to no Error Correction data was needed to successfully read your code. A low UEC value may be due to poor printing, poor imaging, an incorrect code, or a damaged code.

Property	Type	1D Standards	2D Standards	Description
modulation	Metric	ISO/IEC 15416	ISO/IEC 15415 (DataMatrix, QR), AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR)	The modulation. Modulation measures how easily separable light cells are from dark cells in a code. Somewhat similar to contrast, higher modulation is better, and low modulation can lead to difficulty telling light cells from dark ones. Low modulation can indicate poor lighting, a code which is difficult to read due to similarities between the print and the background, or that a printer is performing poorly.
fixedPatternDamage	Metric		ISO/IEC 15415 (DataMatrix, QR), AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR)	The fixed pattern damage. Fixed pattern damage is a measure of how much of the fixed patterns around the outside of the code (the solid finder patterns and the alternating clocking patterns) are intact. If the fixed patterns are damaged, then the code may be difficult to find at all, let alone decode. A poor fixed pattern damage score usually indicates a code which has been damaged or smudged, or it indicates a quiet zone violation.

Property	Type	1D Standards	2D Standards	Description
gridNonUniformity	Metric		ISO/IEC 15415 (DataMatrix, QR, DotCode), AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR)	The grid non-uniformity. Grid non-uniformity measures the difference between the optimal placement of cells based on the overall grid and their actual placements. This is similar to the axial non-uniformity measurement, but instead of measuring a stretching or compressing of the whole grid, this measures how much the individual cells deviate from their expected positions. Poor grid non-uniformity usually indicates a printing process which is not consistent in its placement of the cells.
extremeReflectance	Metric		ISO/IEC 15415 (DataMatrix, QR)	The extreme reflectance. This metric measures the brightness of the background on which the code is printed. A too high value might indicate lighting or imaging trouble that could lead to a code being washed out and difficult to read. A low value may indicate that not enough light is being applied to the code, and that contrast may be poor, leading to difficulty in reading. A poor extreme reflectance grade may also indicate trouble relating to the positioning of lights such as hotspots.

Property	Type	1D Standards	2D Standards	Description
reflectMin	Metric	1D Readability, ISO/IEC 15416		The reflectance minimum. This metric measures how dark the dark part of a barcode is. A low value indicates that the dark parts of the code are dark, and a high value indicates that they are not. A too low value may indicate that there is not enough light or too short exposure time is being used. A too high value might indicate a hotspot, too much light, or that a too high exposure time is being used. Print quality troubles, like a printer depositing less ink than intended, may also be indicated by the minimum reflectance grade.
edgeContrastMin	Metric	1D Readability, ISO/IEC 15416		The edge contrast minimum measures the ratio of minimum edge contrast to the maximum contrast in the symbol. The metric is designed to pick up any artifacts in the symbol, such as a damaged bar, which generate low contrast variations in the symbol. A poor grade here might indicate poor focus in the optical system, poor lighting, or poor printing.

Property	Type	1D Standards	2D Standards	Description
multiScanInt	Metric	1D Readability		The multi-scan integrity. Multi-scan integrity is a general measure of the ease of decoding a symbol by using multiple scans across the barcode. This metric is a way of measuring how advanced decoders might perform in decoding a particular barcode. A low multiScanInt metric may indicate many different problems, as it is a general measure of code quality.
signalToNoiseRatio	Metric		SEMI T10 (DataMatrix)	Signal To Noise Ratio (SNR) is a relative measure of the Symbol Contrast to the maximum deviation in light or dark grayscale levels in the symbol (ie. noise).
horizontalMarkGrowth	Metric		SEMI T10 (DataMatrix)	Horizontal Mark Growth is the tracking of the tendency to over or under mark the symbol, that is, a horizontal size comparison between the actual marked cells vs. their nominal size.
verticalMarkGrowth	Metric		SEMI T10 (DataMatrix)	Vertical Mark Growth is the tracking of the tendency to over or under mark the symbol, that is, a vertical size comparison between the actual marked cells vs. their nominal size.
dataMatrixCellWidth	Metric		SEMI T10 (DataMatrix)	Data Matrix Cell Width is the average width of each cell in the matrix (in pixels).
dataMatrixCellHeight	Metric		SEMI T10 (DataMatrix)	Data Matrix Cell Height is the average height of each cell in the matrix (in pixels).

Property	Type	1D Standards	2D Standards	Description
horizontalMarkMisplacement	Metric		SEMI T10 (DataMatrix)	Horizontal Mark Misplacement is the average horizontal misplacement of Data Matrix marks from their optimal Data Matrix Cell Center Points.
verticalMarkMisplacement	Metric		SEMI T10 (DataMatrix)	Vertical Mark Misplacement is the average vertical misplacement of Data Matrix marks from their optimal Data Matrix Cell Center Points.
cellDefects	Metric		SEMI T10 (DataMatrix)	Cell Defects is the ratio of incorrect pixels to total pixels in the grid.
finderPatternDefects	Metric		SEMI T10 (DataMatrix)	Finder Pattern Defects is the ratio of incorrect pixels to total pixels in the finder pattern.
overallGrade	Metric	ISO/IEC 15416	ISO/IEC 15415 (DataMatrix, QR), AIM/DPM ISO/IEC TR-29158 (DataMatrix, QR), SEMI T10	Overall grade calculated from the individual metrics.
edgeDetermination	Metric	ISO/IEC 15416		Edge Determination is the number of edges detected in the Scan Reflectance Profile. If the number of detected edges is greater than or equal to the expected number of edges, the grade is 4. Otherwise, the grade is 0.
defects	Metric	ISO/IEC 15416		Defects are irregularities in elements (bars and spaces) and quiet zones. The parameter is used to measure the ‘noise’ that results from unwanted dips and spikes in the Scan Reflectance Profile. The smaller the defect, the better the grade.

Property	Type	1D Standards	2D Standards	Description
`referenceDecode`	Metric	ISO/IEC 15416		Reference Decode is an indication of whether the standard 2D Data Matrix algorithm was able to locate and decode this particular mark. This metric generates a grade of either A or F.
`decodability`	Metric	ISO/IEC 15416		Decodability is the measure of bar code printing accuracy in relation to the symbology-specific reference decode algorithm. Decodability indicates the scale of error in the width of the most deviant element in the symbol. The smaller the deviation, the higher the grade.
`contrastUniformity`	Metric	ISO/IEC 15416	ISO/IEC 15415 (DataMatrix, QR)	Contrast Uniformity is an optional parameter that is used for measuring localized contrast variations. It does not affect the overall grade.
`reflectanceMargin`	Metric	ISO/IEC 15416	ISO/IEC 15415 (DataMatrix, QR)	Reflectance Margin measures how each module is distinguishable as light or dark compared to the global threshold. Factors (like print growth, certain optical characteristics of the substrate, uneven printing, encodation errors) can reduce or eliminate the margin for error between the reflectance of a module and the global threshold. A low Reflectance Margin can increase the probability of a module being incorrectly identified as dark or light.

Metric

Describes the quality of a measured parameter.

Property	Type	Description
raw	float	The raw metric.
grade	string	The grade of quality in a range from grade A to F, where A is the highest.

Reader Properties

The following tables list the details of the reader properties.

ReaderProperties

Reader properties not tied to the actual decode result.

Property	Type	Description
name	string	The name of the device that decoded the image.
trigger	Trigger	The details of Trigger property type are listed in the Trigger table below.
stats	Statistics	The details of the Statistics property type are listed in the Statistics table below.
inputstr	string	This property serves the same function as the <Input String> data formatting token in Standard Formatting: it holds the string that was sent to the reader via the InputString feature (only configurable through DMCC).

Trigger

Describes the details of the initiating trigger event.

Property	Type	Description
type	integer	These are the available trigger types: • single • presentation • manual • burst • self • continuous The format of this property is “trigger.type.single”.
index	integer	The unique trigger identifier.
burstLength	integer	The number of images in case of burst trigger.
interval	integer	The trigger interval in microseconds.
delayType	integer	These are the available trigger delay types: • none • time • distance The format of this property is “trigger.delayType.none”.
startDelay	integer	The trigger start delay in milliseconds (when using Trigger.delayTime.time) or millimeters (when using Trigger.delayTime.distance).

endDelay	integer	The trigger end delay in milliseconds (when using Trigger.delayTime.time) or millimeters (when using Trigger.delayTime.distance).
creationTime	integer	Creation time.
creationTicks	integer	Encoder ticks corresponding to trigger signal time.
groupIndex	integer	The unique trigger identifier property of the reader which triggered the group.
endTime	integer	Trigger event end time (in ms).
endTicks	integer	Encoder tick counter at trigger end event time.

Statistics

Operational information about the reader.

Property	Type	Description
reads	integer	The total number of decoded symbols.
noReads	integer	The number of times the trigger was received but no symbol was decoded.
triggers	integer	The total number of triggers calculated by *totalReads+totalNoReads+missedTriggers*.
bufferOverflows	integer	The number of images that were not buffered because of image buffer full condition.
triggerOverruns	integer	The number of missed triggers because acquisition system was busy.
itemCount	integer	The number of no reads when buffered no read images are allowed.
passedValidations	integer	The number of reads that passed the data validation.
failedValidations	integer	The number of reads that failed the data validation.

Output

Output describes the result and events after a decode. It is possible to specify different results for individual protocol targets. The output object has target-specific properties of type string. The name of the output property is the same as the target protocol name. If no target-specific output is assigned, the result falls back to the default result taken from the output.content property.

Property	Type	Description
content	string	The string that is sent as decode result.
events	event	These are the output events that are activated. The details of the DecodeEvents property type are listed in the DecodeEvents table below.
SetupTool*	string	The string that is sent to the Setup Tool as decode result.
Serial*	string	The string that is sent to serial and USB connections as decode result.
Telnet*	string	The string that is sent to the Telnet connection as decode result.
Keyboard*	string	The string that is sent to the HID connection as decode result. Not available for 5.2.
FTP*	string	The string that is sent to the FTP connection as decode result.

PS2*	string	The string that is sent to the PS2 connection as decode result. Not available for 5.2.
NetworkClient*	string	The string that is sent to the NetworkClient connection as decode result.
IndustrialProtocols*	string	The string that is sent to the connected PLC as decode result.

*These properties suppress the output information that was previously set via the output.content property.

An example for the protocol-specific formatting feature can be found here:

```
function onResult (decodeResults, readerProperties, output)
{
      if (decodeResults[0].decoded)
      {
            var mymsg = decodeResults[0].content;
            // output['Serial'] is identical to output.Serial
            output['Serial'] = "serial: "+mymsg;
            output.Telnet = "telnet: "+mymsg;

            output.content = mymsg;
      }
      else
      {
      output.content = "bad read";
      }
}
```

Note: For every channel that is not addressed in special, the output is the normal content text. For example:

```
function onResult (decodeResults, readerProperties, output)
{
            if (decodeResults[0].decoded)
            {
                        /* save decoded result to variable */
                        var mymsg = decodeResults[0].content;

                        /* output to telnet channel a different result
                        */
                        output.Telnet = "telnet: " + mymsg;

                        /* to all other channel output the saved
                        result */
                        output.content = mymsg;
            }
            else
            {
                        /* On bad read output to all channels the same
                        */
                        output.content = "bad read";
            }
}
```

DecodeEvents

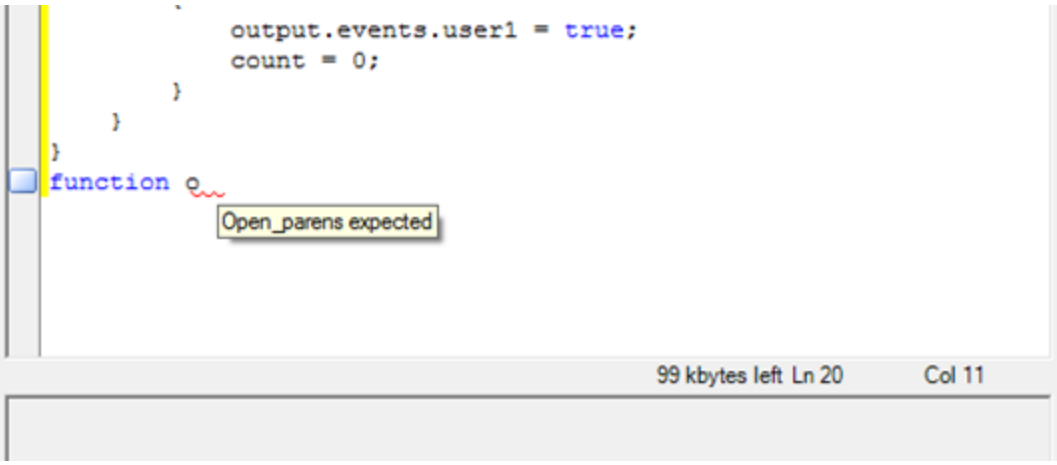
Describes the events to be emitted after a decode.

Property	Type	Description
`system`	integer	These are the system generated events: • 0 = none • 1 = good read • 2 = no read • 3 = validation failure*
`user1`	boolean	True if user event 1 is raised.
`user2`	boolean	True if user event 2 is raised.

* Only changing between good read and validation failure is supported.

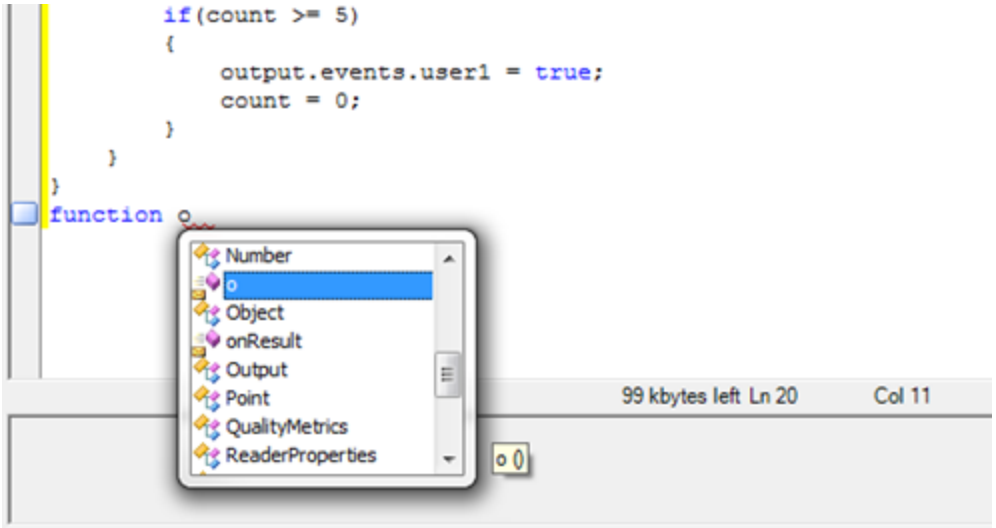
Code Completion and Snippets

The script editor features automatic code completion, which shows pop-up messages with information regarding the code that is being written. Pop-up messages usually appear when typing special characters, for example period, opening or closing brackets, etc. These messages can also be used manually with the Ctrl+Space key combination.



Code completion works in the following scenarios:

- complete a code fragment (Ctrl-Space)
- provide function list (Ctrl-Shift-Space)



The toolbar at the top of the editor collects the following actions available within the editor:

- Cut (Ctrl-x)
- Copy (Ctrl-c)
- Paste (Ctrl-v)
- Complete Word (Ctrl-k and then press w)
- Insert Snippet (Ctrl-k and then press x)

Snippets

The editor provides a selection of preset code fragments as examples. You can insert these snippets by right-clicking in the editor, using the toolbar or using the Ctrl-k and x key combination.

Custom Communication Protocol API

Custom communication scripting can be activated by a boolean VT entry that can be accessed in the DataMan Setup Tool.

The methods are encapsulated in a communication object. Each communication channel creates an instance of the communication object.

When the Custom Protocol Communication API is enabled through a private DMCC command, the scripting context adds the following capabilities and requirements:

- The constructor function of the communication object, CommHandler, contains a list of functions that the script must contain:
  - **onConnect**
  - **onDisconnect**
  - **onExpectedData**
  - **onTimer**
  - **onUnexpectedData**
  - **onError**
  - **onEncoder**

The user must implement these functions, as the custom communications function will call them.

- There are five member functions that the reader script engine offers, implemented by the reader:
  - **send**
  - **close**
  - **setTimer**
  - **expectFramed**
  - **setEncoder**

By using these functions, a user could write javascript code that allows the reader to interact with another system. In particular, the user can write code to send messages back to the other system, something that is not supported in basic scripting.

Advantage

The script engine uses the same context for function execution and object creation. This allows the sharing of data between the script-based formatting and the custom communication scripts using global accessible objects.

List of functions

The communication member functions define the following method prototypes to be implemented by the user:

CommHandler – The constructor function for the communication object. The constructor must return a new communication handler object implementing the user methods in the communication script. The reader methods are added to the communication handler object directly after construction. The implementation of the constructor is mandatory and an error will be thrown if it does not exist. Since software version 5.5 the constructor function call offers the following argument:

- ***localName***: The local name of the connection. The local name of a network connection is ”<READER_IP>:<PORT>”. An example for a Telnet connection is “10.82.80.156:23” with the default telnet port of 23. An example for a Network Client connection is “10.82.80.156:57350”. The local name for the serial connection is

"COM1" or "COM USB".

- ***onConnect*** – Initialize state upon connection (a network connection established or protocol stack starts on serial connection). If the method is not implemented an error occurs. The method has one argument and a return value:
  - *peerName* – The peer name of the connection. The peer name for a network connection is "<PEER_IP>:<PORT>". An example peer name for a Telnet connection is "10.82.80.71:19772", for a "Network Client" connection it is "10.82.80.71:1000", where the host port is configured to 1000. The peer name for the serial connection is "COM1" or "COM USB".
  - *return* – The boolean return value defines if the handler for this connection should be activated:
    - true: Enables customization of the communication protocol. Therefore, if you want to use your own protocol for communicating with the Dataman device, return true.
    - false: If you do not need the customized protocol for this peer, return false.
- ***onDisconnect*** – Cleanup method called when closing the connection channel
- ***onExpectedData*** – Method called if data matching the set properties has arrived. The method has one argument:
  - *inputString* – The received frame matched data excluding header and termination
  - *return* – Determines if the data should be removed from input buffer
    - true: clear the buffer.
    - false: keep the value in buffer.
- ***onTimer*** – The timer value expired
- ***onUnexpectedData*** – The recieved data is not matching the requirements. The boolean return value determines if the data should be removed from input. The method has one argument:
  - inputString – The received data
  - return – Determines if the data should be removed from input buffer
    - true: clear the buffer.
    - false: keep the value in buffer.
- ***onError*** – An error occurred in the firmware and may be reported. The implementation of the method is mandatory. The method has one argument and no return value:
  - *errorMsg* – The error message for trigger overruns ("Trigger Overrun"), buffer overruns ("Buffer Overflow") and general errors reported by the firmware.
- ***onEncoder*** – Executed if a configured encoder distance is reached. The distance can be configured by the **setEncoder** method. The method has no arguments or return value.
- ***send*** – Send data to channel, returns the number of send characters. The method must be called with one argument:
  - Data argument is a string
- ***close*** – Actively terminates connection for the communication object (for example, close TCP/IP socket). On UART, this causes **onConnect** to be called right afterwards.
- ***setTimer*** – Set the one-shot timer value when the onTimer will be executed. The timer can be re-initialized and aborted.
  - Timeout in seconds of type double, internal resolution is us (1e-6 sec). A zero value aborts a running timer. Active timer will be overwritten.

- ***expectFramed*** – Tells the communication listener which data to pass on to the **onExpectedData** and onUnexpectedData methods. It is possible to change the match parameter at runtime. The following three arguments are required:
  - *header* of type string, may be empty (””)
  - *terminator* of type string, may be empty (””)
  - *max length* of type integer, specifies the maximum length of an input message to check for a match [required]
- ***setEncoder*** – Units of the distance argument are millimetres. The encoder is configured in Setup Tool under System Settings -> Pulse Encoder. If encoder ticks should be used instead of distance set the value of the parameter “Resolution (mm)” to 1.
  - *distance (double)* – The encoder distance in which the **onEncoder** method will be called.

The methods must be implemented in the public section of the object.

Examples

API usage of custom communication protocol object

This example below demonstrates the API usage of the custom communication protocol object. The example implements custom commands read from the connection. The commands are framed by a ”#” header and terminated by ”;\r” (for example, a serial PuTTY connection). A timer sends periodically timer messages that may be stopped using the custom **stop** command. The timer handler may be changed once by the switch command.

```
function CommHandler()
{
      // private properties and methods:
      var num_trigger = 0;
      var num_send;
      // public properties and methods:
      function onTimeout()
      {
            num_send = this.send(my_name + ': timer callback\r\n');
            this.setTimer(1.0);
      }
      function onTimeout2()
      {
            today = new Date();
            var msg = today.getSeconds() * 1000 + today.getMilliseconds();
            num_send = this.send(my_name + ': time is: ' + msg + '\r\n');
            dmccCommand("TRIGGER", true);
            this.setTimer(1.0);
      }
      function replace_crlf(input_str)
      {
      return input_str.replace(/\r/g, '\\r').replace(/\n/g, '\\n');
      }
      return {
            onConnect: function (peerName)
            {
                  my_name = peerName;
                  // we may ignore the connection
                  if(my_name == "COM1")
                        return false;
            num_send = this.send(my_name + ": connected\r\n");

            this.expectFramed("#", ";\r\n", 64);
            return true;
      },
      onDisconnect: function ()
      {
      },
            onExpectedData: function (inputString) {
                  var msg = 'ok';
                  this.expectFramed("#", ";\r\n", 64);
                  if (inputString == "name")
                  {
                        msg = dmccGet("DEVICE.NAME");
                        msg = msg.response;
            }
            else if(inputString == "trigger")
```

```
{
      this.send(my_name + ': issue a trigger...\r\n');
       dmccCommand("TRIGGER", true);
      msg = 'done';
}
else if (inputString == "close")
{
      this.close();
}
else if (inputString == "stop")
{
      this.setTimer(0.0);
}
else if (inputString == "start")
{
      this.setTimer(10.0);
}
else if (inputString == "switch")
{
      this.onTimer = onTimeout2;
}
else if (inputString == "time")
{
      today = new Date();
      msg = today.getSeconds() * 1000 + today.getMilliseconds();
}
else
{
      msg = "unknown command: " + replace_crlf(inputString);
}
num_send = this.send(my_name + ': ' + msg + "\r\n");
return inputString.length;
},
onUnexpectedData: function (inputString) {
      this.expectFramed("#", ";\r\n", 128);
      msg = replace_crlf(inputString);
      num_send = this.send(my_name + ': ' + msg + "?\r\n");
      return true;
},
onTimer: onTimeout
};
}
```

Generic use case: Heartbeat

Send out a periodic heartbeat message if reader is idle.

```
// Data Formatting:

var comm_handler = new Array(0);

// Converts read data to all upper case. Single code only.
function onResult (decodeResults, readerProperties, output) {

      if (decodeResults[0].decoded) {

            output.content = decodeResults[0].content+'\r\n';
            for (var i = 0; i < comm_handler.length; i++)
            {

                  comm_handler[i].resetHeartBeat();

            }

      }

}

// Communication:

// Heart beat example without disturbing the DMCC communication function CommHandler() {

      var beat_timer = 10.0; // beat timer in sec
      var peer_name;

      return {

            onConnect: function (peerName)

            {

                  peer_name = peerName;
                  this.resetHeartBeat(); // initial timer
                  this.expectFramed("\0", "\0", 128); // some pattern
                  unlikely to happen
                  comm_handler.push(this); // register the handler for
                  results
                  // enable the handler for this connection:
                  return true;

            },
            onDisconnect: function ()
            {

                  var index = comm_handler.indexOf(this)
                  comm_handler.splice(index, 1);

            },
            onError: function (errorMsg)
            {

            },
            onExpectedData: function (inputString) {
            return false;
            },

            onUnexpectedData: function (inputString) {
            return false;
            },
```

```
            onTimer: function () {
                  today = new Date();
            var msg = today.getSeconds() * 1000 + today.getMilliseconds();
            num_send = this.send(peer_name + ': time is: ' + msg + '\r\n');
            this.resetHeartBeat(); // schedule next timer event [sec]
            },
            resetHeartBeat: function () {
            this.setTimer(beat_timer); // schedule next timer event [sec]
            }
      };
}
```

Generic use case: Real time timestamp

Implements a custom DMCC command to set the real time (send current time in seconds starting Jan 1 1970, as output by date +"%s" command). Prepend output with real time timestamp.

```
// communication script
var time_offset=0;

function CommHandler()
{
      var peer_name;


      return {
            onConnect: function (peerName)
            {
                  peer_name = peerName;
                  this.expectFramed("||;1>SET TIME.NOW ", "\r\n", 128); //
                  some pattern unlikely to happen
                  // enable the handler for this connection:
                  return true;
            },
            onDisconnect: function ()
            {
            },
            onError: function (errorMsg)
            {
            },
            onExpectedData: function (inputString) {
                  realTime = parseInt(inputString)*1000;
                  localTime = (new Date()).getTime();
                  time_offset = realTime - localTime;
                  this.send("||[0]\r\n");
                  return true;
            },
```

```
            onUnexpectedData: function (inputString) {
                  return false;
            },
            onTimer: function () {
            }
            };
}

// data formatting script
function onResult (decodeResults, readerProperties, output)
{
      var d = new Date();
      var real = new Date(time_offset+d.getTime());
      output.content = real.toString() + " " + decodeResults[0].content + "\r\n";
}
```

Customer Protocols implemented in various CR releases

Communication with cmf400 Profibus gateway

```
var CMF400_PROTOCOL_STATUS =
{
      RUNNING: {value: 0, name: "Running"},
      SYNCRONIZING: {value: 1, name: "Sync"},
      CONFIGURING: {value: 2, name: "Config"},
      STOPPED: {value: 3, name: "Stop"}
};
// make the enum non-modifyable
Object.freeze(CMF400_PROTOCOL_STATUS);

var cmf400_protocol_stx = '\x02'; // header
var cmf400_protocol_etx = '\x03'; // termination

// VT Parameter to be converted into script configuration constant values:
// "/Communication/Interfaces/COM1/Protocol")
var vt_param_comif_com1_protocol = 1;
// "/Communication/Protocols/CMF400/Profibus node number"), 3);
var vt_param_profibus_node_number = 1;
// "/Communication/Protocols/CMF400/Profibus mode"), 3);*/
var vt_param_profibus_mode = 1;

// TODO: how to configure parameter, where to store them with a out of stock firmware?
var cmf400_protocol_profibus_node_number = 1;

var cmf400_protocol_profibus_mode = 1;

var cmf400_protocol_test_diagnostic_enabled = 0;

var cmf400_protocol_test_diagnostic = 'TEST';

// Protocol strings
var cmf400_gateway_init = '+Gateway-Init+';
var cmf400_gateway_ident_ok = '+GW SOK TSICDPS';
var cmf400_gateway_ident_no = '+GW SNO TSICDPS';
var cmf400_gateway_run = '+GW-RUN+';
var cmf400_gateway_error = '+GW-ERR';
```

```
// Formatting helper function
function zero_prefix(num, size)
{
      var s = "0000000000" + num;
      return s.substr(s.length - size);
}

function CommHandler()
{
      // The current protocol state
      var cmf400_status = CMF400_PROTOCOL_STATUS.STOPPED;

      function _configTimedOut()
      {
            if (cmf400_status == CMF400_PROTOCOL_STATUS_CONFIGURING)
            {
            cmf400_status = CMF400_PROTOCOL_STATUS_STOPPED;
            this.setTimer(30.0);
            onTimer = _onSync;
            }
      }
      function _onSync()
      {
            if (cmf400_status == CMF400_PROTOCOL_STATUS.SYNCRONIZING)
            {
                  this.send(cmf400_protocol_stx + cmf400_gateway_init +
                  cmf400_protocol_etx);
                  this.setTimer(1.0);
                  onTimer = _onSync;
            }
      }
      function _onTimer()
      {
            if (cmf400_status == CMF400_PROTOCOL_STATUS.STOPPED)
            {
            cmf400_status = CMF400_PROTOCOL_STATUS.SYNCRONIZING;
            return;
            }
      }
            return {
                        onConnect: function (peerName)
                        {
                        expectFramed("", cmf400_protocol_etx, 510); //
                        is 510 an arbitrary limit?
                        cmf400_status = CMF400_PROTOCOL_
                        STATUS.SYNCRONIZING;
                        this.onTimer = _onSync;
                        this.setTimer(0.0001);
                        return true;
```

```
        },
        onDisconnect: function ()
        {

        },
        onExpectedData: function (inputData)
        {
              data = inputData.slice(1,inputData.length-1);
              if (cmf400_status == CMF400_PROTOCOL_STATUS.SYNCRONIZING)
              {
                    if (data == cmf400_gateway_ident_ok || data ==
                    cmf400_gateway_ident_no)
                    {
                          cmf400_status = CMF400_PROTOCOL_
                          STATUS.CONFIGURING;
                          var msg = cmf400_protocol_stx;

                          msg += "+GW S000 H000";
                          msg += " X" + zero_prefix(vt_param_
                          comif_com1_protocol, 3);
                          msg += " N" + zero_prefix(vt_param_
                          profibus_node_number, 3);
                          msg += " M" + zero_prefix(vt_param_
                          profibus_mode, 3);
                          msg += cmf400_protocol_etx;
                          this.send(msg);
                          this.onTimer = _configTimedOut;
                          this.setTimer(10.0);

                    }

              }
              if (data == cmf400_gateway_error)
              {
                    cmf400_status = CMF400_PROTOCOL_
                    STATUS.STOPPED;
                    this.setTimer(30.0);
                    this.onTimer = _onTimer;

              }
              else if (data == cmf400_gateway_run) // missing check for
              status, e.g. CMF400_PROTOCOL_STATUS.CONFIGURING?
              {
                    cmf400_status = CMF400_PROTOCOL_STATUS.RUN;
                    this.setTimer(0);
                    this.onTimer = _onTimer;

              }
              return true;

        },
        onUnexpectedData: function (inputData)
        {
              // ignore all unexpected data
              return true;

        },
        onTimer: _onSync

    };
}
```

```
function onResult (decodeResults, readerProperties, output)
{
      //assuming single code
      var content = cmf400_protocol_stx+decodeResults[0].content+cmf400_protocol_etx;
      output.content = content;
}
```

Pass weight string input along with decode string

```
// the constructor:

var input_string = "";

function CommHandler()
{
      // private properties and methods:

      var num_trigger = 0;
      var my_name;
      var num_send = 99;

      function privFunc ()
      {
      }

      // public properties and methods:

      return {
            onConnect: function (peerName)
            {
                  my_name = peerName;
                  num_send = this.send(my_name + ": connected\r\n");
                  num_send = this.expectFramed("\x02", "\x03", 128);
                  return true;
            },
            onDisconnect: function ()
            {
            },
            onExpectedData: function (inputString) {
            input_string = inputString;
            return true;
            },
            onUnexpectedData: function (inputString) {
            return true;
            }
      };
}
```

```
//Empty data formatting entry point function
function onResult (decodeResults, readerProperties, output)
{
      if (decodeResults[0].decoded)
      {
            output.content = input_string + decodeResults[0].content + "\r\n"
            input_string = "";
      }
}
```

FMPCS protocol

```
// This must be in the global scope, otherwise, it is undefined
var bConnected = false;

dmccSet('TRIGGER.TYPE', 0);
dmccSet('SYMBOL.4STATE-IMB', 1);
dmccSet('SYMBOL.DATAMATRIX', 1);
dmccSet('SYMBOL.I2O5', 1);
dmccSet('SYMBOL.PDF417', 1);
dmccSet('SYMBOL.POSTNET', 1);

function CommHandler()
{
      var tray = "0000";
      var speed = 0;

      var package_id_expr = new RegExp("I([0-9]{9})");
      var package_idtray_expr = new RegExp('^I([0-9]{9}),T([0-9]{4})');
      var config_msg_expr = new RegExp('^CS([0-9]{3}),M([ab]),L([0-9]{4})$');

      var ErrorToId = {
            'Buffer Overflow': 101,
            'Trigger Overrun': 102
            };


      return {
            onConnect: function (peerName)
            {
                  if(peerName == "COM1" || bConnected)
                  return false;
                  this.expectFramed("", "\r", 128);
                  this.send(dmccGet('DEVICE.FIRMWARE-VER').response +
                  ',"Cognex ' + dmccGet('DEVICE.TYPE').response + '"\r\n');
                  this.send('Ha,"DataMan READY"\r\n');
                  bConnected = true;
                  return true; // activate this connection
            },
            onError: function (msg) // TODO: this is new!
            {
```

```
var errno = ErrorToId[msg];
if (!errno)
errno = 100;
this.send('E' + errno + ',"' + msg + '"\r\n');
},
// We delay sending the result until trigger off to be sure that the
package id is received.
setResult: function (decodeResults) {
storedDecodeResults = decodeResults;
},
onDisconnect: function ()
{
bConnected = false;
},
onExpectedData: function (input)
{
      var input = input.replace(/\n/g, '');
      switch(input.charAt(0).toUpperCase())
            case 'B':
                  dmccCommand("TRIGGER", true);
                  break;
            case 'E':
                  dmccCommand("TRIGGER", false);
                  break;
            case 'I':
                  var match = package_idtray_
                  expr.exec(input);
                  if(!match)
                        match = package_id_
                        expr.exec(input);
                  packageID = match[1];
                  if(match[2])
                        tray = match[2];
                  else
                        tray = "0000";
                  break;
```

```
                        case 'C':

                        var match = config_msg_expr.exec(input);
                        if (match.length == 4)
                        {

                              speed = parseInt(match[1], 10);
                              mode = match[2];
                              lengthLimit = parseInt(match[3],
                              10);

                        }
                        break;

                  case 'P':

                        this.send('Q\r\n');
                        break;

                  case 'Q':

                        // pong response, not used
                        break;

            }
            return true;

      },
      onUnexpectedData: function (input) {
      return true;
      }

};
}
```

The data formatting formats the result based on global variables set by the communication handler:

```
var packageID = "000000000"; // reset the package id
var mode = 'a';
var lengthLimit = 9999;

function getFixedPsocId(id_)
{
      var id = id_;
      switch (id.charAt(1))
      {

            case 'd':

                  id = "[D0";
                  break;

            case 'X':

                  switch (id.charAt(2))
                  {
```

```
                        case '0':
                        case '1':
                        id = "[P0";
                        break;
                        case '2':
                        case '3':
                        id = "[L0";
                        break;
                        case '5':
                        case '6':
                        case '7':
                        case '8':
                        case '9':
                        case 'A':
                        id = "[O0";
                        break;

                  }
                  break;

      }
      return id;

}
function onResult (decodeResults, readerProperties, output)
{

      var my_decode_results = new Array();

      for(var i = 0; i < decodeResults.length; i++)
      {

            if(!decodeResults[i].decoded)
            continue;
            switch (decodeResults[i].symbology.name)
            {

                  case 'Interleaved 2 of 5':

                        // b=throw away 6 digit I2of5 ending in 9
                        if ((mode == 'b' && decodeResults
                        [i].content.length == 6 && decodeResults
                        [i].content.charAt(5) == '9'))

                              continue;

                  case 'Data Matrix':

                        if (decodeResults[i].content.length >
                        lengthLimit)

                              continue;

                  case 'PDF417':

                        if (decodeResults[i].content.length >
                        lengthLimit)

                              continue;

                  default:

                        my_decode_results.push(decodeResults[i]);

            }

      }
```

```
      var msg = 'D' + packageID + ',S,W,V';
      if (my_decode_results.length == 0)
      {
            msg += ',?';
            output.content = "no result";
      }
      else
      {
            for(var i = 0; i < my_decode_results.length; i++)
            {
                  msg += ',' + getFixedPsocId(decodeResults
                  [i].symbology.id);
                  switch (my_decode_results[i].symbology.name)
                  {
                        case 'Data Matrix':
                        case 'PDF417':
                        msg += encode_base64(my_decode_results
                        [i].content);
                        break;
                        case 'POSTNET':
                        case 'PLANET':
                        case 'XYZ OneCode':
                        case 'Interleaved 2 of 5':
                        default:
                              msg += my_decode_results
                              [i].content;
                  }
            }
      }
      packageID = "0000000000"; // reset the package id
      output.Telnet = output.Serial = msg + '\r\n';
}
```

Input match string, output from script (30x)

```
function CommHandler()
{
      return {
            onConnect: function (peerName)
            {
                  this.expectFramed('\x02', '\x03', 256);
                  return true;
      },
      onDisconnect: function ()
      {
      },
      onExpectedData: function (inputString) {
```

```
            if (inputString.length >= 11)
            {
                  var new_match_string = inputString.substr(11,
                  inputString.length);
                  for (var i = 1; i <= 3; i++) {
                        dmccSet("DVALID.PROG-TARG", i);
                        dmccSet("DVALID.MATCH-STRING", new_match_
                        string);
                  }
                  // The following DMCC command resets all statistic values
                  // the CR reset only a view of them
                  dmccCommand("STATISTICS.RESET");
            }
            this.send("DEBUG: "+inputString + "\r\n");
            return true;
      },
      onUnexpectedData: function (inputString) {
            return true;
      },
      onTimer: function (inputString) {
      }
};
}
```

Data formatting delegates output to communication handler objects

```
var comm_handler = new Array(0);

// Converts read data to all upper case. Single code only.
function onResult (decodeResults, readerProperties, output)
{
      output.content = '';
      output.SetupTool = decodeResults[0].content;
      if (decodeResults[0].decoded) {
            for (var i = 0; i < comm_handler.length; i++)
            {
            comm_handler[i].sendResultTelegram(decodeResults);
                  }
      }
}
```

```
// Parameter:
var system_id = '\x43'; // the system ID
var heartbeat_time_s = 5.0; // heartbeat timer in sec [0-50] (0 is disabled)
var append_crlf = true; // wether to

function CommHandler()
{
      function getChecksum(data)
      {
            var sum = 0;
            for(var i = 0; i < data.length; i++)
            sum += data.charCodeAt(i);
            return 0x7F - (sum % 0x7f);
      }
      var TelegramState = {
            WAIT4CONTENT: {value: 0, name: "Wait For Content"},
            CHECKSUM: {value: 1, name: "Header Received"}
      };


      var errorCodes = {
            undef_index: 0x31,
            multi_index: 0x32,
            index_in_use: 0x33,
            telegram_error: 0x34,
            trigger_overrun: 0x40,
            buffer_overflow: 0x41,
      };


      var filler = '#';
      var separator = ',';

      var telegram_types = {
            heartbeat: {type: 'F', content: system_id+'\xf7'},
            init_resp: {type: 'J', content: system_id},
            };


      // initialization: J
      // index: S


      var telegram;
      var status;
      var index;
      var all_index = new Array();

      return {


            sendResultTelegram: function (decodeResults)
            {
                  var data = system_id;
                  var length = 0;
```

```
            if (!index)
            {
this.sendErrorTelegram(errorCodes.undef_index);
            index = '9999';
      }
      data += index;
      for (var i = 0; i < decodeResults.length; i++) {
            length = decodeResults[i].content.length;
            data += String.fromCharCode(length / 256, length % 256);
      }
      data += separator + filler;
      length = 0;
      for (var i = 0; i < decodeResults.length; i++) {
            length += decodeResults[i].content.length;
            data += decodeResults[i].content;
      }
      if (length & 0x1)
            data += filler;
      data += String.fromCharCode(getChecksum(data));
      this.sendTelegram({type: system_id, content: data});


      index = null; // invalidate the used index
},
sendErrorTelegram: function (errcode)
{
      var errtel = {type: 'F', content: system_id+String.fromCharCode
      (errcode)}


      this.sendTelegram(errtel);
},
sendTelegram: function (telegram)
{
      var data = telegram.type + telegram.content;
      data = '\x02'+data+String.fromCharCode(getChecksum(data))+'\03';
      this.send(data);
      if (append_crlf)
            this.send('\r\n');
},
checkTelegram: function(data, checksum)
{
      var exp_checksum = getChecksum(data);
      if (checksum != exp_checksum) {
this.sendErrorTelegram(errorCodes.telegram_error);
      } else {
            switch (data[0])
            {
                  case 'I':
```

```
      this.sendTelegram(telegram_types.init_resp);
                              this.setTimer(0.0); // disable the
                              heartbeat timer
                              all_index = new Array(0);
                              break;
                              case 'S':
                              if (index) {
      this.sendErrorTelegram(errorCodes.multi_index);
                                    break;
                        }
                        index = data.substr(1, 4);
                        if (all_index.indexOf(index) >= 0)
this.sendErrorTelegram(errorCodes.index_in_use);
                        else
                              all_index.push(index);
                        break;
            default:
                  break;
            }
      }
},
      onConnect: function (peerName)
      {
            status = TelegramState.WAIT4CONTENT;
            this.expectFramed('\x02', '\x03', 203);
            this.setTimer(heartbeat_time_s);
            index = null;
            comm_handler.push(this);
            all_index = new Array();
            return true;
      },
      onDisconnect: function ()
      {
            var index = comm_handler.indexOf(this)
            comm_handler.splice(index,1);
      },
      onExpectedData: function (inputString) {
            switch (status)
            {
            case TelegramState.WAIT4CONTENT:
                  this.expectFramed('', '', 1); // actually, disable framing
                  telegram = inputString;
                  status = TelegramState.CHECKSUM;
                  break;
            case TelegramState.CHECKSUM:
                  this.expectFramed('\x02', '\x03', 203); // enable framing
                  for the next telegram
                  this.checkTelegram(telegram, inputString.charCodeAt(0));
                  status = TelegramState.WAIT4CONTENT;
                  break;
```

```
            default:
                  throw("unknown state");
            }
            return true;
            },
            onUnexpectedData: function (inputString) {
            this.expectFramed('\x02', '\x03', 203); // enable framing for the
            next telegram
            status = TelegramState.WAIT4CONTENT;
            return true;
      },
      onTimer: function (inputString) {
      this.sendTelegram(telegram_types.heartbeat);
      this.setTimer(heartbeat_time_s);
      }
};
}
```

Event Callback

The callback mechanism allows to register handler for trigger and input events. Handler for these events can be registered by the registerHandler method:

```
callback_handle registerHandler(eventid, callback, ...)
```

The **registerHandler** function requires the following arguments:

- *eventid* – identifier for the event type to register for
- *callback* – function object to execute on event

Available events identifier are defined in a constant object named “Callback”. Optional arguments can be used to configure the event, e.g. to filter the sensitivity.

A handle is returned that must be used to de-register the callback. To de-register the handler use the **deregisterHandler** function:

```
deregisterHandler(callback_handle)
```

- *callback_handle* – handle returned by the registerHandler method.

It is possible to register the callback handler within the global scope, e.g. to be used in data formatting.

Event Types

Current available events that can be registered are “onInput” and “onTrigger” events.

onInput event: It calls the callback function on input signal and button changes. The optional third argument allows to set filter for certain inputs. The object “ConstInput” defines masks for inputs:

- Input0:
- Input1:
- Input2:
- Input3:
- Input4:
- Input5:

- Input6:
- InputAll
- BnTrig
- BnTune

The input mask can be combined. The input values are sampled with an accuracy of 1 ms. The callback function for the onInput event has one argument for the new state of the input.

onTrigger event: It executes the callback function on trigger start and trigger end events. The callback function for the onTrigger event has two arguments: The first argument is the trigger object, the second argument the boolean state of the trigger, true for a trigger start and false for a trigger end.

Examples

The example defines three event handler:

- *onInput0* – reacting on input0 signal and the switch button
- *onInput1* – reacting on input1 signal
- *onTrigger* – reacting on trigger events

```
function CommHandler()
{
      return {
            onConnect: function (peerName)
            {
                  this.peer = peerName;
                  this.input1 = registerHandler(Callback.onInput,
                  this.onInput0.bind(this),
                  ConstInput.Input0|ConstInput.BnTrig);
                  this.input2 = registerHandler(Callback.onInput,
                  this.onInput1.bind(this), ConstInput.Input1);
                  this.ontrigger = registerHandler(Callback.onTrigger,
                  this.onTrigger.bind(this));
                  return true;
            },
      onDisconnect: function ()
      {
            deregisterHandler(this.input1);
            deregisterHandler(this.input2);
            deregisterHandler(this.ontrigger);
      },
      onTrigger: function (trigger, state) {
            if (state)
                  this.send("call onTrigger: started trigger with index " +
                  trigger.index + "\r\n");

            else
                  this.send("call onTrigger: end trigger with index " +
                  trigger.index + "\r\n");
```

```
      },
      onInput0: function (inputs) {

            this.send("call onInput0 for '" + this.peer + ", inputs=" + inputs +
            "\r\n");

      },

      onInput1: function (inputs) {

            this.send("call onInput1 for '" + this.peer + ", inputs=" + inputs +
            "\r\n");

      }

};
}
```

With the following event sequence: input1 on, input0 on, input0 off, input1 off, software trigger, switch on, switch off, we get the following output on the terminal:

```
call onInput1 for 'COM1, inputs=2
call onTrigger: start trigger with index 9
call onInput0 for 'COM1, inputs=1
call onTrigger: end trigger with index 9
call onInput0 for 'COM1, inputs=0
NO-READ
call onInput1 for 'COM1, inputs=0
call onTrigger: start trigger with index 10
NO-READ
call onTrigger: end trigger with index 10
call onInput0 for 'COM1, inputs=4096
call onTrigger: start trigger with index 11
call onInput0 for 'COM1, inputs=0
call onTrigger: end trigger with index 11
NO-READ
```

The following example registers a handler on Input1 events and stores the state in a global variable. The state of the input is output by the data formatting.

```
var ginputs = false;

registerHandler(Callback.onInput, onInput, ConstInput.Input1);

// Default script for data formatting
function onResult (decodeResults, readerProperties, output)
{

      output.content = "Input: "+ginputs+" \r\n";

}

function onInput(inputs)
{

      ginputs = (inputs & ConstInput.Input1) ? true : false;

}
```