COGNEX

DataMan® Communications and Programming Guide

Legal Notices

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](#)
- [DataMan Control Commands (DMCC) API](#)

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

- using the DataMan Setup Tool
- trigger bits
- through a DMCC command
- manipulating objects through industrial protocols

For information on industrial protocols, see the *DataMan Industrial Protocol Manual*. All the other methods are explained in detail in this document.

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 Fixed-Mount DataMan Reader to the Network

Supply power to the reader using either a Power over Ethernet (PoE) injector (DataMan 260 series readers) or the Power and I/O breakout cable (CCB-PWRIO-xx).

Cognex recommends the following connection sequence in case of using a PoE injector:

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.
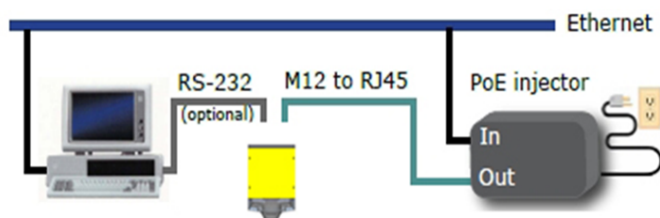
To disconnect the reader:

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 Handheld DataMan Reader to the Network

If you are using an Ethernet slide-in with your handheld DataMan reader, power your reader through Power over Ethernet (PoE) and connect the Ethernet cable of the PoE to the network.
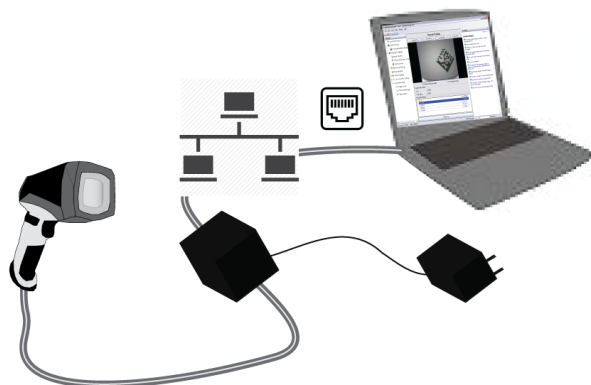
If you are using a serial slide-in with your handheld DataMan reader, connect the serial cable to your PC, power your reader through an external power supply and connect your PC to the network.
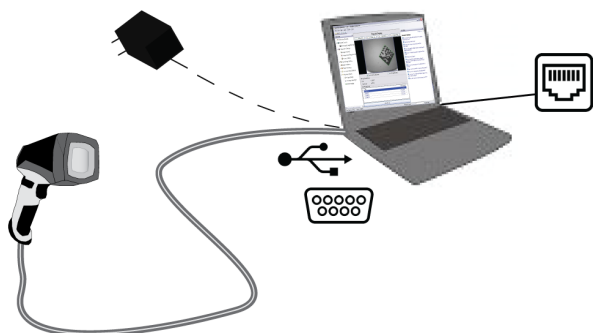
Connecting Your DataMan to the Network Wirelessly

You can connect to your DataMan reader via the wireless network as well. For this, you need to use the Wi-Fi 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.
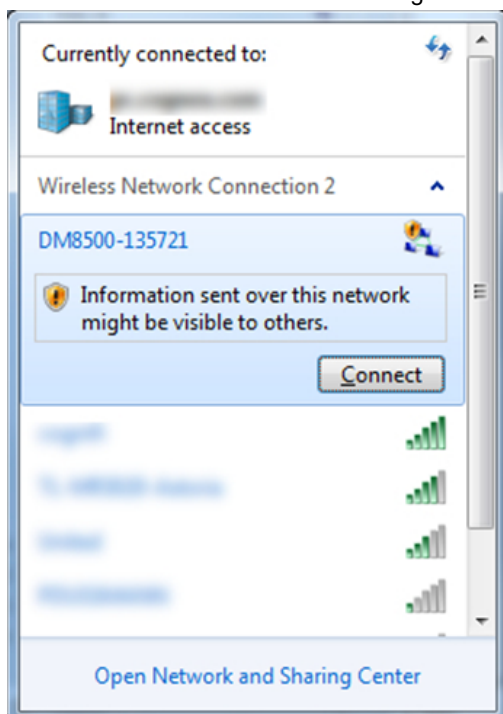
3. Open the DataMan Setup Tool.

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 results in the misconfiguration of the device.

Infrastructure Mode
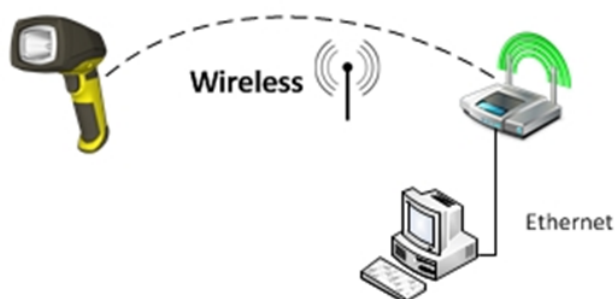
3. Select from the following authentication modes:

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 are 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 (for example, on a company webpage).

- CA's certificate (CA = Certificate Authority). One such file is created for each authentication server within the company. It can be publicly accessible.

- Client's private key. This must be different for each reader. It must not be publicly accessible, and 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 and select the appropriate file to upload it 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

In the DataMan Setup Tool, the following restrictions apply to 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 an unencrypted private key and certificate files are allowed.

- The Client's private key and certificate must contain exactly one section; the CA's certificate can contain one or more certificates.

- Make sure that you know the user name stored within your own certificate file, and use the same name in the **Client's user name** text box. This is necessary because the Setup Tool does not look into the certificate files to extract this user name information.
  When you leave the Wireless tab, a reboot confirmation window pops up and the settings are saved to the device.

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:

3-pin terminal block
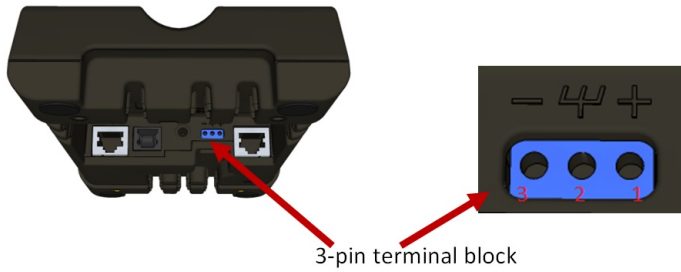
Pin #	Signal
1	+24V
2	Shield
3	GND

ⓘ **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*

Ethernet connection
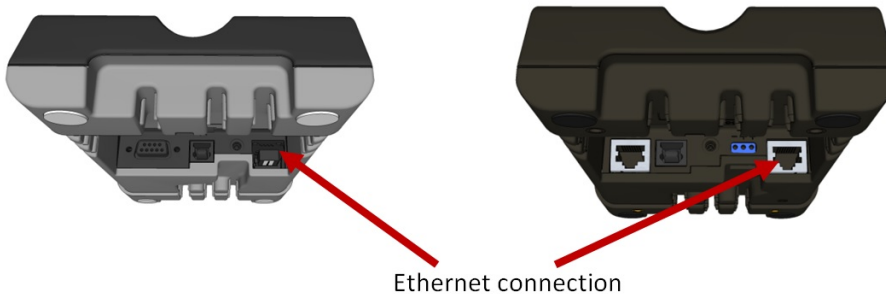
3. The base station becomes visible as connected through Ethernet, and it routes data through the wireless (Wi-Fi or Bluetooth) interface to the reader.
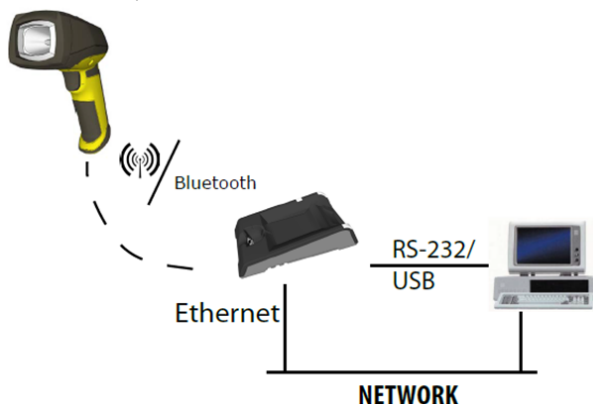
Direct Connection to Your Computer

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. This option is detailed in the following section.

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

In the DataMan Setup Tool's **Communications** application step's **Ethernet** tab, click the **Copy PC Network Settings** button. Choose the network you want to use and the settings will be copied in the **Use Static IP Address** window.

> (i) **Note**: Remember to update the IP address of your DataMan device. The IP address that is copied belongs to your PC.

A triangle with an exclamation mark in the upper right-hand corner reminds you that you have to reboot the device in order for the changes to take effect.
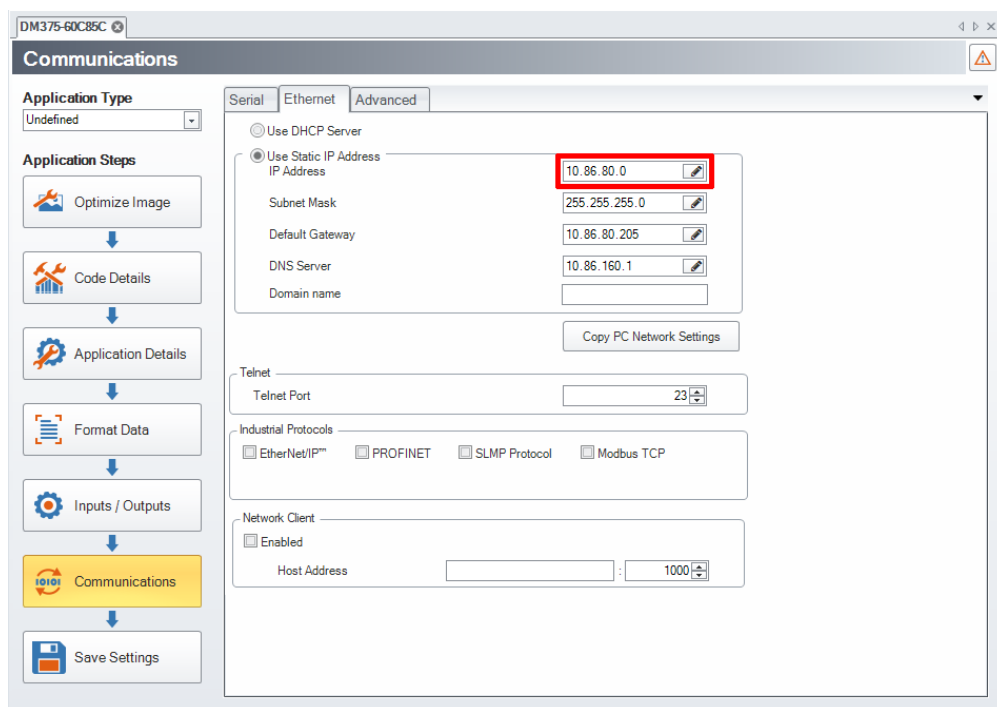
To force the network settings on your DataMan, select **Use Static IP Address** and enter an IP Address and Subnet Mask that will be on the same subnet as the PC. Make sure this IP address is not yet in use (for example, test by pinging it). For example:

- IP Address: 169.254.135.200
- Subnet Mask: 255.255.0.0

> (i) **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.

Your DataMan device is now configured to the specified network settings, and it reboots automatically. After the address has been resolved, your DataMan device appears under the **Network** node. This can take up to 60 seconds. 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**.
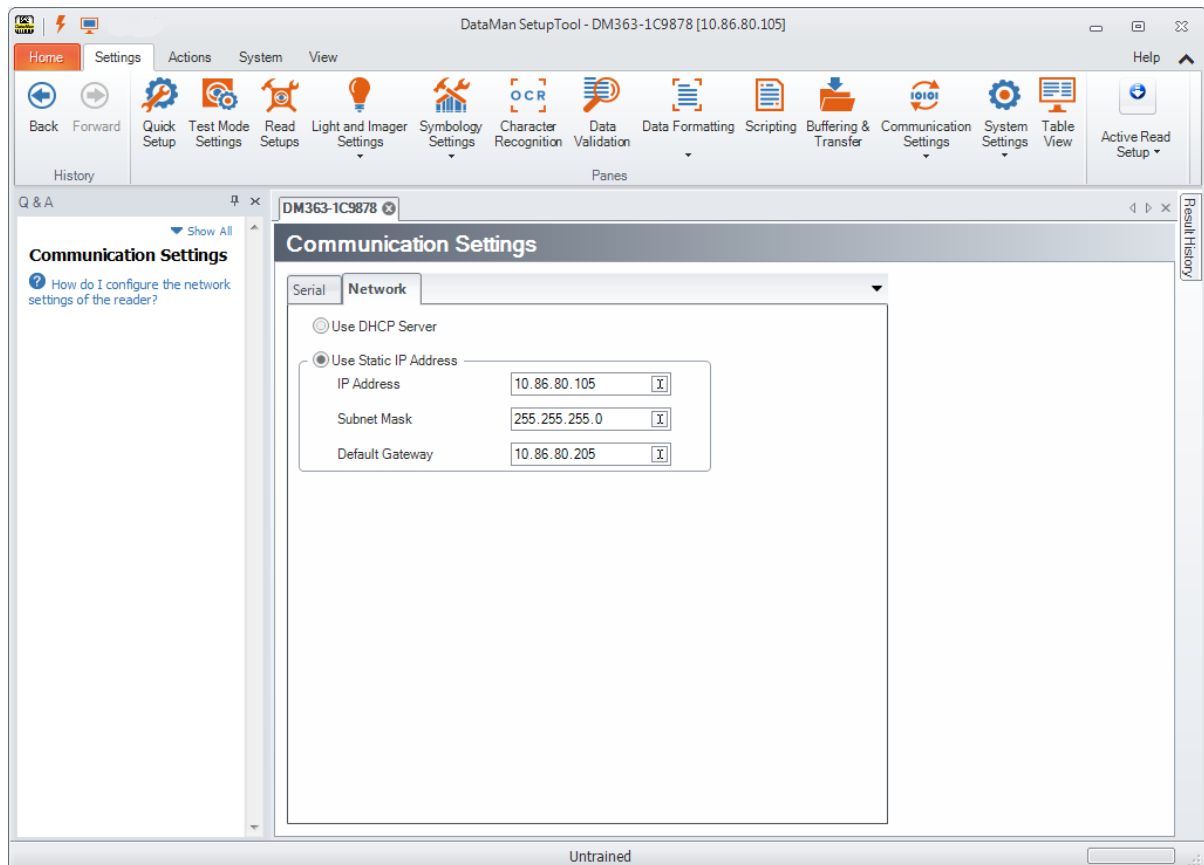
Once the IP Address and Subnet Mask of the DataMan device are known, the PC's network settings can be changed.

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

1. In the Start Menu, start typing Control Panel and open it.

2. Click **Network and Internet**.

3. Click **Network and Sharing Center** and under active networks, click **Ethernet**.
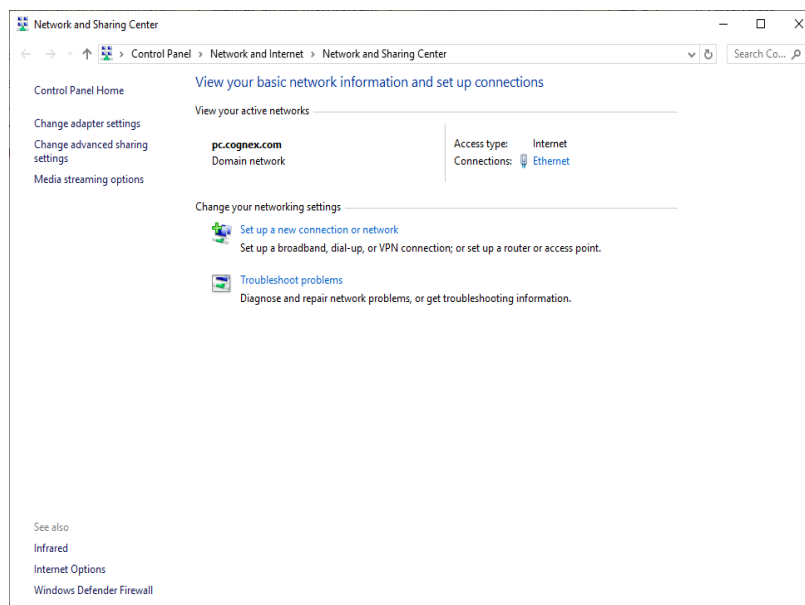
4. In the **Ethernet Status** window that pops up, click **Properties**.
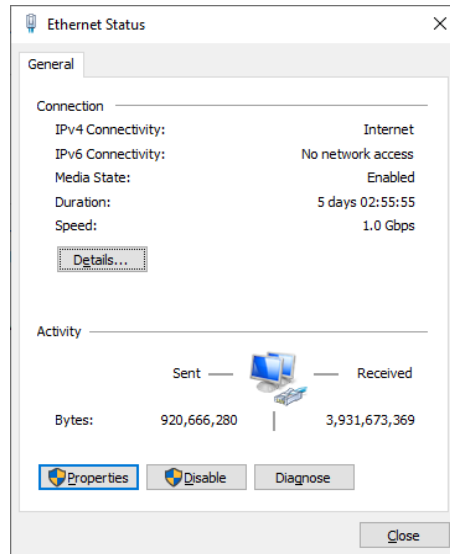
5. In the **Ethernet Properties** window that pops up, select **Internet Protocol Version 4 (TCP/IPv4)** and click **Properties**.
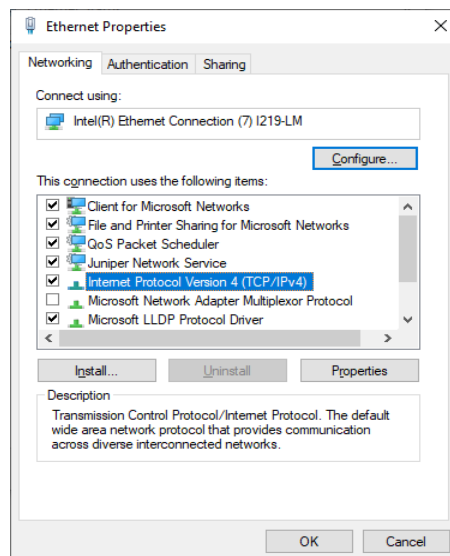
6. Under the **General** tab, select the **Use the following IP address** option and enter an IP address and Subnet mask that are on the same subnet as your DataMan. Click **OK**.
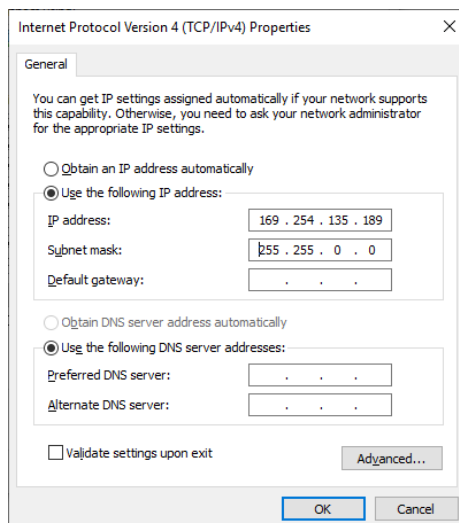
7. Click **Close**. The network settings of your PC will change to the new specified values.

8. Reboot the DataMan device. It appears under the **Discovered Devices** node on the **Connect** page after the network address has been resolved.

9. 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**.
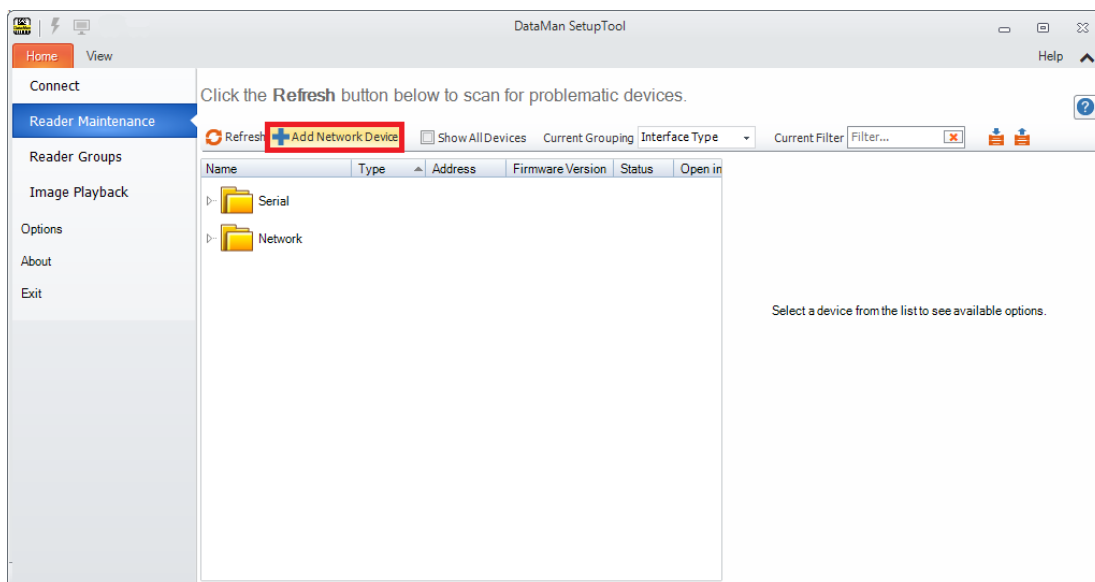
2. Enter the IP Address of the target DataMan device.

Add / Edit Network Device

IP Address: 169.254.135.200

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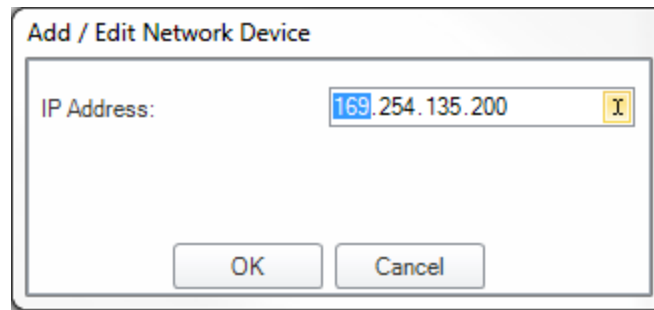
3. Click OK. The reader appears under the **Network** node. Double click the new node or select it and click the **Connect** button. If the device is available, the reader will be connected.

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.

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.

> (i) **Note**: 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 can 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.

> (i) **Note**: Use a third party client such as PuTTY to make changes to the Telnet client provided by Windows to communicate with the DataMan.

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 can 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

DataMan SDK 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:
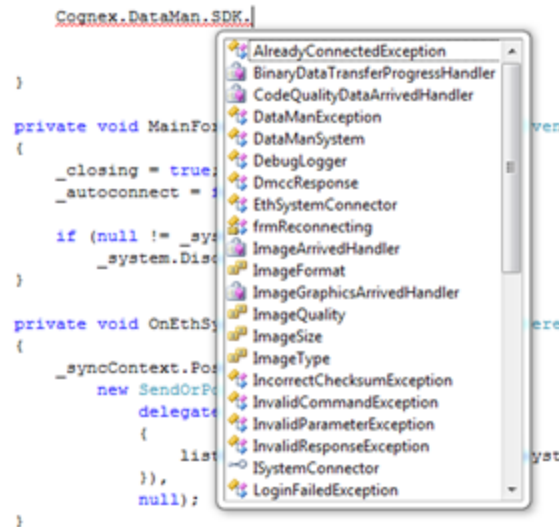
Enumerating DataMan Devices

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();
```

> (i) **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");
```

> (i) **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, and so on.

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, and so on.)

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, for example 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

(i) **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.
trucheck	TruCheckProperties	Outputs the TruCheck quality values for verification. The values of this property are listed in TruCheck Properties table listed below.

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.

> (i) **Note**: The following TruCheck metrics are only available for devices with TruCheck verifier capability, such as the DM475 Verifier and the DM8072 Verifier.

TruCheckMetric

A graded verification parameter that has a measurement associated with it.

Property	Type	Description
grade	string	The grade of quality in a range from grade A to F, where A is the highest.
raw	float	The raw metric.
numericGrade	float	The continuous grade value as shown in ISO 15416:2016.

TruCheckMetricGradeOnly

A graded verification parameter that has a measurement associated with it.

Property	Type	Description
grade	string	The grade of quality in a range from grade A to F, where A is the highest.
numericGrade	float	The continuous grade value as shown in ISO 15416:2016.

TruCheckApplicationStd

Property	Type	Description
grade	string	The data title for each section of the parsed data.
data	string	The raw parsed data.
check	string	A Pass/Fail assessment determined by the specific application standard selected.

TruCheckCodeword

Property	Type	Description
codeword	integer	Outputs the codewords associated with the parsed data.
isCorrected	boolean	Returns a 0 if the codeword is not corrected using error correction; returns a 1 if the codeword is corrected using error correction.

TruCheckEncodationAnalysis

Property	Type	Description
name	string	Outputs the codeword for the parsed data.
mode	string	Outputs the encodation mode in effect.
result	string	Outputs the raw data decoded.

TruCheckMetricGeneral

Property	Type	Description
contrastUniformity	integer	The contrast uniformity value is the minimum modulation of any codeword according to ISO 15415.
horizontalBWG	integer	Print growth in the horizontal axis.
MRD	integer	Minimum difference between the brightest bar and the darkest space within the symbol.
verticalBWG	integer	Print growth in the vertical axis.
xDimension	float	The normal cell size spacing in mils (thousandths of an inch).

TruCheckMetricModulation

Property	Type	Description
raw	integer	Outputs the raw modulation values for each module.
grade	string	Outputs the letter grade (A-F) for each module.
isBlack	boolean	Outputs a 0 if the module is white or a 1 if the module is black.

TruCheckMetricOverall

Property	Type	Description
aperture	integer	The aperture in mils used to verify the symbol.
applicationStandardName	string	The application standard used to verify the symbol.
applicationStandardPass	string	The PASS or FAIL assessment of the symbol according to the application standard.
gradeLetter	string	The overall letter grade (A-F).
gradeValue	float	The overall numeric grade (0.0-4.0).
gradingStandard	string	The grading standard used to verify the symbol.
lighting	string	The lighting option selected for verification.
wavelength	integer	The wavelength used for verification in nm.

TruCheckResult

Outputs the TruCheck values used for verification.

Property	Type	Description
alignmentPatterns	TruCheckMetricGradeOnly	The grade values for the alignment pattern of a QR Code symbol.
applicationStdArray	TruCheckApplicationStd	An array of parsed data according to the selected application standard.
applicationStdNotation	string	The notes associated with the application standard data parsing.
asciiArray	array of integers	The ASCII values for the symbol.
averageGrade	TruCheckMetric	Data Matrix fixed pattern damage metric that averages multiple segments of its finder pattern.
axialNonUniformity	TruCheckMetric	The axial nonuniformity (ANU) which is the aspect ratio grade.
batch	string	The batch number, if parsed from the symbol data.
bottomLSide	TruCheckMetricGradeOnly	The grade for the bottom of the L-side of the symbol.
bottomQuietZone	TruCheckMetricGradeOnly	The grade for the bottom quiet zone.
calibrationDate	string	The last date of calibration.
cellContrast	TruCheckMetric	The cell contrast value according to AIM-DPM (ISO 29158).
cellModulation	TruCheckMetricGradeOnly	The cell modulation value according to AIM-DPM (ISO 29158).
codewordArray		The array of codeword values.
decode	TruCheckMetricGradeOnly	The success or failure of the reference decode algorithm.
distributedDamageGrade	TruCheckMetric	The distributed damage grade parameter.
encodationAnalysisArray	TruCheckMetricGradeOnly	The array of encodation analysis values.
fixedPatternDamage	TruCheckMetric	The fixed pattern damage parameter according to ISO 29158 (AIM-DPM).

Property	Type	Description
formatInformationBlock	TruCheckMetricGradeOnly	The grade for the format information block of a QR code.
general		A structure containing the general characteristic information.
gridNonUniformity	TruCheckMetric	The grid nonuniformity (GNU) grade according to ISO 15415.
horizontalClockTrack	TruCheckMetricGradeOnly	The grade for the horizontal clock track.
jpegImage	string	The jpeg image of the symbol encoded as a base64 string.
LeftLSide	TruCheckMetricGradeOnly	The grade for the left L-side of the symbol.
leftQuietZone	TruCheckMetricGradeOnly	The grade for the left quiet zone of the symbol.
linearDecodability	TruCheckMetric	The decodability for linear (1D) symbols.
linearDecode	TruCheckMetricGradeOnly	The decode grade for linear (1D) symbols.
linearDefect	TruCheckMetric	The defect average value for linear (1D) symbols.
linearEdge	TruCheckMetric	The edge value for linear (1D) symbols.
linearMinimumEdgeContrast	TruCheckMetric	The minimum edge contrast (minEC) for linear (1D) symbols.
linearMinimumReflectance	TruCheckMetricGradeOnly	The minimum reflectance (minRefl) for linear (1D) symbols.
linearModulation	TruCheckMetric	The modulation (MOD) for linear (1D) symbols.
linearQuietZone	TruCheckMetric	The quiet zone value for linear (1D) symbols.
linearSymbolContrast	TruCheckMetric	The symbol contrast (SC) value for linear (1D) symbols.
lowerLeftPattern	TruCheckMetricGradeOnly	The value for the lower left pattern in QR code.
minimumReflectance	TruCheckMetric	The minimum reflectance (minRefl) value.
Modulation	TruCheckMetricGradeOnly	The modulation (MOD) value.
modulationArray	TruCheckMetricGradeOnly	The array of modulation values.
overall		All components of formal grade defined by a grading and/or application standard.
reflectanceMargin	TruCheckMetricGradeOnly	The grade for the reflectance margin (RM).
rightClockTrack	TruCheckMetricGradeOnly	The grade for the right clock track (RCT).
rightQuietZone	TruCheckMetricGradeOnly	The grade for the right quiet zone (RQZ).
rightTransitionRatio	TruCheckMetric	The grade for the quiet zone value for linear symbols.
symbolContrast	TruCheckMetric	The grade for symbol contrast (SC) for 2D symbologies.
topClockTrack	TruCheckMetricGradeOnly	The grade for the top clock track (TCT).
topQuietZone	TruCheckMetricGradeOnly	The grade for the top quiet zone (TQZ).
topTransitionRatio	TruCheckMetric	The grade for the top transition ratio (TTR).
UII	string	The unique item identifier (UII) according to MIL-STD 130.
unusedErrorCorrection	TruCheckMetric	The grade for the unused error correction (UEC).
upperLeftPattern	TruCheckMetricGradeOnly	The grade for the upper left pattern (ULP).
upperRightPattern	TruCheckMetricGradeOnly	The grade for the upper right pattern (URP).
versionInformationBlock	TruCheckMetricGradeOnly	The version information block (VIB) value for QR Code.
verticalClockTrack	TruCheckMetricGradeOnly	The grade for the vertical clock track (VCT).

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";
    }

}
```

(i) 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, and so on. These messages can also be used manually with the Ctrl+Space key combination.
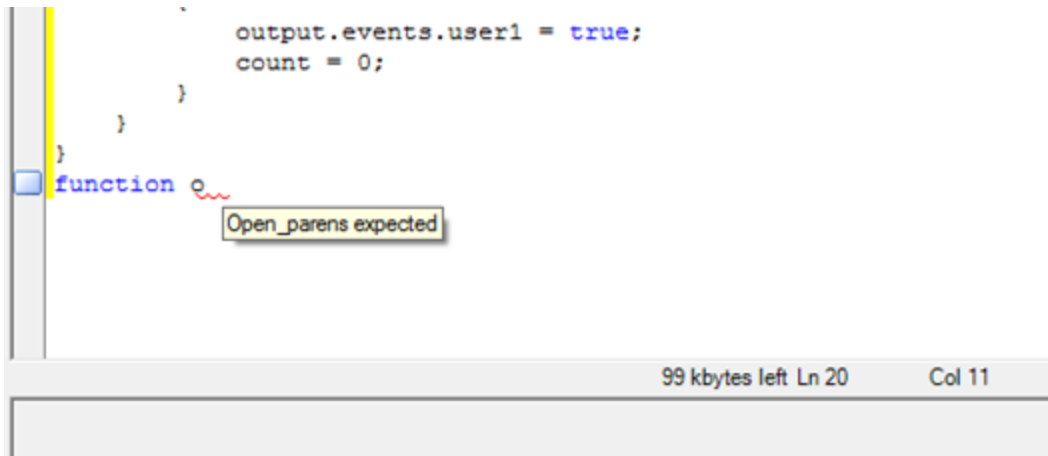
Code completion works in the following scenarios:

- complete a code fragment (Ctrl-Space)
- provide function list (Ctrl-Shift-Space)
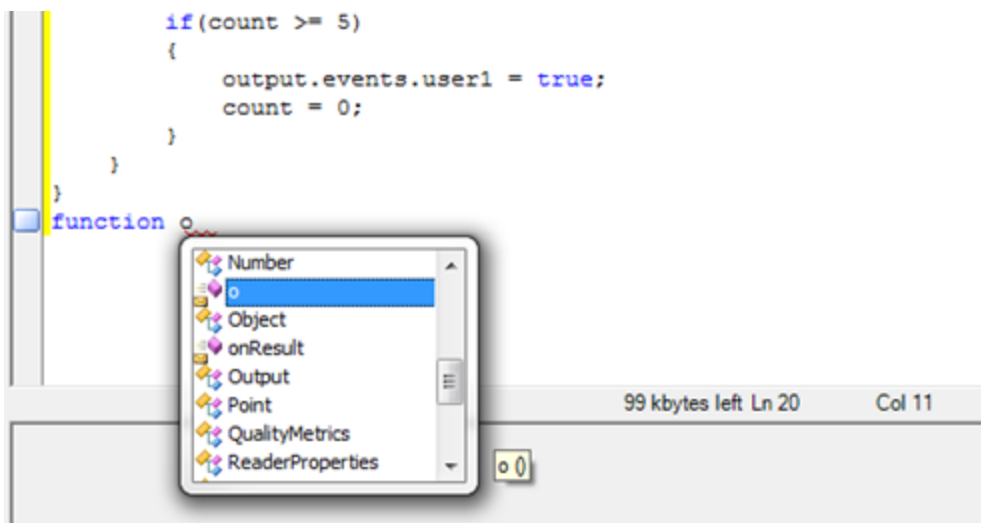
The toolbar at the top of the editor collects the following actions available within the editor:

- 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

53

"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 can 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, can be empty ("")
    - ▪ *terminator* of type string, can 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 periodically sends timer messages. Use the custom **stop** command to stop them. You can change the timer handler 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 = "000000000" + 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 = "000000000"; // 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;

}
```