



Using Ethernet/IP to communicate between DVT and ControlLogix Systems



Table of Contents

Abstract	4
Introduction	4
Types of EIP messages	4
Levels of support – Product Classes	4
Description of Support – Statement of conformance	6
Activation	8
From Intellect’s User Interface	8
Using a Intellect Terminal	8
Configuration for Explicit Messaging	9
DVT’s Configuration	9
Accessing Data Blocks Through Scripts	9
Accessing Data Blocks Using Tags	10
ControlLogix Configuration	12
Explicit Messaging Example	17
Script Examples	17
Configuration for Implicit Messaging	18
DVT’s Configuration	18
ControlLogix’s Configuration	19
Inputs	24
Outputs	24
References	27



© 2001-2008 Cognex[®] Corporation

1st Edition (January 2008). All rights reserved.

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the written permission of Cognex Corporation.

DVT[™] and Intellect[™] are trademarks of Cognex Corporation.

Microsoft Excel[®], Windows[®], Windows 2000[®], Windows ME[®], Windows XP[®] and Windows NT[®] are registered trademarks or trademarks of the Microsoft Corporation in the United States and other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.



Abstract

This document describes how to use EtherNet/IP to transfer data between DVT cameras running Intellect and a ControlLogix PLC. The purpose of this document is to describe DVT's support for EtherNet/IP and detail the configuration steps needed to use this feature. This document does NOT describe EtherNet/IP, only a very brief introduction is provided in order to define some terms that are used throughout the document. For readers that are not familiar with EtherNet/IP, the references provide some useful sources of information.

Introduction

The Ethernet/IP protocol (EIP) is an application level protocol implemented on top of Ethernet TCP/UDP/IP. It shares its object model with ControlNet and DeviceNet through the Common Industrial Protocol or CIP (also known as Control and Information Protocol). This protocol allows the transfer of data and I/O control over Ethernet.

Types of EIP messages

EIP transactions can be classified into two major categories: explicit and implicit messaging. Explicit messages are not time critical and are typically used for data collection. These messages are transferred across TCP/IP and are unscheduled. Implicit or I/O messages are considered time critical and are scheduled to be produced or consumed at a Requested Packet Interval (RPI). I/O connections are established via TCP/IP, but once the connection is initiated, the actual I/O data is sent over UDP/IP. The application context of each of these raw packets is pre-defined in the connection establishment and indicated by a simple connection ID that is added to each UDP transaction. This scheme provides for an efficient mechanism for transferring I/O data.

Levels of support – Product Classes

EtherNet/IP specifies a layered product model. There are different product classes differentiated by the level of support provided. The following figure shows the relationship between the different product classes and the characteristics of each are discussed below.



Figure 1: EtherNet/IP Product Classes - Levels of support

Level 1: Explicit Message -- Server/Target Only

Level 1 of the product model is specified for explicit messaging applications only. It acts as a target for connected and unconnected explicit messages. For example, program uploads/downloads, data collection, status monitoring, etc.

Level 2: I/O Message -- Server/Target Only

The second level of the product model adds I/O messaging support to level 1. It acts as a "responder" for both explicit and I/O messages.

Level 3: Explicit Message -- Client/Originator + Server/Target

Level 3 of the product model adds client support to level 1 explicit messaging application only. It acts as a target and an originator for messaging applications. For example, computer interface cards, HMI, and MMI devices.

Level 4: I/O Message -- Client/Originator + Server/Target

Level 4 of the product model adds I/O origination support to levels 1, 2 and 3. It acts as a target and an originator for explicit and I/O messages; for example, PLC processors, I/O scanners, logic controllers, high-end routers, etc.

Object Model

CIP capable devices have application objects that abstract the different types of data used by the device and implement the services needed for communication. Some common CIP objects are the Identity Object that holds information about the device and the Input and Output point objects that emulate the behavior of a digital I/O point.

Description of Support – Statement of conformance

DVT's current implementation provides level-2 support. In an EtherNet/IP network, DVT systems are servers with support for explicit and implicit I/O messaging. Data from the inspections can be transferred to clients via explicit messages and I/O connections that map to DVT's virtual I/O.

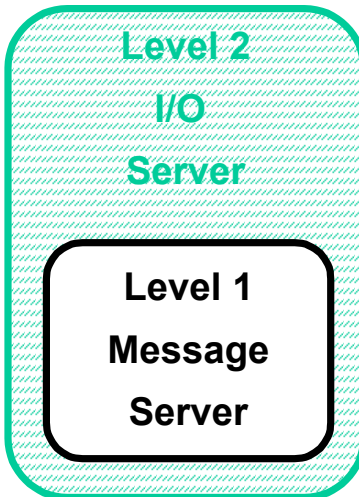


Figure 2: DVT's level of support for EtherNet/IP¹

The current object model provided by DVT systems is illustrated in the figure below. The Identity, Ethernet Link, TCP/IP and the other internal objects are required by the EtherNet/IP specification to provide the level 1 support. The different instances of the assembly object are used to exchange application data with EtherNet/IP clients. The details of how this occurs are discussed in following sections.

¹ DVT has provided level 1 support since FWK 2.1 in Beta form. With Release FWK 2.4 the support was expanded to level 2 and officially released. All versions of Intellect support level 2 functionality.

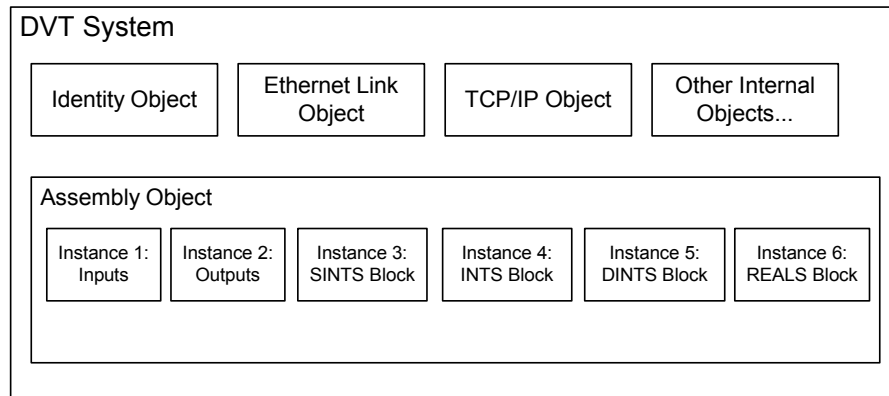


Figure 3: EtherNet/IP objects for a DVT System

Identity Object:

This object contains all the services associated with the network identity of the product when attached to the network. This object provides identification of and general information about the device.

Ethernet Link Object:

The Ethernet Link Object maintains link-specific counters and status information for a Ethernet 802.3 communications interface.

TCP/IP Object:

The TCP/IP Interface Object provides a mechanism to query and possibly configure a device's TCP/IP network interface configuration. Examples of items of interest include the device's IP Address, Network Mask, and Gateway Address.

Assembly Object: The Assembly Object binds attributes of multiple objects, which allows data to or from each object to be sent or received over a single connection. Assembly objects can be used to bind input data or output data. The terms "input" and "output" are defined from the network's point of view. An input will produce data on the network and an output will consume data from the network.

Activation

To conserve system resources in cases where the protocol is not needed, DVT systems power up without Ethernet/IP support by default. The user must specifically select his option from the user interface or alternatively through an Intellect terminal. After specifying this option, **power must be cycled to the system in order for the protocol to be supported**. See below for a description of the procedure.

From Intellect's User Interface

Connect to the system using Intellect. From the **System** menu select **Communication Settings** locate and select the **EtherNet/IP** section. On the **Properties** window set the **Enabled** property to **True**.

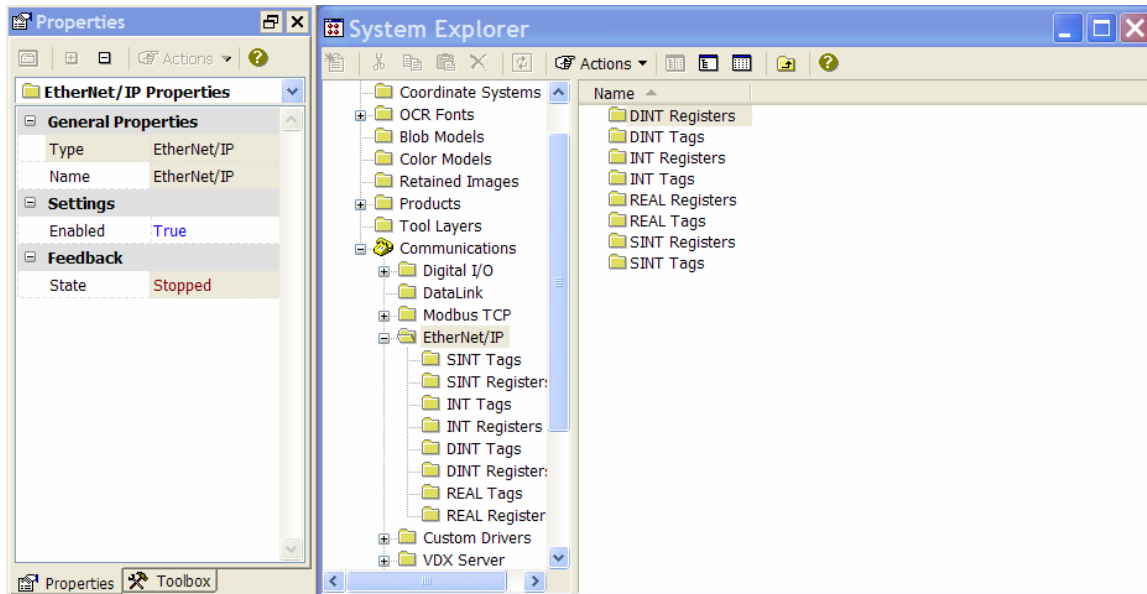


Figure 4: Activate EtherNet/IP driver

You will notice that the **State** property still reads **Stopped**. **You need to cycle power to the camera to have the EtherNet/IP driver actually start.**

Using a Intellect Terminal

Connect to the system and bring up a terminal window, then type `#Ye 1 <CR>`. You should see a `%0` at the end of the reply. Then cycle power to the system and the protocol will be activated. To de-activate the protocol, simply type `#Ye 0` at a terminal and cycle power again.

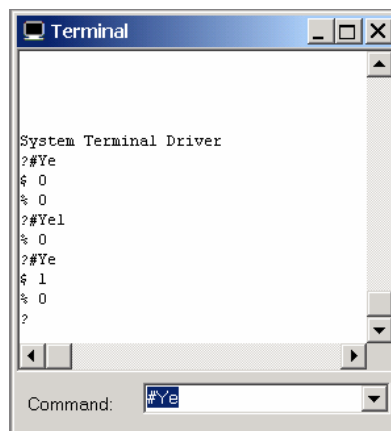


Figure 5: Activation of EtherNet/IP over a system terminal connection

When the EtherNet/IP support has been activated, the DVT system will respond to connection requests from Ethernet/IP clients (such as the ENET module on a ControlLogix PLC) over Ethernet.

Configuration for Explicit Messaging

DVT's Configuration

The user interface for DVT's implementation of EIP explicit messaging transfer involves a simple model for transferring data. Special script function calls or tags are used to read and write inspection data to and from reserved EIP registers. There are 4 blocks of data, and they are grouped according to four basic data types in the ControlLogix PLC:

Name	Description	Number Available	Range of Valid Indices	Size of Block
SINTS	8-Bit Signed Integer	256	0-255	256 bytes
INTS	16-bit Signed Integer	128	0-127	256 bytes
DINTS	32-bit Signed Integer	64	0-63	256 bytes
REALS	32-bit Floating Point	64	0-63	256 bytes

Table 1: EIP Data blocks inside DVT systems

Accessing Data Blocks Through Scripts

The following DVT script functions allow the user to read and write from individual registers in these blocks of memory. They can be found under the OEM node in the foreground script editor "Functions" tree. They can be used in both foreground and

background scripts even if you do not appear on the “Functions” tree as it is the case for background scripts.

```
AB_RegisterWriteSINT (index, value);
AB_RegisterWriteINT (index, value);
AB_RegisterWriteDINT(index, value);
AB_RegisterWriteREAL(index, value);
AB_RegisterWriteString(index, value); -Uses the SINTS block

value= AB_RegisterReadSINT (index);
value= AB_RegisterReadINT (index);
value= AB_RegisterReadDINT (index);
value= AB_RegisterReadREAL (index);
strvar =AB_RegisterReadString (index);-Uses the SINTS block
```

Note: the ranges for index and value in the above function calls are determined by the data types and size of the data blocks (see [TABLE 1](#)).

Note: each of these blocks of memory is separate. This means that there is no need to consider the size of the data types or keep track of the indexes for reading/writing as is the case with the general purpose RegisterWrite, RegisterRead family of script functions.

Accessing Data Blocks Using Tags

In Intellect we can use tags to write literal values or system, product or tool-level parameters into these data memory blocks. To do this, from the **System** menu select **Communication Settings** locate and select the **EtherNet/IP** section. In that section you will see set of tag and register subsections for each of the four supported data types. Select the tags section for the data type that could appropriate hold the value you intend to write. To create new tags double-click on the area on the right (see [FIGURE 6](#) below).

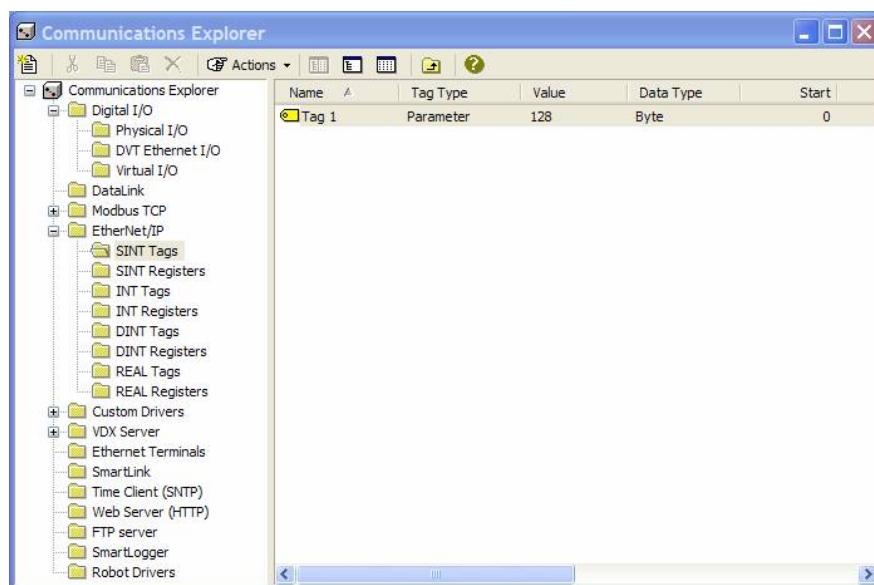


Figure 6: Using tags to write data into EIP memory blocks

Once the tag is selected go to the **Parameters** window and set the **Name**, **Source Expression** and any other parameter (for example the **Start** address).

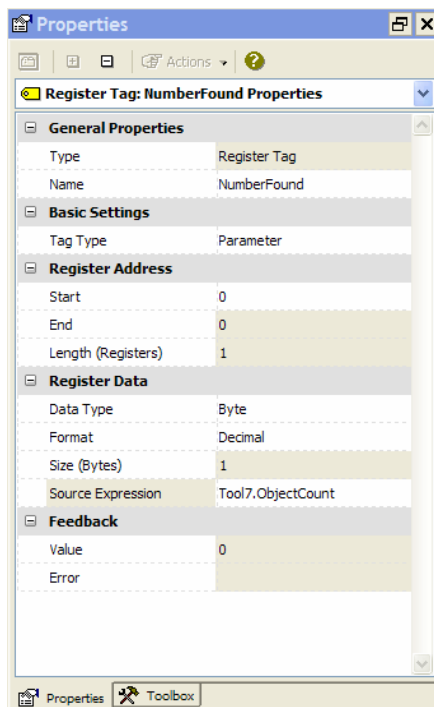


Figure 7: Properties for EtherNet/IP tag

It is important to understand that when the mentioned script functions are executed or when tags are updated (in the case of product tags, they are updated at the end of each inspection) data is updated **only** in registers within the DVT system. Data transfers with the ControlLogix only occur when initiated by a MSG instruction from the PLC (see next section). This happens independently and asynchronously from the inspections.

ControlLogix Configuration

On the ControlLogix PLC, a MSG instruction is used to read and write entire blocks of data from the DVT systems. To avoid confusion, and simplify programming it is suggested that two different sets of array tags are created to manage communications with the DVT system. There should be a *destination* set of tags and a *source* set of tags for each data type. Both the destination and source tags are array tags. The following table shows a possible configuration:

Destination Tags	Source Tags
dest_SINTS[256]	source_SINTS[256]
dest_INTS[128]	source_INTS[128]
dest_DINTS[64]	source_DINTS[64]
dest_REALS[64]	source_REALS[64]

Table 2: Suggested tags in the ControlLogix PLC

The MSG instruction can be used to “read in” blocks of data from the DVT system and update one of the destination PLC tags. The MSG instruction can also be used to write data blocks to the DVT system using the source PLC tags as the source of the data. Notice that all four memory blocks in the camera are shared for reading and writing operations. In other words, there are only four memory blocks and not eight (four for reading and four for writing).

Before Intellect version 1.3.2 a MSG instruction always exchanged an entire data block with the DVT system. This was the case even when the script functions or tags only accessed part of the data block. In every transaction the entire 256 bytes of data were transferred. This is why the number of elements in the source and destination arrays always had to represent a total of 256 bytes.

Because there same block (for each data type) is used for reading and writing, users had to be very careful in not to overwrite information that was written on the camera side and was meant to be read by the PLC by the information written by the PLC to the camera. This created some synchronization challenges.

Starting with Intellect 1.3.2 and in later versions, the entire block for each data type is still transferred when the PLC is reading from the camera. However, when the PLC is writing to

the camera, the number of elements transferred can be less than the equivalent to 256 bytes. This allows the PLC to write to the first part of the data blocks on the camera (it always has to be the first part since there is no way to specify an offset through the MSG CIP Generic message) and use the rest of the block to read data written by the camera.

The following figure shows the configuration needed for a MSG instruction that will read the block of SINT registers from the camera:

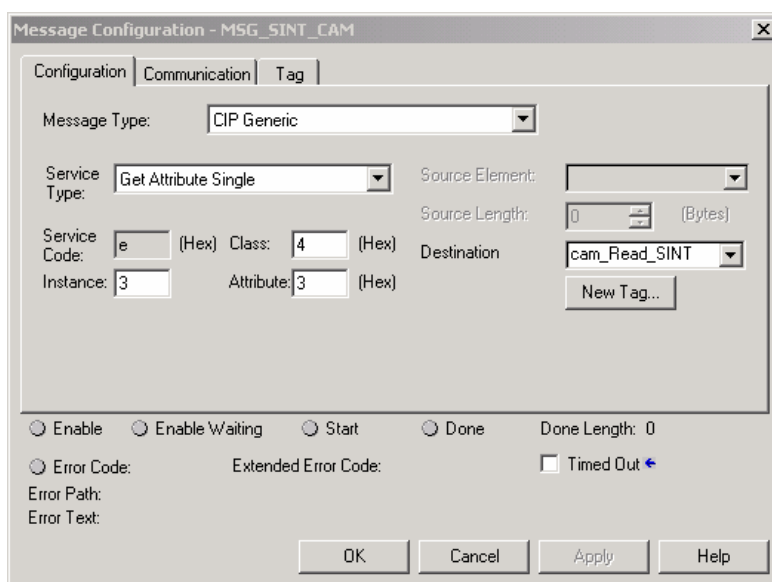


Figure 8: MSG instruction configuration

Pay close attention to the **Message Type**, CIP Generic. This selection allows the ControlLogix PLC's to exchange explicit CIP messages with any CIP capable device. In the current DVT EIP implementation, application data is handled by the Assembly Object. The user can regard this object as block of data that has read/write access from the DVT inspections and from the ControlLogix PLC via Ethernet/IP (or CIP) messages. For this purpose, DVT supports four instances of the object corresponding to each of the blocks of data mentioned earlier: SINTS, INTS, DINTS and REALS instances.

In order to configure the MSG instruction correctly we need to specify the different parameters in the configuration tab appropriately.

Service Type indicates what type of operation is intended on the object. In our case, only two service codes will be needed:



- **Get Attribute Single (code 0x0E):** This service reads a specified block of data from the DVT system and places the data in the source tag under the **Destination** parameter.
- **Set Attribute Single (code 0x10):** This service writes from a tag in the ControlLogix PLC to a block of data in the DVT system. The **Source Element** parameter indicates what tag to use for the source of the data. When using this service is necessary to specify the **Source Length** parameter as the size in bytes of the data to be transferred. For Intellect version older than 1.3.2, this always needs to be set to 256 bytes. Starting with Intellect 1.3.2, you have the ability to use only a subset of the register block on the DVT camera (always starting at register 0). In this case, the length of the PLC tag should match the source length.

Class indicates the type of object that the CIP MSG needs to access. The Ethernet/IP's CIP protocol specifies many different objects identified with a unique code. DVT currently implements the Assembly Object; its CIP code is (0x04). This value should be used in all MSG instruction configurations.

Instance indicates the object instance that the CIP MSG will be addressing. In a given device there may be several instances of the same type of object. In the current implementation a DVT system has 6 instances of the assembly object. Four of these correspond to the 4 blocks of data mentioned earlier. The following table shows the instance numbers for the Assembly Object.

Instance Name	Instance Number	Message Type
Input	1	Used in Implicit Messages
Output	2	
SINTS	3	Used in read/write MSG instructions (explicit messages)
INTS	4	
DINTS	5	
REALS	6	

Table 3: Instances of DVT Assembly Object

The **Attribute** indicates what piece of information, from a particular instance of an object is accessed by the CIP message. Each assembly object has several attributes or pieces of data, like the revision number or its own **Instance**. Each attribute is identified by a code. We are interested in the data attribute of the assembly object. This attribute is identified with code (0x03).

The following table illustrates possible MSG instruction configurations that allow explicit data exchange with a DVT system:

Service Type	Service Code	Class	Instance	Attribute	Source Element (example)	Source Length	Destination (example)	Description
Get Attribute Single	e	4	3	3	N/A	N/A	dest_SINTS	Read the SINTS data block from DVT system into dest_SINTS array
Get Attribute Single	e	4	4	3	N/A	N/A	dest_INTS	Read the INTS data block from DVT system into dest_INTS array
Get Attribute Single	e	4	5	3	N/A	N/A	dest_DINTS	Read the DINTS data block from DVT system into dest_DINTS array
Get Attribute Single	e	4	6	3	N/A	N/A	dest_REALS	Read the REALS data block from DVT system into dest_REALS array
Set Attribute Single	10	4	3	3	source_SINTS	256	N/A	Write data in source_SINTS array into the SINTS data block in DVT. Could be a subset of the 256-bytes long block.
Set Attribute Single	10	4	4	3	source_INTS	256	N/A	Write data in source_INTS array into the INTS data block in DVT. Could be a subset of the 256-bytes long block.
Set Attribute Single	10	4	5	3	source_DINTS	256	N/A	Write data in source_DINTS array into the DINTS data block in DVT. Could be a subset of the 256-bytes long block.
Set Attribute Single	10	4	6	3	source_REALS	256	N/A	Write data in source_REALS array into the REALS data block in DVT. Could be a subset of the 256-bytes long block.

Table 4: MSG instruction configuration parameters

On the **Communications** tab, the user needs to specify the **Path** to the DVT system. The path is a succession of comma-separated values that indicates the route for the CIP MSG starting at the Ethernet module on the ControlLogix PLC and ending at the target device. The following figure indicates the meaning of each of its components.

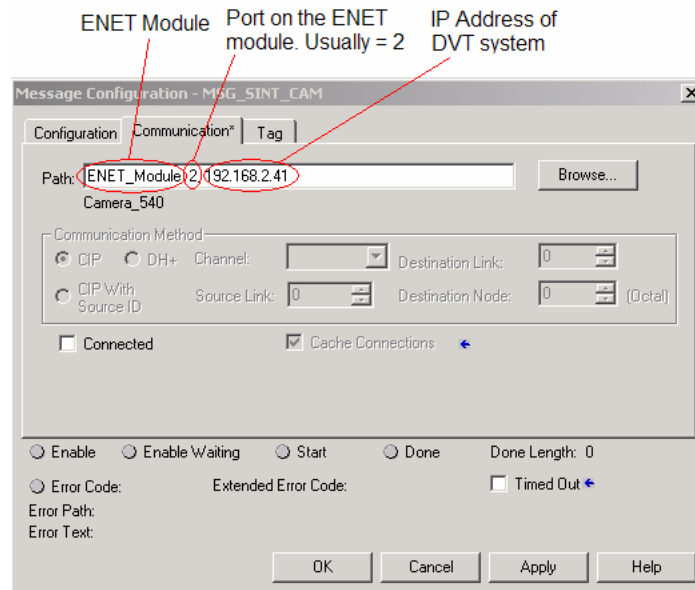


Figure 9: MSG instruction configuration - Communication tab

The path above can also be expressed as 1,2,2,192.168.2.41 in our particular case. Where the 1 represents the slot number of the processor in the rack and the first 2 (second element) represents the slot number of the ENET module in the rack. If you have added the camera as an I/O device (see next section) you can simply browse to that module and select it from the tree.

It is recommended to leave the “Connected” checkbox unchecked; especially for infrequent exchanges of information.

The following diagram shows the direction of data flow between the two systems. The Data blocks in the DVT system are accessed both by the PLC and by the inspections through the script. The source and destination tags are shown on the ControlLogix PLC.

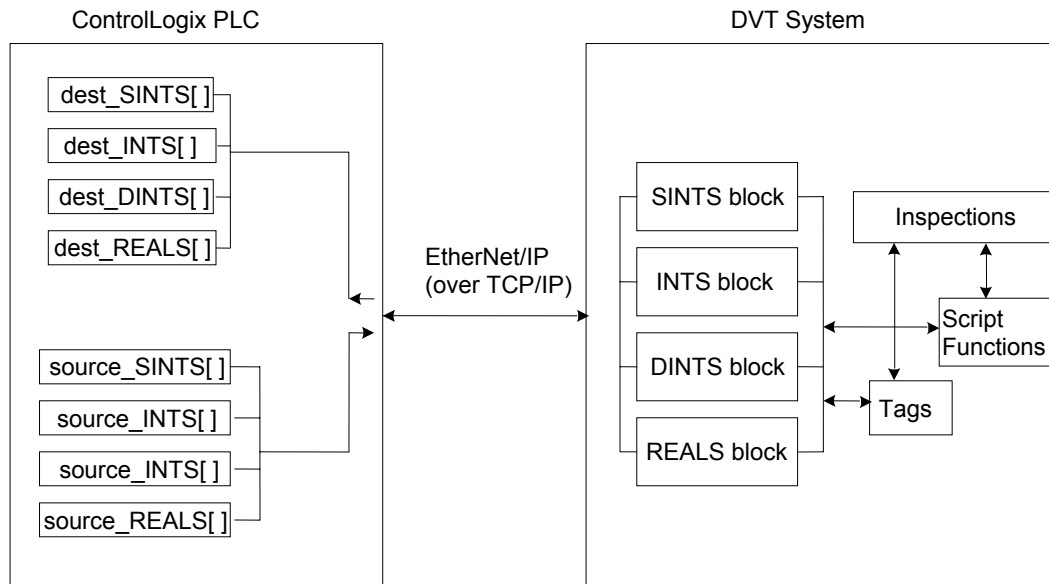


Figure 10: Schematic of data transfer for explicit messages

Explicit Messaging Example

Script Examples

The following example script snippet writes values from the inspections into the registers for subsequent transfer to the PLC

```
// Places the number of blobs at index 0 of the INTS block.
AB_RegisterWriteINT(0, Counter.ObjectCount);

//Each char of the string is put in the SINTS block (in this case
//starting @ index 10)and adding a Null termination at the end.
AB_RegisterWriteString (10, Reader.String);

//Or you can write a literal, static value.
AB_RegisterWriteString (10, "test");
```

The following example script snippet reads values from the DVT data blocks into local variables for use in the script. The DVT data blocks is assumed be populated by the PLC.



```
//Read in a value from the INTS tag in the PLC (@index 10) and place //it
in the local variable i.

int i;
i = AB_RegisterReadINT(10);

// Read in characters from SINTS tag in the PLC starting at
// index 20 until a null char is found (i.e a SINT with a value of
// zero.) Place the Null terminated string in local variable s.

String s;
s = AB_RegisterReadString(20);
```

Sequence of Events

The correct sequence of events that need to happen is the following:

1. With inspections running, the script containing the AB_Register*() functions has to be executed at least once. If tags are being used, they will be automatically updated at the end of each inspection. This places the most recent inspection data in the DVT data blocks (data from the blocks can also be loaded into local script variables.) Note that if the system is triggered w/o inspections running, the AB_RegisterWrite() functions will not execute! This is because they are treated as an output of the system.
2. A MSG instruction from a ControlLogix PLC is sent to the DVT system. This instruction will then exchange data with the DVT data blocks that contain the latest information updated from the last inspection.

Typical Operation:

1. PLC triggers the DVT system with a Digital Output.
2. PLC monitors outputs from the DVT system to know when the inspection has taken place (use the inspection toggle bit or the busy bit) and if the result is PASS.
3. A MSG instruction is then enabled to exchange data between the two systems.

Note: When the PLC writes a block into the DVT system it will overwrite as many values as the Source Length parameter specifies for that particular data type.

Configuration for Implicit Messaging

DVT's Configuration

The user interface for DVT's implementation of EIP implicit messaging is extremely simple. There are two instances of the assembly object (1 and 2) that are automatically

mapped to DVT's virtual inputs and outputs. After activation of the EIP protocol, the user needs no further configuration steps.

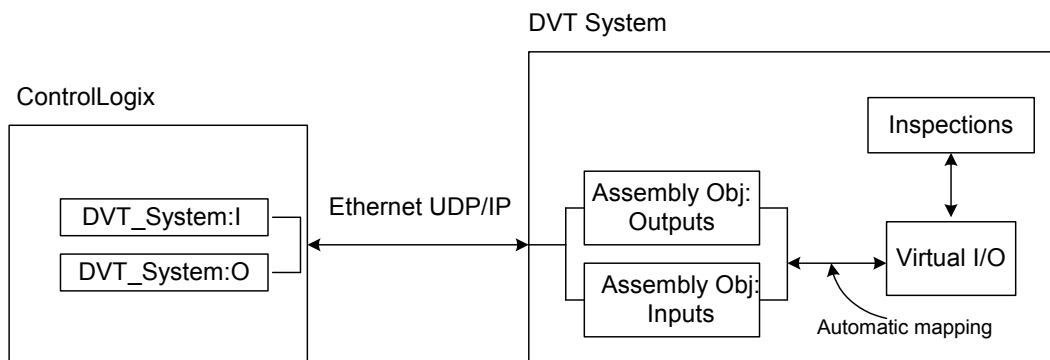


Figure 11: Schematic of data transfer for implicit messaging (I/O connection)

As the values of the virtual outputs change during an inspection, they are reflected in the assembly object instance associated with the outputs. If there is an EIP implicit messaging connection to this object (see next section) then these values are updated on the client at the Requested Packet Interval (RPI.) In the same manner, if the client updates values across an EIP implicit messaging connection to the assembly object instance associated with the inputs, then these changes are immediately reflected in the virtual inputs and the DVT system reacts accordingly. These capabilities represent Ethernet-based I/O through EtherNet/IP.

ControlLogix's Configuration

Most of the information in this section is based on public documentation from Rockwell Automation (see [REFERENCES](#) at the end of this document).

RSLogix 5000 version 8 supports Generic Ethernet Devices. To setup EtherNet/IP implicit messages transfer between a DVT system and a ControlLogix processor, you will need to configure the DVT system as a Generic Ethernet Module in your ControlLogix I/O tree. You can only "schedule" implicit transfers to a 1756-ENBT or a 1756-ENET/B (FRN 2.06 or greater). We will assume here that you already such a module configured as an I/O device.



Right Click here and select
New Module...

Figure 12: Adding the DVT system as an I/O device in a ControlLogix I/O tree

Right click on the 1756 module and select “New Module...” When the following dialog appears, select the “Generic Ethernet Module” option. This option is included in the ControlLogix system to allow communications to devices from any vendor that support EtherNet/IP but that does not have a module profile.

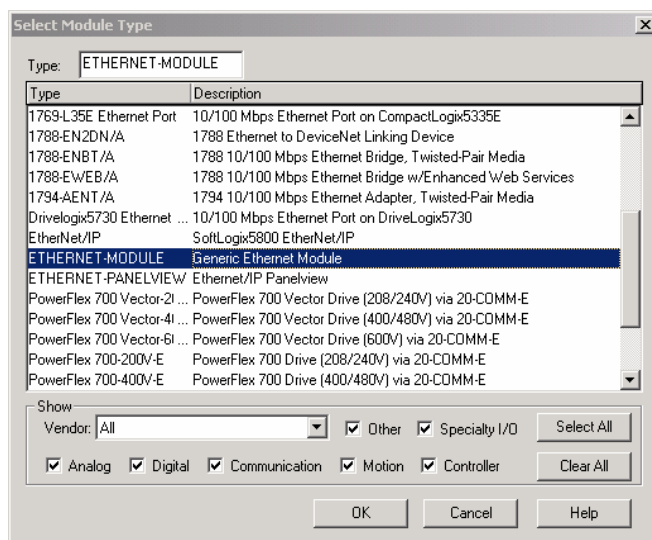


Figure 13: Adding the DVT camera as a Generic Ethernet Module

After making this selection, the configuration dialog is shown. Below is a figure of the configuration window for the 1756 Generic Module Profile”:

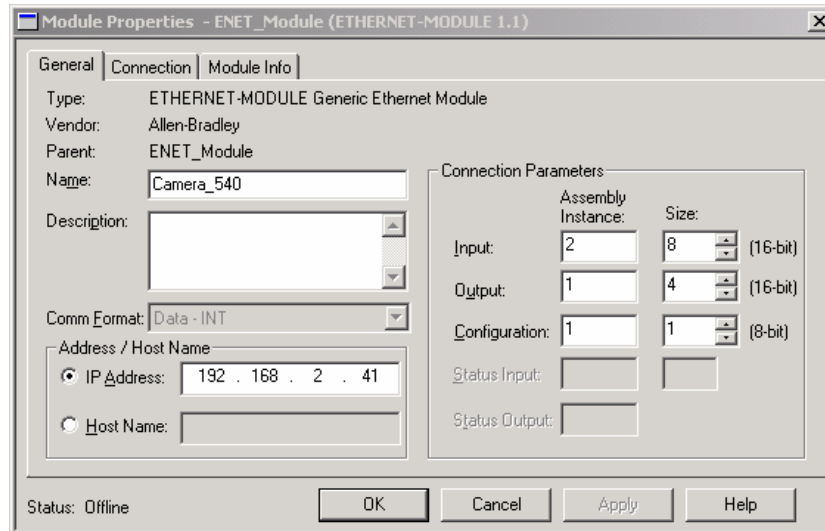


Figure 14: Configuration of the DVT camera as a Generic Ethernet Module

Let us look at each of the fields that need to be configured:

Name: Name given to the generic EIP device. It is suggested to use the DVT's system name to maintain consistency. Tags will be created in RSLogix 5000 based on this name.

IP Address: IP Address that the DVT system will have.

Comm Format: Set to Data-INT. This will determine how the I/O bits are arranged. It is more convenient in this case to set it Data-INT. Note that this cannot be changed once the module is configured the first time.

Host Name: Do not use- this is only used when there is a DNS server on your network (Domain Name Server) and this device support DNS.

Input Assembly Instance: This field should contain the instance number of the Assembly object in the server that acts as the output (input to the client). For DVT systems this is instance number 2, which is mapped to the virtual outputs.

Input Size: This field specifies the size of the output assembly instance in the DVT system. The size for instance 2 of the DVT's assembly object is 128 bits, one bit per virtual output point. Thus the size should be set to 8 (16-Bit) words, since we selected the data type to be INT. The Data type selection determines how the corresponding tags are formed in the ControlLogix system.

Output Assembly Instance: This field should contain the instance number of the Assembly object in the server that acts as the input (output to the client or EtherNet/Ip master, in this case the PLC). For DVT systems this is instance number 1, which is mapped to the virtual inputs.

Output Size: This field specifies the size of the input assembly instance in the DVT system. The size for instance 1 of the DVT's assembly object is 64 bits, one bit per virtual input point. Thus the size should be set to 4 (16-Bit), since we selected the data type to be INT. The Data type selection determines how the corresponding tags are formed in the ControlLogix system.

Configuration Instance: DVT systems do not support the configuration instance, but we must supply a value for the Generic Profile.

Configuration Size: DVT systems do not support the configuration instance, but we must supply a value for the Generic Profile.

The figure below shows the next tab in the configuration window for the 1756 Generic Module Profile.

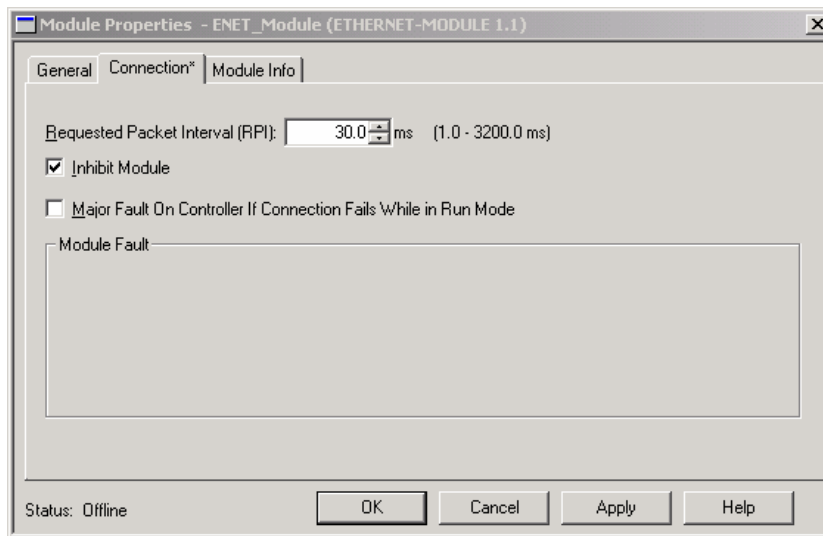


Figure 15: Generic Ethernet Module (camera) configuration - Connection tab

RPI: This field specifies the Requested Packet Interval which is the period of production/consumption across the implicit messaging connection. The units for this value are given in ms. When several connections are being used, it is recommended not to go below 10 ms.

Inhibit Module Checkbox: Checking this box will ensure that the connection is not attempted immediately after downloading the project to the processor. Once online, the connection can be initiated by un-checking this box.

Major Fault On Failed Connection Checkbox: This option will cause the processor to throw a major fault when the connection fails.

The last tab labeled ‘Module Info’ provides information about the remote device once we are online. There is no need to configure anything here. After adding the module the ControlLogix I/O tree should look like the one shown in the figure below (in this case, we have three DVT cameras configured).

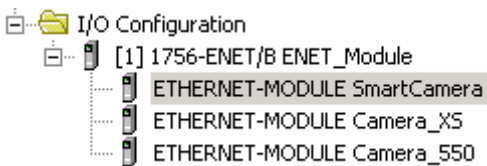


Figure 16: DVT systems added as Generic Ethernet Modules in a ControlLogix I/O tree

After adding the Generic Module profile to the I/O Tree, tags will be created in the controller based on the name that you have given this device, as shown below:

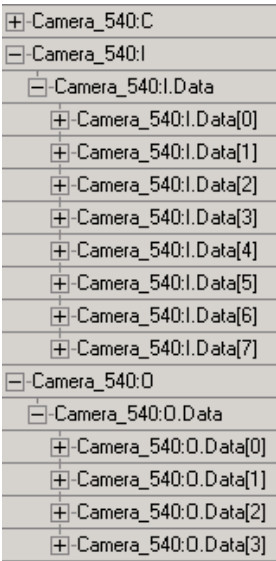


Figure 17: Controller-level tags automatically added for each Generic Module created

Camera_540: C Configuration Data

This data is not used.

Inputs**Camera_540: I Scheduled Input**

This data is sent FROM the DVT system to the EIP client.

Tag Name	Bit Position															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DVT:I:Data[0]	Rrsvd	Rrsvd	Rrsvd	Wrong Code	Inspection Toggle	Inspecting	Acquiring	Busy	Trigger Mode	Run Mode	Resource Conflict	Fail	Pass	Product Select Fail	Product Select Pass	Trigger Ack
DVT:I:Data[1]	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Strobe 3	Strobe 2	Strobe 1
DVT:I:Data[2]	User 16	User 15	User 14	User 13	User 12	User 11	User 10	User 9	User 8	User 7	User 6	User 5	User 4	User 3	User 2	User 1
DVT:I:Data[3]	Script 16	Script 15	Script 14	Script 13	Script 12	Script 11	Script 10	Script 9	Script 8	Script 7	Script 6	Script 5	Script 4	Script 3	Script 2	Script 1
DVT:I:Data[4]	Total Inspection Count															
DVT:I:Data[5]	Current Inspection Product Digital ID															
DVT:I:Data[6]	Reserved															
DVT:I:Data[7]	Reserved															

Table 5: Mapping of DVT virtual outputs (inputs to the PLC) to ControlLogix tags

Outputs**Camera_540: O Scheduled Output Data**

This Data Sent TO the DVT system from the EIP client.

Tag Name	Bit Position															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DVT:O:Data[0]	Rrsvd	Rrsvd	Rrsvd	Rrsvd	Digital Relearn 4	Digital Relearn 3	Digital Relearn 2	Digital Relearn 1	External Trigger Mode	Disable Run Mode	Quadrature Input	Event Counter	Results Ack	Rrsvd	Product Select	Trigger
DVT:O:Data[1]	Rrsvd	Product ID Bit 14	Product ID Bit 13	Product ID Bit 12	Product ID Bit 11	Product ID Bit 10	Product ID Bit 9	Product ID Bit 8	Product ID Bit 7	Product ID Bit 6	Product ID Bit 5	Product ID Bit 4	Product ID Bit 3	Product ID Bit 2	Product ID Bit 1	Product ID Bit 0
DVT:O:Data[2]	User 16	User 15	User 14	User 13	User 12	User 11	User 10	User 9	User 8	User 7	User 6	User 5	User 4	User 3	User 2	User 1
DVT:O:Data[3]	User 32	User 31	User 30	User 29	User 28	User 27	User 26	User 25	User 24	User 23	User 22	User 21	User 20	User 19	User 18	User 17

Table 6: Mapping of DVT virtual inputs (outputs to the PLC) to ControlLogix tags

Example

For I/O messaging, we have added the DVT system as an input/output device in our ControlLogix project. After this project is downloaded to the processor, the I/O connection will be established, as is the case with other remote I/O devices (unless the “Inhibit” checkbox was set offline). After a successful connection establishment, cyclic data transfers at the requested RPI will be initiated, even if the processor is in run mode.

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

1. Working Offline, perform the ControlLogix configuration steps described above.
2. Download the project to the ControlLogix processor.
3. Upon completion of the download process, the I/O device that corresponds to the DVT system should show no errors. This signifies that the I/O connection has been completed successfully.
4. To verify correct 2-way transfer of I/O data, connect to the DVT system with the Intellect UI, place the system in external trigger mode with play mode on, create any product and enable inspections. Back in RSLogix, go to controller tags and note the status of bit 11 of input word 0 (output 11 to the DVT system, which is mapped to Inspection Toggle signal). Then, change the state of bit 0 of output word 0 (input 0 to the DVT system, which is mapped to the Trigger signal) from 0 to 1 as shown below.

- Camera_540:0	{ ... }
- Camera_540:0.Data	{ ... }
- Camera_540:0.Data[0]	1
- Camera_540:0.Data[0].0	1
- Camera_540:0.Data[0].1	0
- Camera_540:0.Data[0].2	0
- Camera_540:0.Data[0].3	0
- Camera_540:0.Data[0].4	0
- Camera_540:0.Data[0].5	0
- Camera_540:0.Data[0].6	0
- Camera_540:0.Data[0].7	0
- Camera_540:0.Data[0].8	0
- Camera_540:0.Data[0].9	0
- Camera_540:0.Data[0].10	0
- Camera_540:0.Data[0].11	0
- Camera_540:0.Data[0].12	0
- Camera_540:0.Data[0].13	0
- Camera_540:0.Data[0].14	0
- Camera_540:0.Data[0].15	0
+ Camera_540:0.Data[1]	0
+ Camera_540:0.Data[2]	0
+ Camera_540:0.Data[3]	0

Figure 18: Triggering a DVT system over EtherNet/IP

Since bit **Camera_540:O.Data[0].0** is mapped to the trigger virtual input, this should cause an inspection to be triggered in the DVT system. After this happens, bit **Camera_540:I.Data[0].11** of the outputs (mapped to the inspection toggle virtual output) should change state; that is, have the opposite value to what you observed before triggering the DVT camera.

[-] Camera_540:I	{...}
[-] Camera_540:I.Data	{...}
[-] Camera_540:I.Data[0]	2248
[-] Camera_540:I.Data[0].0	0
[-] Camera_540:I.Data[0].1	0
[-] Camera_540:I.Data[0].2	0
[-] Camera_540:I.Data[0].3	1
[-] Camera_540:I.Data[0].4	0
[-] Camera_540:I.Data[0].5	0
[-] Camera_540:I.Data[0].6	1
[-] Camera_540:I.Data[0].7	1
[-] Camera_540:I.Data[0].8	0
[-] Camera_540:I.Data[0].9	0
[-] Camera_540:I.Data[0].10	0
[-] Camera_540:I.Data[0].11	1
[-] Camera_540:I.Data[0].12	0
[-] Camera_540:I.Data[0].13	0
[-] Camera_540:I.Data[0].14	0
[-] Camera_540:I.Data[0].15	0
[+] Camera_540:I.Data[1]	0
[+] Camera_540:I.Data[2]	0
[+] Camera_540:I.Data[3]	0
[+] Camera_540:I.Data[4]	0
[+] Camera_540:I.Data[5]	0
[+] Camera_540:I.Data[6]	0
[+] Camera_540:I.Data[7]	0

Figure 19: Observing the Inspection Toggle output over EtherNet/IP



References

- The primary source for information on EtherNet/IP is the Specification document. The specification can be downloaded from <http://www.odva.org>.
- Secondary information sources are available from the [EtherNet/IP library](#) at the ODVA site:
 - “Recommended IP Addressing Methods for EtherNet/IP Devices”
 - “Network Infrastructure for EtherNet/IP”: contains very important and useful documentation on recommended switches, network architectures, etc. for EtherNet/IP devices.
 - “Utilization of Modern Switching Technology in EtherNet/IP Networks”
- Rockwell Automation Knowledge Base publication ID 13968 “Using Ethernet PanelViews with ControlLogix processors.”