



# **Using Ethernet/IP to Communicate between DVT Cameras and Allen Bradley SLC PLCs**

Table of Contents

Abstract ..... 4

Introduction ..... 4

SLC5/05 Firmware Versions ..... 4

Activation ..... 5

    From Intellect’s User Interface..... 5

    Using a Intellect Terminal ..... 5

Configuration for EIP/PCCC Explicit Messaging -DVT ..... 6

    DVT’s Configuration ..... 6

        Accessing Data Blocks Through Scripts ..... 7

        Accessing Data Blocks Using Tags..... 8

SLC5/05 Configuration ..... 9

Explicit Messaging Examples ..... 12

    Example 1 ..... 12

    Example 2 ..... 13

    Example 3 ..... 15

        Sequence of Events For Examples 1 and 2 ..... 16

        Sequence of Events For Example 3 ..... 16

References ..... 17



© 2001-2008 Cognex<sup>®</sup> Corporation

1<sup>st</sup> 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<sup>™</sup> and Intellect<sup>™</sup> are trademarks of Cognex Corporation.

Microsoft Excel<sup>®</sup>, Windows<sup>®</sup>, Windows 2000<sup>®</sup>, Windows ME<sup>®</sup>, Windows XP<sup>®</sup> and Windows NT<sup>®</sup> 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 SLC 5/05 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 SLC5/05 processor communicates using PCCC (Programmable Controller Communication Commands) application-level commands. To allow peer-to-peer communications between different families of Allen-Bradley processors, the PLC5E and SLC5/05 added support for EtherNet/IP with embedded PCCC commands ("EIP/PCCC.") While ControlLogix systems fully support EIP with generic implicit and explicit messaging, the SLC5/05 only supports EIP explicit messages directed to the vendor specific PCCC object (class 0x67.)

All versions of the Intellect software for DVT cameras support EIP/PCCC to allow direct communication with the SLC5/05 processor over EtherNet/IP.

## SLC5/05 Firmware Versions

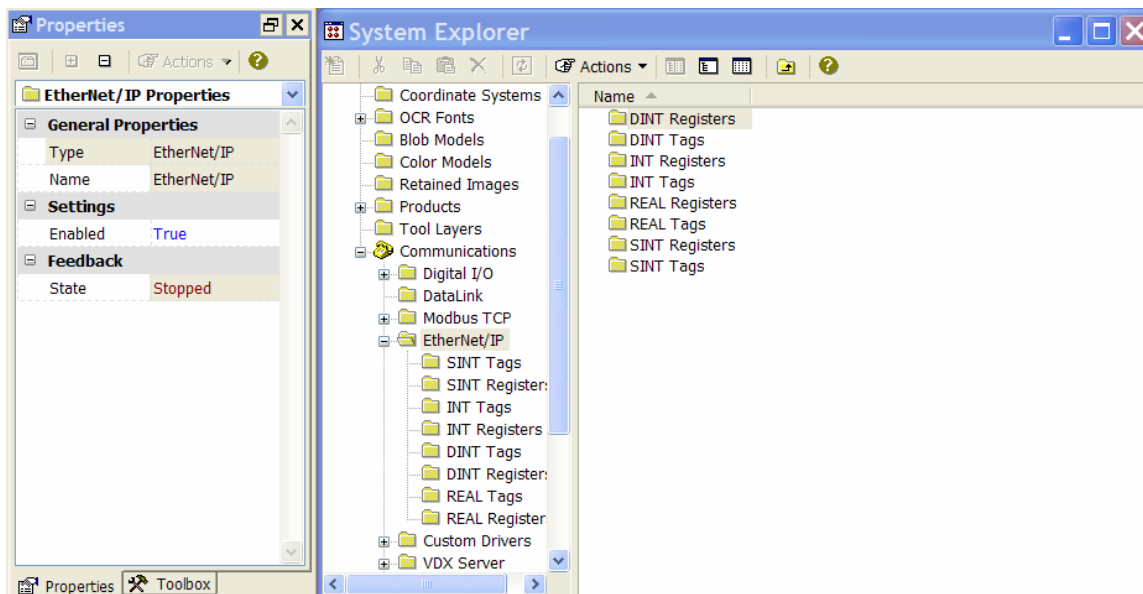
According to Allen-Bradley's documentation, The **SLC5/05** was upgraded with EIP support in May 1999, with **series A** firmware revision **OS501, FRN5**. If the SLC5/05 is an earlier revision it can be Flash updated with new firmware. All **series B and C** SLC5/05's support EIP.

## 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**.

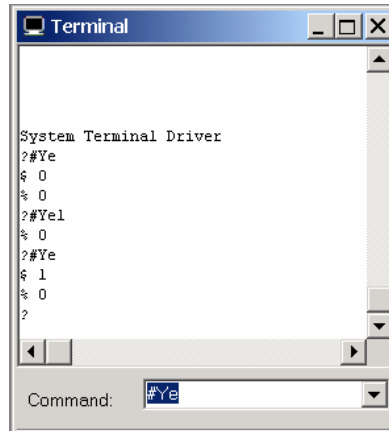


**Figure 1: Activate EtherNet/IP driver**

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 2: 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 embedded Ethernet module on an SLC 5/05 PLC) over Ethernet.

## Configuration for EIP/PCCC Explicit Messaging -DVT

### *DVT's Configuration*

The user interface for DVT's EIP/PCCC explicit messaging is the same as for the generic EIP support. It 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 EIP data, as follows:

| 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**

Note: the DINTS data block is not actually used for EIP/PCCC messages; the other three blocks are mapped to SLC-style data files.

The SLC5/05 uses files of different types to store data. Each data element in the file is accessed via a file number and an offset. For example, to access the third element on an integer file numbered 7, one would write “N7:3”.

To implement EIP/PCCC feature, DVT has assigned 4 different file numbers to EIP data blocks to allow use of the above syntax to refer to each of their elements. One can think of DVT systems as having 4 SLC-style files with the following characteristics:

| Name<br>(File Letter)   | File<br>Number | Associated EIP<br>Data Block | Number of<br>elements | Size of Block<br>(bytes) |
|-------------------------|----------------|------------------------------|-----------------------|--------------------------|
| DVT Integer File<br>(N) | 7              | INTS                         | 128                   | 256 bytes                |
| DVT Float File<br>(F)   | 8              | REALS                        | 64                    | 256 bytes                |
| DVT ASCII File<br>(A)   | 9              | SINTS                        | 256                   | 256 bytes                |
| DVT String File<br>(S)  | 10             | SINTS                        | 256                   | 256 bytes                |

**Table 2: DVT’S SLC-style data files for use in EIP/PCCC messages**

So, for instance, if we want to transfer a 16-bit integer value to the PLC, we would write the value into the INTS data block (let’s say at address or index 0) from the camera and access it from the PLC by reading it from offset 0 in SLC-style file N7 in the camera (address N7:0).

### Accessing Data Blocks Through Scripts

The following DVT script functions allow the user to read and write from individual registers (register, address and index are used interchangeably here) 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. In the following discussion we only reference the three data blocks that are used for EIP/PCCC messages.

```

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

value= AB_RegisterReadSINT (index);

```

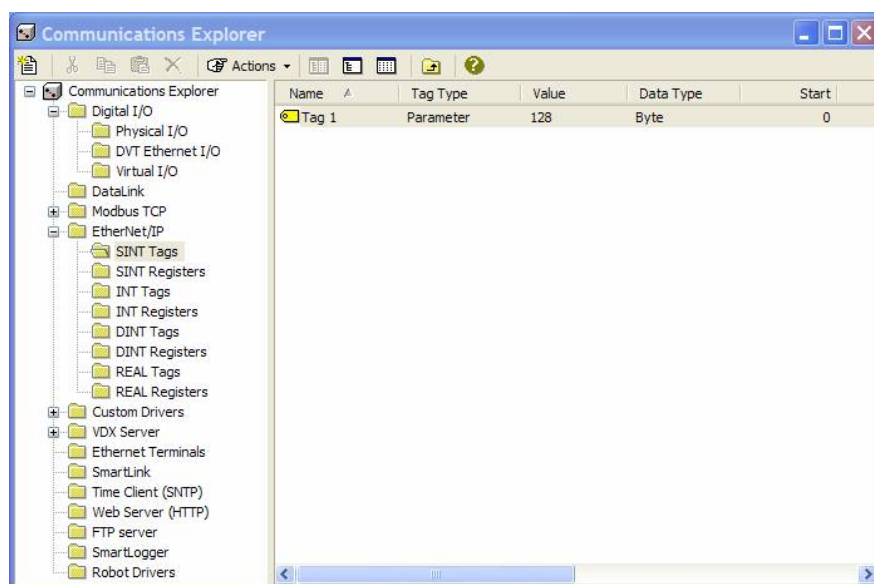
```
value= AB_RegisterReadINT (index);
value= AB_RegisterReadREAL (index);
strvar =AB_RegisterReadString (index);-Uses the SINTS block
```

**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. The notable exception is the case of strings which will take one element per character plus an additional element for the null terminator.

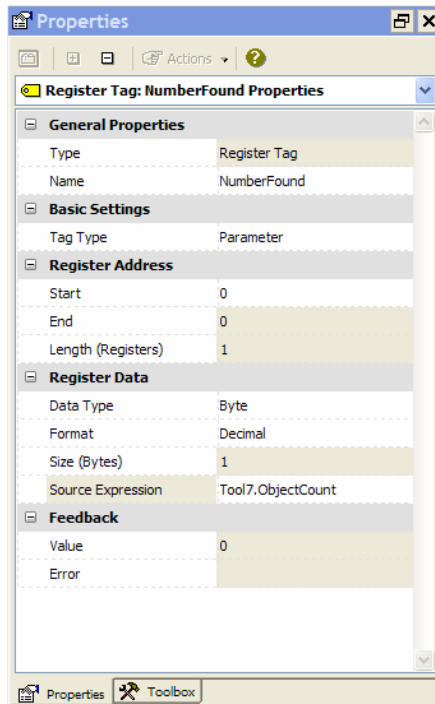
### Accessing Data Blocks Using Tags

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 3](#) below).



**Figure 3: 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 4: 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 SLC only occur when initiated by a MSG instruction from the PLC (see next section). This happens independently and asynchronously from the inspections.

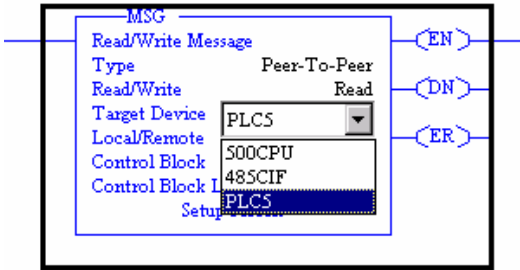
### *SLC5/05 Configuration*

In the SLC5/05, use a MSG instruction to read and write data from/to DVT systems. File numbers and types in the DVT system are fixed and must be specified in the MSG configuration as described above. In the SLC5/05, the user can configure most files and the numbers used are not fixed. For the purpose of this example let us assume that the following files exist in the PLC and will be used for communication with a DVT system.

| File | Data Type                   | Number of Elements |
|------|-----------------------------|--------------------|
| N7   | 16-bit Integer              | 8                  |
| F8   | 32-bit Floating-point value | 102                |
| A9   | ASCII                       | 23                 |
| ST15 | String                      | 5                  |

**Table 3: Sample Configuration of Data Files in an SLC5/05**

The first step in the MSG configuration is to designate the type of target device to a PLC5. This tells the SLC to use Typed Read/Write commands. These commands are embedded inside EIP explicit messages in EIP/PCCC.



**Figure 5: Selection of PLC5 as Target Device**

Next, bring up the MSG setup screen and select **Channel 1** for Ethernet communications. Notice this adds a new field called **MultiHop** on the general tab. Also notice the **Message Timeout** field; it specifies the number of seconds to wait for a response after the MSG is sent before timing out with an error. This value depends on the application.

**Figure 6: MSG configuration - channel selection**

Since the SLC supports other protocols over Ethernet TCP/IP there are certain configuration parameters that must be specified in the MSG instruction to ensure that the SLC uses EIP/PCCC. To force the use of EIP/PCCC, choose **MultiHop=Yes** in the MSG setup. This brings up an additional tab in the MSG configuration dialog labeled MultiHop. The only thing to edit in this tab is the IP address on the first line. This IP address **MUST** correspond to the IP of the target DVT system. Do not write leading zeroes for octets that take only two digits (e.g. write 192.168.0.86 and NOT 192.168.0.086).

| From Device            | From Port | To Address Type          | To Address   |
|------------------------|-----------|--------------------------|--------------|
| This SLC500            | 1         | 1756-ENet I.P. (str)     | 192.168.0.86 |
| ControlLogix Backplane | N/A       | 1756 Backplane Slot(dec) | 0            |

**Figure 7: MSG configuration IP address**

Finally, the destination and target **Data Table Address** plus the **Size in Elements** fields must be specified to complete the MSG instruction configuration. Here the SLC is expecting addresses of the form “N7:3”. The following examples illustrate this aspect. Remember that the examples assume the SLC has the files listed in [TABLE 3](#).

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). Because the same block (for each data type) is used for reading and writing, users have to be very careful in not to overwrite information.

### *Explicit Messaging Examples*

#### Example 1

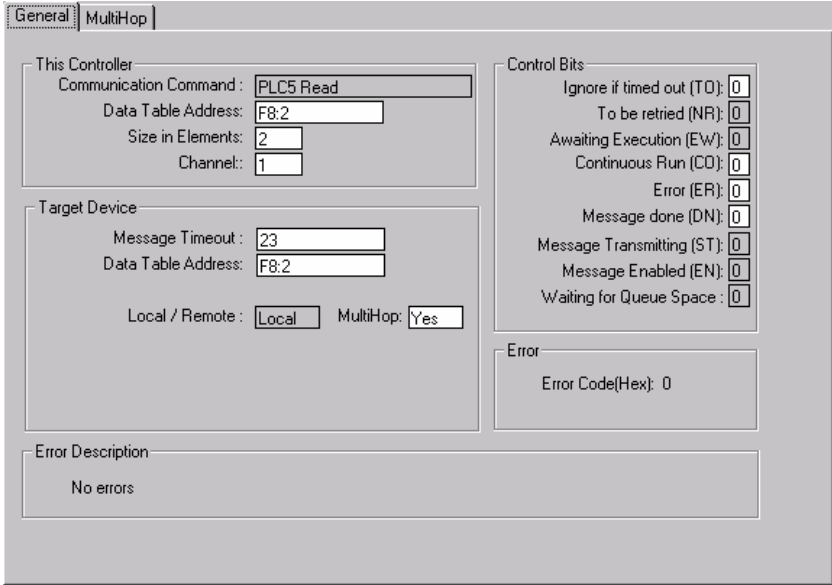
Suppose you are measuring the radius of a hole with at DVT system and you need to send the diameter value out to an SLC5/05. Because we need to do a simple mathematical operation (multiply the hole’s radius by 2 to get the diameter) we need to use a script. Otherwise, if we just had to transfer the radius value we could create a tag in the REAL data block instead of writing a foreground script.

The following example script writes the value from the inspections into the EIP registers for subsequent transfer to the SLC.

```
// Writes a measurement value at index 2 of the
// DVT FLOAT File F8 (this is mapped to the REALS data block)

AB_RegisterWriteREAL(2, (float)(Hole.Radius * 2));
```

In the SLC configure the following MSG instruction:



The image shows a software window titled 'MSG configuration for example 1'. It has two tabs: 'General' and 'MultiHop', with 'General' selected. The window is divided into several sections:

- This Controller:**
  - Communication Command:
  - Data Table Address:
  - Size in Elements:
  - Channel:
- Target Device:**
  - Message Timeout:
  - Data Table Address:
  - Local / Remote: ☐ ☐
  - MultiHop:
- Control Bits:**
  - Ignore if timed out (TO): ☐
  - To be retried (NR): ☐
  - Awaiting Execution (EW): ☐
  - Continuous Run (CO): ☐
  - Error (ER): ☐
  - Message done (DN): ☐
  - Message Transmitting (ST): ☐
  - Message Enabled (EN): ☐
  - Waiting for Queue Space: ☐
- Error:**
  - Error Code(Hex):
- Error Description:**
  - No errors

**Figure 8: MSG configuration for example 1**

In the **This Controller** area, the value of “F8:2” for the **Data Table Address** field tells the SLC that data from the MSG is to be placed in float 2 of float file #8.

In the **Target Device** area, the value of “F8:2” for the **Data Table Address** field tells the DVT system that DVT Float File #8 is to be accessed starting at index 2. Remember that the DVT Float File has 64 elements (0-63.)

Notice we are using F8:2 as the “Data Table Address” for both the controller and the camera, these do not necessarily need to be the same.

### Example 2

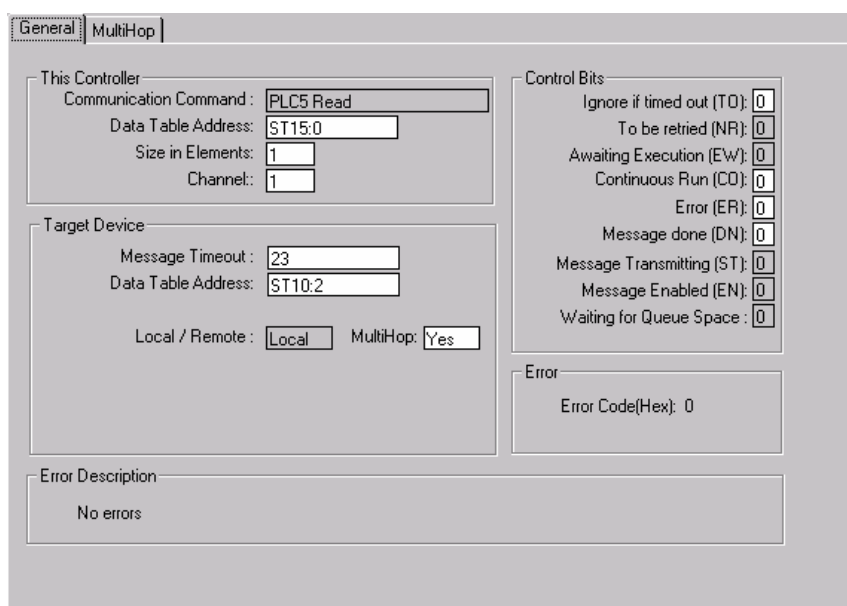
Suppose you are reading a 2D Data matrix code with at DVT system and you need to send the code into a string in an SLC5/05. In this case we will again be using a script instead of tags. This is because, when associating a tag to a string, we need to specify the size in bytes (that is the length of the string, the number of characters plus one for the null terminator). If the length of the decoded string is not always constant, we would need to estimate the largest possible string length. When using a script, we do not need to worry about that since the script function will automatically determine it. We would only need to worry about the length of the string when using scripts in cases where multiple strings are being transferred. In that case, the starting offset for the second string would need to be always larger than the largest string written at the beginning of the data block.

The following example script writes the code from the inspections into the EIP registers for subsequent transfer to the SLC.

```
// Puts DataMatrix Code at index 2 of the
// DVT String File #10 (this is mapped to the SINTS block)

AB_RegisterWriteString (2, Reader.String);
```

In the SLC configure the following MSG instruction:



**Figure 9: MSG configuration for example 2**

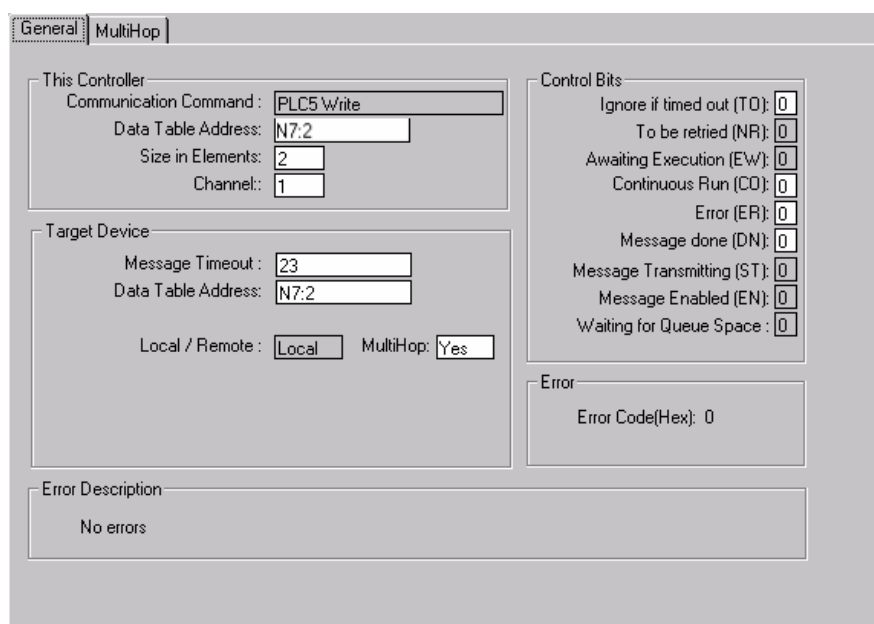
In the **This Controller** area, the value of “ST15:0” for the **Data Table Address** field tells the SLC that string data from the MSG is to be placed in string 0 of string file #15.

In the **Target Device** area, the value of “ST10:0” for the **Data Table Address** field tells the DVT system that DVT String File #10 is to be accessed starting at index 2. Notice how the offset into the DVT string file is treated differently than for the other types. In the case of strings this offset specifies the starting location of the string in the SINTS data block rather than the element index as for all other types. Notice that in the case of strings the SLC only allows transferring 1 at a time. Any other value for the **Size in Elements** field will result in an error. There is a size limitation of 82 characters on strings imposed by the SLC.

## Example 3

Suppose you need to send data to a DVT system from an SLC5/05, specifically a single integer. This situation may arise in application where the operator must change a parameter from time to time without the need to use Intellect. A good example might be the maximum number of defects allowed. Once this value is transferred to the EIP data blocks in the DVT system, it can be accessed by background and foreground scripts and used to make PASS/FAIL decisions.

In the SLC configure the following MSG instruction:



**Figure 10: MSG Configuration for example 3.**

In the **This Controller** area, the value of “N7:2” for the **Data Table Address** field tells the SLC that the data to be sent must come from integer 2 of integer file #7.

In the **Target Device** area, the value of “N7:2” for the **Data Table Address** field tells the DVT system that the incoming data is to be placed in DVT Integer File #7 at index 2. Remember that the DVT Integer File has 128 elements (0-127.)

The following script reads a value from the EIP registers (placed there during the last EIP/PCCC message from the SLC.)

```
// Reads value at index 2 of the DVT integer File #7
// (this is mapped to the INTS Block

int NumDefects;
NumDefects = AB_RegisterReadINT (2);

// Use NumDefects to make PASS/FAIL Decisions
[...]
```

### Sequence of Events For Examples 1 and 2

The correct sequence of events is:

1. With inspections running the script containing the AB\_RegisterWrite() functions has to be executed at least once. 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 without inspections running, the AB\_RegisterWrite() functions will not execute! This is because they are treated as outputs of the system.
2. An MSG instruction from a SLC5/05 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. Note that this message is asynchronous from the camera inspections, so the user needs to make sure that the inspection has, in fact, completed.

### 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 and if the result is PASS.
3. A MSG instruction is then enabled to exchange data between the two systems.

### Sequence of Events For Example 3

The correct sequence of events is:

1. An MSG instruction from a SLC5/05 is sent to the DVT system. This instruction will update the data in the DVT EIP data blocks.
2. An inspection occurs. A script within the inspection reads from the EIP data blocks and uses the data to make decisions.

### Typical Operation:

1. PLC sends MSG to DVT system.
2. PLC triggers the DVT system with a Digital Output.
3. PLC monitors outputs from the DVT system to know when the inspection has finished and if the result is PASS.

## 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 Publication “Communicating with RA Products Using EtherNet/IP Explicit Messaging.”