Using Ethernet/IP to communicate between DVT and ControlLogix Systems

Abstract

This document describes how to use EtherNet/IP to transfer data between DVT systems (Series 600 and Series 500) 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 Control and Information Protocol (CIP). This new protocol allows the transfer of data and I/O over Ethernet.

Types of EIP messages
EIP transactions can be classified into two major categories: explicit and implicit messaging. Explicit massages 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 explicit messages, 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.
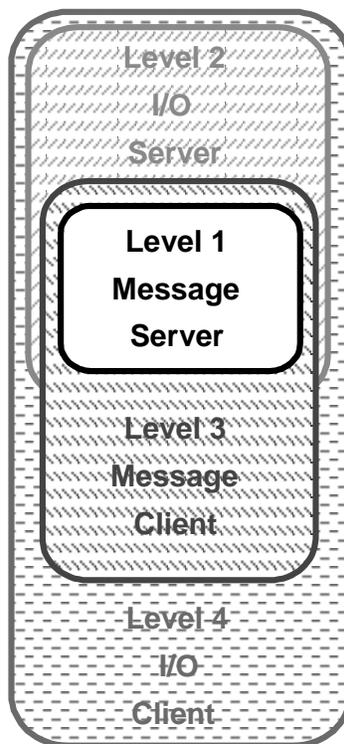

Figure 1: EtherNet/IP Product Classes - Levels of support

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.
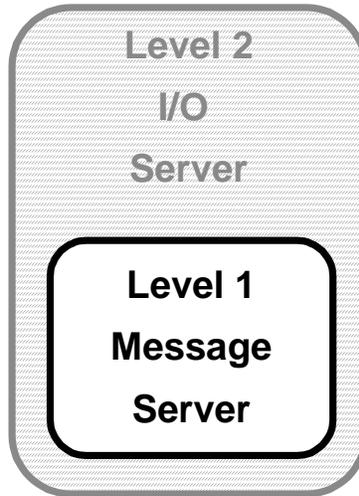

Figure 2: DVT's level of support for EtherNet/IP

Note: 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.

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.
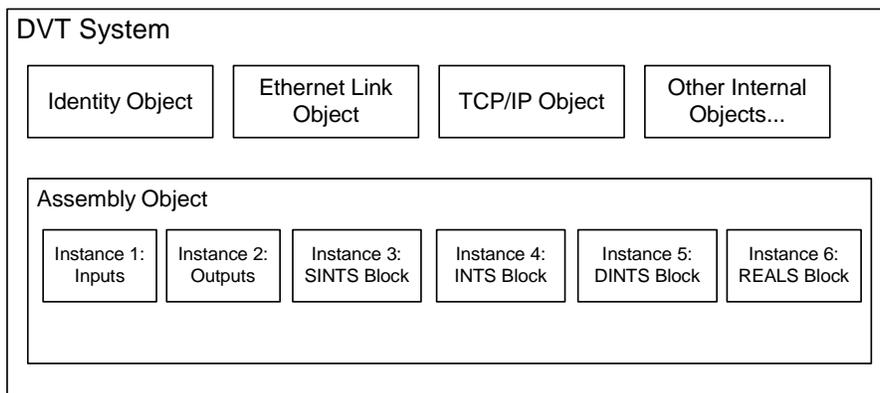

Figure 3: EtherNet/IP objects for a DVT System

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 a Framework terminal. After specifying this option, power must be recycled to the system in order for the protocol to be supported. See paragraphs below for a description of the procedure.

From Framework's User Interface
Connect to the system from the user interface. From the **I/O** menu select **EtherNet/IP Enable**.
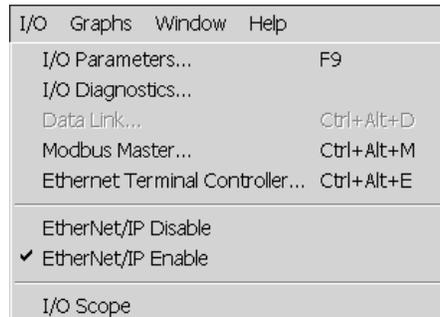

Figure 4: Menu selection to activate EtherNet/IP

Using a Framework 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.

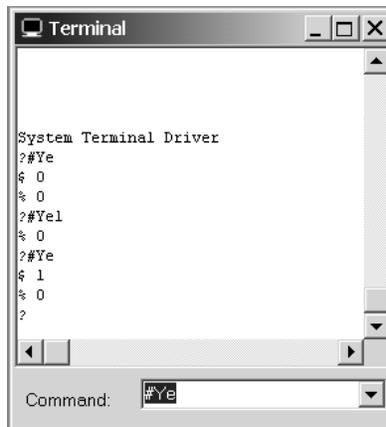
Figure 5: Activation of EtherNet/IP over a terminal.

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 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 Indexes	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.

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 script editor tree.

```
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.

It is important to understand that when these functions are executed data is updated **<u>only</u>** in registers within the DVT system. Communication with the ControlLogix only occurs 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 shall be a destination set of tags and a source set of tags. 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-600 and update one of the destination tags. The MSG instruction can also be used to write data blocks to the DVT system using the source tags as the source of the data. It is important to understand that a MSG instruction **always exchanges an entire data block with the DVT system.** This is the case even is the script functions only access part of the data block. In every transaction the entire 256 bytes of data are transferred. The following figure shows the configuration needed for a MSG instruction.
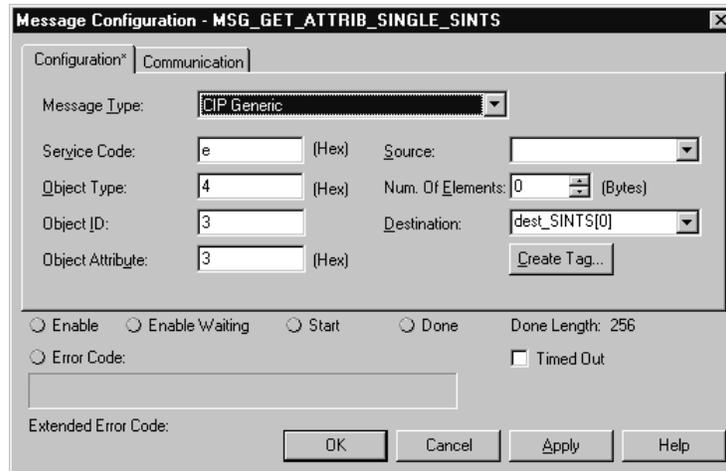

Figure 6: MSG instruction configuration

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 specify the MSG instruction correctly we need to specify the different parameters in the configuration tab.

The **Service Code** 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 specified source tag under the **Destination** parameter. The value of **Num. of Elements** parameter should be left at zero.

- 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** parameter indicates what Tag to use for the source of the data. When using this service is necessary to specify the **Num. of Elements** parameter as the size in bytes of the data to be transferred. Currently all instances of the assembly object (data blocks in the DVT system) have the same size of 256 bytes.

The **Object Type** indicates the type 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.

The **Object ID** 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	Number
Inputs	1
outputs	2
SINTS	3
INTS	4
DINTS	5
REALS	6

Table 3: Instance Numbers for DVT Assembly Object

The **Object Attribute** indicates what piece of information, from a particular instance of an object, is accessed by the CIP MSG. Each Assembly object has several attributes or pieces of data, like the revision number or its own Object ID. 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 all possible MSG instruction configurations that allow explicit data exchange with a DVT system.

Name of service	Service Code	Object Type	Object ID	Object Attribute	Source	Num of Elements	Destination	Meaning
Get_attribute_Single	e	4	3	3	Blank	0	dest_SINTS[0]	Read the SINTS data block from DVT, place it in the dest_SINTS tag
Get_attribute_Single	e	4	4	3	Blank	0	dest_INTS[0]	Read the INTS data block from DVT, place it in the dest_INTS tag
Get_attribute_Single	e	4	5	3	Blank	0	dest_DINTS[0]	Read the DINTS data block from DVT, place it in the dest_DINTS tag
Get_attribute_Single	e	4	6	3	Blank	0	dest_REALS[0]	Read the REALS data block from DVT, place it in the dest_REALS tag
Set_attribute_Single	10	4	3	3	source_SINTS[0]	256	Blank	Write the source_SINTS tag into the SINTS data block in DVT
Set_attribute_Single	10	4	4	3	source_INTS[0]	256	Blank	Write the source_INTS tag into the INTS data block in DVT
Set_attribute_Single	10	4	5	3	source_DINTS[0]	256	Blank	Write the source_DINTS tag into the DINTS data block in DVT
Set_attribute_Single	10	4	6	3	source_REALS[0]	256	Blank	Write the source_REALS tag into the REALS data block in DVT

Table 4: MSG Instruction configuration parameters.

On the Communications tab, the user needs to specify the **Path**. 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.
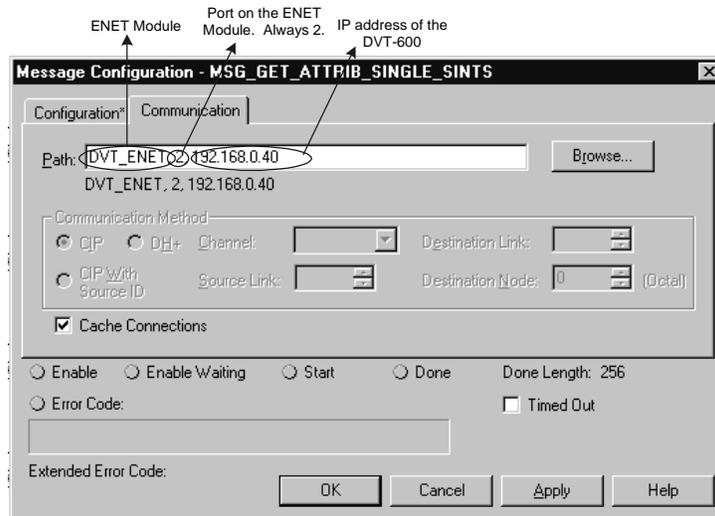

Figure 7: MSG Instruction Configuration – Comm. Tab

The path above can also be expressed as 1,2,2,192.168.0.40. Where the 1 represents the slot number of the processor in the rack and 2 in the second position represents the slot number of the ENET module.

The following diagram shows the direction of data flow between the two systems. The Data blocks in the DVT-600 are accessed both by the PLC and by the inspections through the script. The source and destination tags are shown on the ControlLogix PLC.
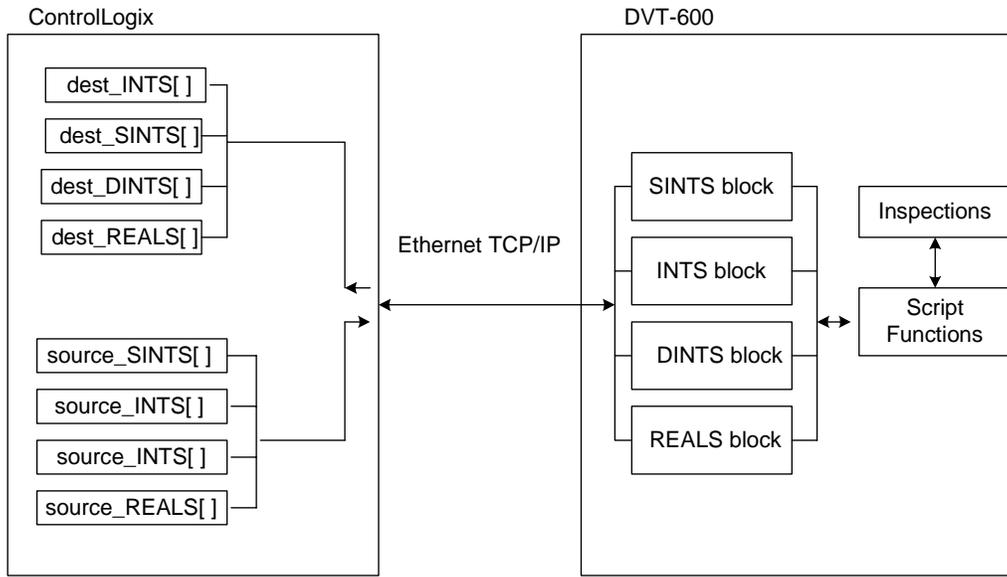
Figure 8: Schematic of data transfer for Explicit messaging

Explicit Messaging Example

<u>Script Examples</u>

The following example scripts write the 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, Selector.NumBlobs);

//Each char of the string is put in the SINTS block starting @ index 10
//and adding a Null termination at the end.

AB_RegisterWriteString (10, Reader.String);

Or

AB_RegisterWriteString (10, "test");
```

The following example scripts read values from the DVT data blocks into local variables for use in the script. The DVT data blocks can 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. 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 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 ALL values in the block at the time.

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.
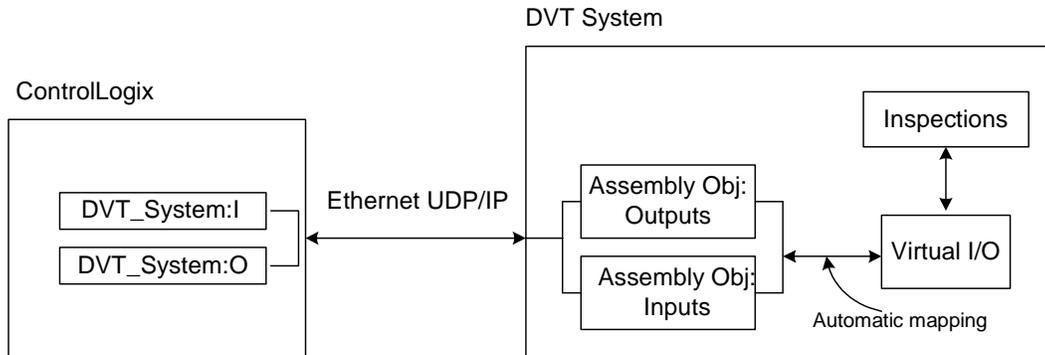
Figure 9: Schematic of data transfer for implicit messaging (I/O connection)

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 a public publication from Rockwell Automation (see references.)
RSLogix 5000 version 8 supports Generic Ethernet Devices. To setup an Ethernet/IP implicit message 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). Add the Generic Ethernet module as I/O device under the node corresponding to the EtherNet module in the rack.

Right Click here and select
New Module…

Figure 10: Adding the DVT system as an I/O device in a Control Logix I/O tree

When the following dialog appears select Generic Ethernet Module. This option is included in the ControlLogix system to allow communications to devices from any vendor that support EtherNet/IP.
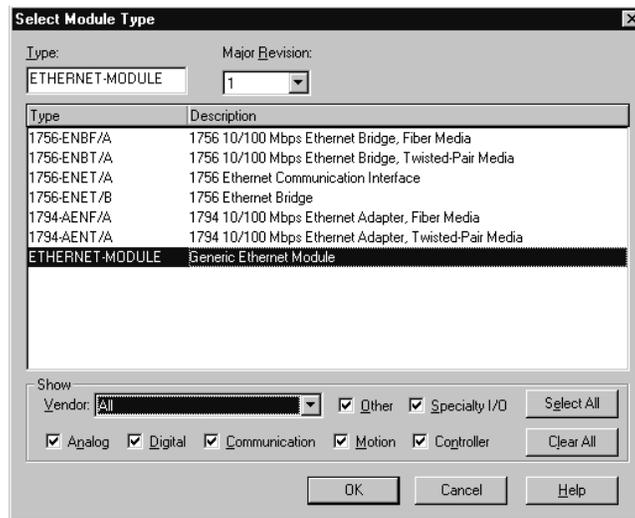

Figure 11: Selection of the Generic Ethernet Module in ControlLogix System

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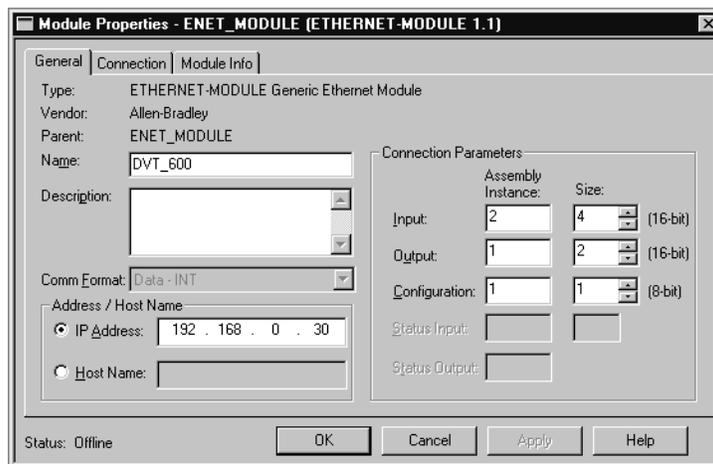

Figure 12: Configuration of the DVT System as a Generic Ethernet Module

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.

Host Name: Optional- use only if you have a DNS server on your network (Domain Name Server.)

Input 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. For instance 2 of DVT's assembly object we know the size is 64 bits, one bit per virtual output 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.

Output Instance: This field should contain the instance number of the Assembly object in the server that acts as the input (output to the client). 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. For instance 1 of DVT's assembly object we know the size is 32 bits, one bit per virtual input point. Thus the size should be set to 2 (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.
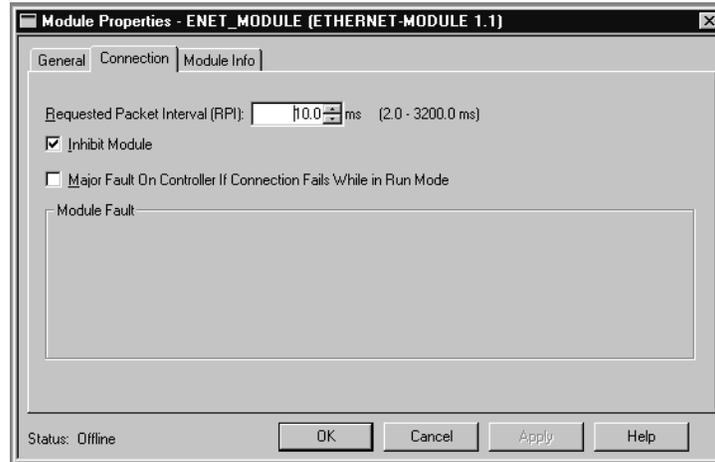

Figure 13: Generic Ethernet Module configuration –Connection tab

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.

Figure 14: DVT System added as to the I/O Tree in a ControlLogix system

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:
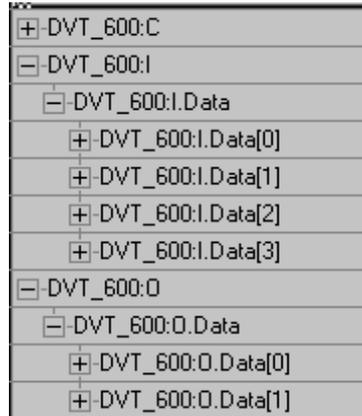
DVT_600: C Configuration Data
This data is not used.

DVT_600: 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_600:I.Data[0]	Insp. Toggle	Inspecting	Acquiring	Run Mode	Mux 4	Mux 3	Mux 2	Mux 1	Select Fail	Select Pass	Res. conflict	Strobe	Busy	Fail	Warn	Pass
DVT_600:I.Data[1]	Output 31	Output 30	Output 29	Output 28	Output 27	Output 26	Output 25	Output 24	Output 23	Output 22	Wrong Code	Strobe 2	Strobe 3	Pf Bus Error	Pf Bus Run	CCD Disc.
DVT_600: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_600:I.Data[3]	Output 63	Output 62	Output 61	Output 60	Output 59	Output 58	Output 57	Output 56	Output 55	Output 54	Ouput 53	Output 52	Ouput 51	Ouput 50	Ouput 49	Ouput 48

Table 5: Mapping of virtual outputs to ControlLogix tabs

DVT_600:O Scheduled Output Data
This Data Sent TO the DVT system from the EIP client.

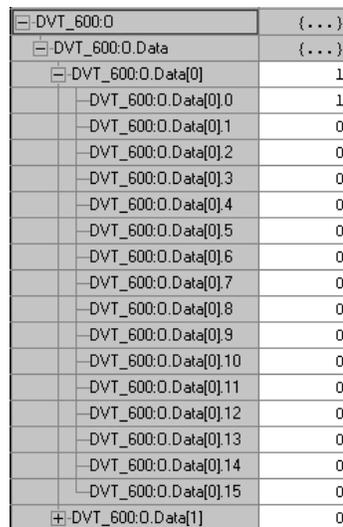| Tag Name	Bit position															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DVT_600:O.Data[0]	ProdID Bit 13	ProdID Bit 12	ProdID Bit 11	ProdID Bit 10	ProdID Bit 9	ProdID Bit 8	ProdID Bit 7	ProdID Bit 6	ProdID Bit 5	ProdID Bit 4	ProdID Bit 3	ProdID Bit 2	ProdID Bit 1	ProdID Bit 0	Select	Trigger
DVT_600:O.Data[1]	Input 31	Input 30	Input 29	Input 28	Input 27	Input 26	Input 25	Input 24	Input 23	Input 22	Input 21	Input 20	Input 19	Dis. Run Mode	Templ Relearn	ProdID Bit 14

Table 6: Mapping of virtual outputs to ControlLogix tabs

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 FWK UI, place the system in external trigger mode with play mode on, create any product and activate inspections. Then in RSLogix, go to controller tags and change the state of bit 0 in the inputs to the DVT system from 0 to 1 as shown below.

⊟ DVT_600:0	{ . . . }
⊟ DVT_600:0.Data	{ . . . }
⊟ DVT_600:0.Data[0]	1
DVT_600:0.Data[0].0	1
DVT_600:0.Data[0].1	0
DVT_600:0.Data[0].2	0
DVT_600:0.Data[0].3	0
DVT_600:0.Data[0].4	0
DVT_600:0.Data[0].5	0
DVT_600:0.Data[0].6	0
DVT_600:0.Data[0].7	0
DVT_600:0.Data[0].8	0
DVT_600:0.Data[0].9	0
DVT_600:0.Data[0].10	0
DVT_600:0.Data[0].11	0
DVT_600:0.Data[0].12	0
DVT_600:0.Data[0].13	0
DVT_600:0.Data[0].14	0
DVT_600:0.Data[0].15	0
⊞ DVT_600:0.Data[1]	0

Figure 15: Triggering a DVT System over EtherNet/IP.

Since bit ***DVT_600: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 ***DVT_600:I.Data[0].15*** of the outputs (mapped to the inspection toggle virtual output) should change state.

⊟ DVT_600:I		{...}
	⊟ DVT_600:I.Data	{...}
	⊟ DVT_600:I.Data[0]	-32764
	DVT_600:I.Data[0].0	0
	DVT_600:I.Data[0].1	0
	DVT_600:I.Data[0].2	1
	DVT_600:I.Data[0].3	0
	DVT_600:I.Data[0].4	0
	DVT_600:I.Data[0].5	0
	DVT_600:I.Data[0].6	0
	DVT_600:I.Data[0].7	0
	DVT_600:I.Data[0].8	0
	DVT_600:I.Data[0].9	0
	DVT_600:I.Data[0].10	0
	DVT_600:I.Data[0].11	0
	DVT_600:I.Data[0].12	0
	DVT_600:I.Data[0].13	0
	DVT_600:I.Data[0].14	0
	DVT_600:I.Data[0].15	1

Figure 16: Observing the Inspection toggle output over EtherNet/IP.

References
- The primary source for information on Ethernet/IP is the specification. The specification can be downloaded from http://www.etherner-ip.org.

- Secondary information sources are available from: http://www.etherner-ip.org.
    - Developer Recommendations White Paper
    - Dev Forum ppt.zip
    - Application Interface Integration.zip
    - CIP over Ethernet Demo.zip
    - EtherNet/IPDeveloper Forum Agenda.zip
    - EtherNet/IPDeveloper Forum Recap.zip
    - EtherNet/IPLevel 2 Example Code.zip
    - IP Network Configuration.zip

- Rockwell Automation Knowledge Base publication ID 13968 "Using Ethernet PanelViews with ControlLogix processors."

Appendix A – DVT's Statement of Conformance