LISTEN.
THINK.
SOLVE.®

INTEGRATION WITH CONTROLLOGIX PROGRAMMABLE AUTOMATION CONTROLLERS (PACS) USING ETHERNET/IP

Concepts and Principles of EtherNet/IP Communication with Rockwell Automation Products

Authors: Vivek Hajarnavis, Richard Piggin, Ray Romito, Viktor Schiffer
Version 1.0

Allen-Bradley • Rockwell Software

Rockwell Automation

Contents

Introduction

Introduction

Many Rockwell Automation products support communication via EtherNet/IP™; however, the type of communication it supports and the details of the communication vary from product to product. The CIP™ Networks Library from ODVA (Open DeviceNet Vendor Association) provides detailed information about EtherNet/IP and the specifications that define Common Industrial Protocol (CIP) as well as its adaptation to various network protocols. What is commonly referred to as "the EtherNet/IP Specification" comprises two volumes (see references [8] and [9]) of this seven-volume library.

All EtherNet/IP-enabled products support the minimum explicit messaging server functionality by allowing explicit messaging access to ID Object and other required objects. Although useful to identify a product, this communication is not the core requirement of an industrial application.

Several guide documents (see references [1] through [6]) detail the communication with Rockwell Automation products. This document does not aim to replace these guides, but intends to highlight the communication concepts.

For more information about the Common Industrial Protocol and EtherNet/IP, please refer to the the CIP eTraining ClassroomTM [10], an instructional CD designed to help developers understand and apply the fundamentals of the Common Industrial Protocol. The CD provides a cost-effective alternative to traditional training and can assist in product development and reducing time to market. For an introduction to EtherNet/IP interface development, the ODVA guide to EtherNet/IP development [11] provides a complete background to EtherNet/IP along with steps to follow for a successful development.

Explicit Messaging

Explicit Messaging

Before the CIP concepts were made public, a range of legacy products were developed. Although these products are true explicit messaging servers (and sometimes clients as well), they use special, older protocol mechanisms such as PCCC (see [5]), meaning they communicate through a vendor-specific object using object-specific services. The data portion of the explicit message contains additional address information.

Newer products, especially those within the Logix family, are explicit messaging servers with fully integrated CIP principles. In order to leverage their capabilities, several product details need to be understood. CIP is the preferred method of communicating to these products.

The Logix family also supports the PCCC encapsulated messages to access controller data organized in PLC-5® style. Please refer to the Logix Data Access manual for details about this communication method.

Use of CIP Routing

Use of CIP Routing

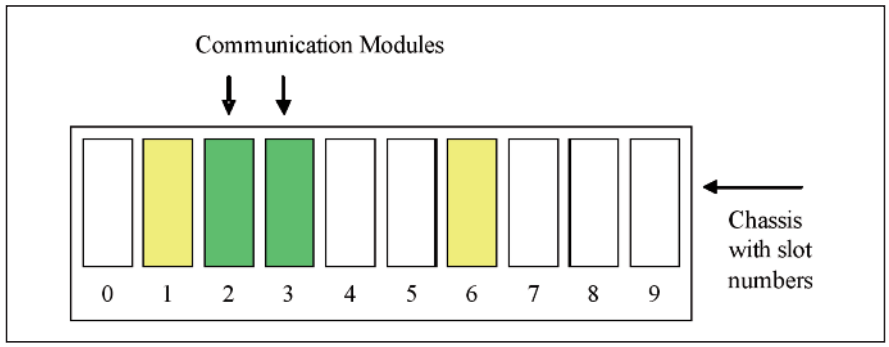

Figure 1 General Principle of a Logix System

As shown in Figure 1, a Logix system is modular in nature, with modules sitting in a physical and/or logical chassis, also called a "rack." Modules in a rack communicate using CIP as implemented on a backplane bus. Network communications modules (e.g. DeviceNet™, ControlNet™ or EtherNet/IP) allow processor modules in the rack to communicate with the outside world via CIP networks.

Communication modules are not an integral part of any processors (at least not logically), meaning that all communication from the outside world into a processor, and vice versa, must use the routing principles built into CIP. These principles state that processors should message across routed connections or through the Unconnected Send service, which carries at least one port segment. To communicate with a Logix system, a routed explicit message (Unconnected Send or Forward Open) must contain a port segment identifying the backplane port (1) and the slot number of the module to be reached. All typical CIP principles apply inside the individual modules of the Logix system.

Data Organization in the Controller

Data Organization in the Controller

Logix processors store and organize all data relevant to the outside world in tags whose user-defined names have meaning in the control application.

Tags have the following properties:
- A name with up to 40 characters

One of two scopes:
- Controller (i.e. global) scope is accessible by external means
- Program (i.e. local) scope is not accessible by external means

A defined data type for organizational purposes:
- Atomic
- BOOL, SINT, INT, DINT, REAL
- These types follow the data definitions in CIP; note that Logix does not support all CIP-defined types. See Appendix C of reference [8] for details
- Structures
- This grouping of atomic data items functions as a single unit for a specific purpose; structures can be predefined (eg: TIMER, COUNTER, CONTROL) or user-defined (eg: UDTs)

Arrays

Arrays

- A sequence of elements of the same data type can be one, two or three dimensional; array members can be atomic or structures

The details of creating and using tags are beyond the scope of this document. Please see reference [7] for further details. For information about how to access tags, please read the chapter entitled, "Appendix A -- Accessing Tags in a Logix Controller."

Outbound Explicit Messages

Outbound Explicit Messages

The MSG (message) instruction handles all Explicit messaging initiated by a Logix Controller program. To utilize connected or unconnected messaging, the user configures MSG instruction. A later section will address how connected messaging can be "cached" to keep the connection open.

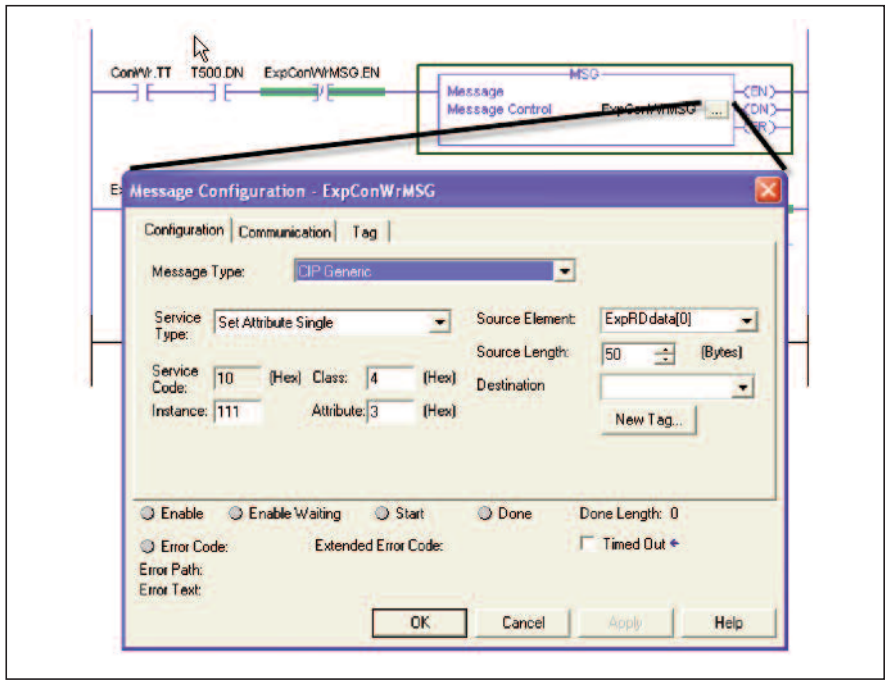

Figure 2 Logix5000™ MSG Instruction Message Configuration

CIP Services that are directed towards data structured into objects can be chosen by selecting "CIP generic" messaging. A user can select from several predefined standard services or enter a valid service code in the "Service Type" field. To access tag-oriented data, select the "CIP Data Table Read" or "CIP Data Table Write" services. Devices that support these vendor-specific services will respond to commands as explained in the chapter, "Appendix A – Accessing Tags in a Logix Controller."

Logix uses default timeout settings that can be changed by the user. These settings are (Please note that items in italics are members of the tag structure for the MSG instruction):

- Approximately 30 seconds for connected explicit messages (ConnectionRate x 2^TimeoutMultiplier x 4); please note that ConnectionRate is the RPI in microseconds

Outbound Explicit Messages

- Approximately 30 seconds for unconnected explicit messages (UnconnectedTimeout)
- Approximately 30 seconds for Forward Open requests (UnconnectedTimeout)

Any communication with another EtherNet/IP product initiated by Logix utilizes the List Services encapsulation command. If a device does not respond to this service (support of List Services is required, per the EtherNet/IP specification), Logix will not proceed with the communication.

Logix EtherNet/IP interfaces automatically create and manage TCP connections and CIP encapsulation sessions. The user has no direct influence on this process. When an interface establishes a TCP connection and CIP encapsulation session, it will continue until it carries no CIP connection or has carried no CIP traffic for 125 seconds. The number of supported TCP connections depends on the interface module used.

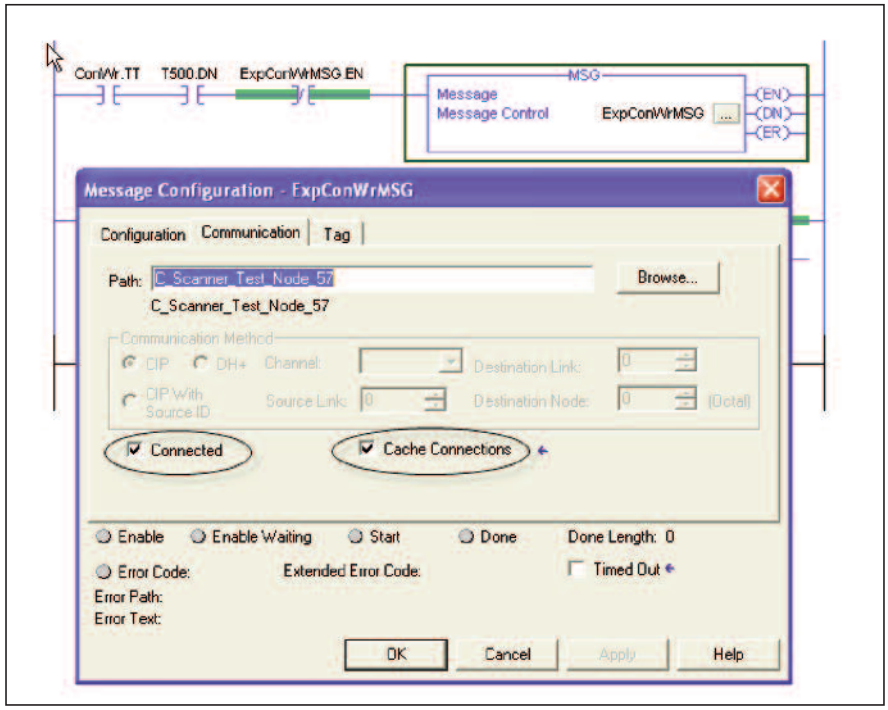

Figure 3 Logix5000 MSG Instruction Communication Configuration

When using the MSG instruction to initiate a connected message, the user may enable the Cache Connection option, which keeps the CIP connection open indefinitely. If a cached connection is inactive for a period equal to the RPI (ConnectionRate) the controller repeats the last message. The repeated message has an unchanged sequence number, which does not affect the application.

Logix controllers have a limit of 32 cached connections. If it exceeds this limit, the controller closes the least recently used connection.

If the user does not enable Cache Connection, the connection closes immediately. An example of this sequence is: ForwardOpen Request, ForwardOpen Response, Explicit Request, Explicit Response, ForwardClose Request, ForwardClose Response. This sequence is repeated each time the MSG rung goes from false to true.

Inbound Explicit Messages

Inbound Explicit Messages

To send explicit messages to a Logix system, connected messaging is recommended instead of unconnected messaging. Unconnected messaging resources are limited and can be come overwhelmed with activity. When this happens, response times become very long and the user application can suffer as a result.

Users should also reuse TCP connections and Encapsulation sessions whenever possible because the Logix server sequentially executes messages in the order received. The client sequentially sends messages using a single connection rather than opening and closing separate connections for each new message. If a message request times out for example, re-try the CIP request. If it fails again, then consider if the session or TCP connection needs to be torn down and rebuilt.

Incoming explicit messages are processed by the message router object (Class ID 0x02). The message router locates the target class code of the service based on the EPATH or ANSI Extended Symbolic Segment, and gives the message to the appropriate object class for processing. See the section “Appendix A -- Accessing Tags in a Logix Controller” for a brief description of the explicit message format.

To close Explicit messaging connections, use the ForwardClose service request instead of letting the connection time out. Too many open, unused connections could result in the Logix system – like any other system – unexpectedly running out of connection buffers.

If a client fails to receive a response to a CIP message request, do not automatically close the EIP TCP session (FIN). Closing the session will cause the EtherNet/IP network interface card to immediately abort the session, which takes several seconds to reopen. Attempt CIP-level retries first. If that does not resolve the issue, close and reopen the CIP connection and try the explicit message request again before closing and reopening a new TCP session.

Explicit messaging connections do not perform as well as I/O data messaging. To yield a timeout of approximately five seconds, an inbound explicit message connection should have an RPI no lower than approximately 1.25 seconds and a connection timeout multiplier set to at least 4x. Although faster RPIs and shorter timeouts may work in ideal situations, heavily loaded systems require longer timeouts.

Explicit Message Response Times

Explicit Message Response Times

Messages get lost or arrive later than expected for myriad reasons. Noise and other network disturbances can cause lost messages, while time-consuming tasks and servers, too preoccupied with other tasks to process incoming requests, result in slow message delivery. If a reply is not received within the RPI time, resend the message using the same

sequence count. Continue resending at the RPI interval until the server receives the response or the client's watchdog times out. If this approach does not work, the client can assume the request has been lost. The client should report this to its application and issue a ForwardClose to free up the reserved resources on the server side.

I/O Message

I/O Message

I/O messaging with a Logix controller can be accomplished in several ways and depends on the version of the controller and RSLogix® 5000 software. In two of the I/O messaging relationships, the controller originates the connection by sending the messages to establish the I/O connection. The controller acts as a target in the third I/O messaging option.

For Version 16 or later, the following options exist:

- I/O Communications via Device Profiles

Add the device to the controller's I/O tree using either the ETHERNET-MODULE or the ETHERNET-BRIDGE profiles in the RSLogix 5000 software so the controller will open a bidirectional I/O connection with the device. For details, please refer to the section "I/O Exchange Using Logix Device Profiles" below. This type of communications can be combined with either of the next two items, if the device has the capability to consume or produce tags.

- Consume Tags

Add the device to the controller's I/O tree using the ETHERNET-MODULE profile and configure Consume Tags in Logix to use this as the data source. Logix will then open a unidirectional I/O connection using a CIP Extended ANSI Symbolic Segment in the connection path. For more information, please refer to the explanation of tag I/O exchanges below.

- Produce Tags

If the device can open a connection using a CIP Extended ANSI Symbolic Segment in the connection path, it can request that the Logix controller produce a tag. Only tags configured as "Produced" – a selection made during tag creation – can be connected to. The device does not need to be in the I/O tree of the Logix controller. For further explanation, please refer to the below section about tag I/O exchanges.

I/O Exchange Using Logix Device Profilles

I/O Exchange Using Logix Device Profiles

I/O data exchange between Logix and networked devices typically use the controller's I/O tree to organize the devices according to how they connect. For example, devices may connect in the local chassis, via a network interface, or in another manner. The controller has device profiles for items in the I/O tree to manage the communications with the device (Figure 4). These device profiles differ from the CIP Device Profiles.

I/O Exchange Using Logix Device Profilles

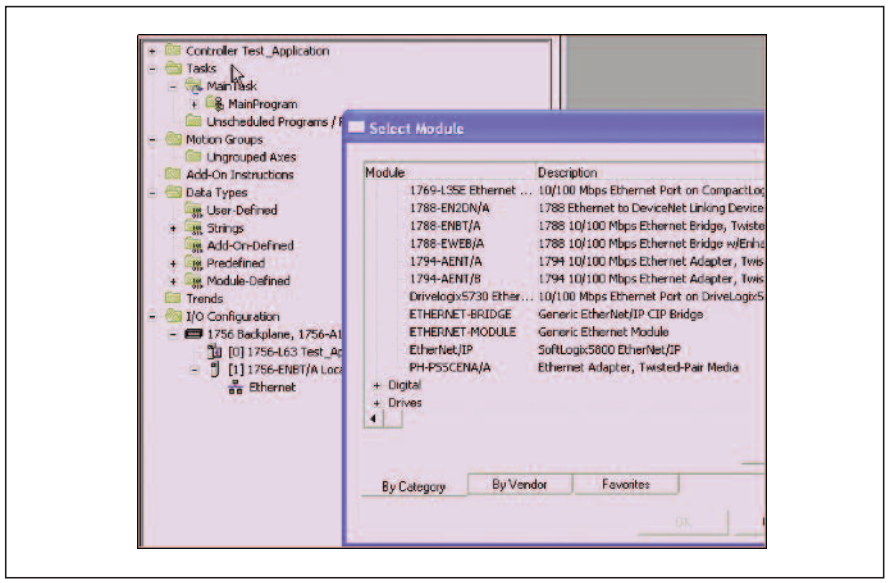

Figure 4 Logix5000 I/O Tree

When connecting third-party EtherNet/IP devices to the I/O tree of a Logix controller, the user can choose either the ETHERNET-MODULE profile or ETHERNET-BRIDGE profile. The ETHERNET-MODULE profile is used for devices directly on EtherNet/IP. The ETHERNET-BRIDGE profile is used for devices with a network adapter and a modular CIP backplane. The controller will individually share real-time data with each slot in the CIP backplane using a single connection for each slot.

ETHERNET-Module Profile

ETHERNET-Module Profile

Configuring the ETHERNET-MODULE profile offers several options for real time data communications through an I/O connection (Figure 5).

When choosing the communications format, selecting the proper data type assures the device's tags correctly represent the data.

The following list outlines the software's only available choices.

Exclusive Owner connections with eight communications format options:

Data – DINT	Data – DINT – with Status
Data – INT	Data – INT – with Status
Data – SINT	Data – SINT – with Status
Data – REAL	Data – REAL – with Status

ETHERNET-Module Profile

Input Only connections with eight communications format options:

Input Data – DINT	Input Data – DINT – with Status
Input Data – INT	Input Data – INT – with Status
Input Data – SINT	Data - SINT – with Status
Input Data – REAL	Data - REAL – with Status

"None" communications format:
This format does not establish any connection at all. Devices using this format have an explicit message relationship with the controller and/or share data with the controller through produced/consumed tags. For more details, refer to the tag I/O data exchanges section below.

The above connection formats identified as "– with Status" have two connections opened with the target; one for the I/O data, and another for the Input Only, "Status" connection. By using the latter connection, the controller receives status information or additional input data from the target device. The target device requires a second set of different connection points – one for the status data, one for the heartbeat – in order to utilize this type of communications format. If the user specifies a configuration size, the first ForwardOpen will contain the configuration data segment. Since it cannot be guaranteed which connection will open first, the device maker should accept the configuration data in either ForwardOpen request.

Please note that Input Only and Listen Only connections appear the same from the controller's point of view because the target side distinguishes the behavior.

ETHERNET-Module Connection Path

ETHERNET-Module Connection Path

The ForwardOpen request parameters include the connection path. It consists of the following elements in the order in which they are sent: electronic key segment, application path, optional port segment and optional data segment.

ETHERNET-Module Electronic Key

ETHERNET-Module Electronic Key

Electronic Key segments contain the vendor ID, Device Type, product code and major/minor revision of the expected target device. Some or all of these fields can be set for "don't care." Before accepting the connection, the target verifies that the identity information matches the electronic key criteria. The ETHERNET-MODULE profile sends an Electronic Key with fields set for "don't care" (zeros).

ETHERNET-Module Application Path

ETHERNET-Module Application Path

The Application Path consists of four items: the Class ID, Configuration Instance, Consume Connection Point and Produce Connection Point. The ControlLogix® controller sends all four items with the ETHERNET-MODULE profile. The Class ID value is always 0x04, which refers to the Assembly object class. The user enters the remaining fields in the module properties screen shown below using values from the device's user manual. For devices that support multiple connections, each connection type has one set of numbers. For example, if the product supports an Exclusive Owner

ETHERNET-Module Application Path

and an Input Only connection, the Produce Connection Point for each connection can be the same value, likewise for the Configuration Instance. But the Consume Connection Point for each connection must be different. Mismatches between Connection Point number and size prevent the connection from opening.

Application Path encoding appears in the form of four logical segments. For example, 20 04 24 03 2C 64 2C 65 corresponds to the Connection Parameter values entered in the new module properties screen shown in Figure 5.

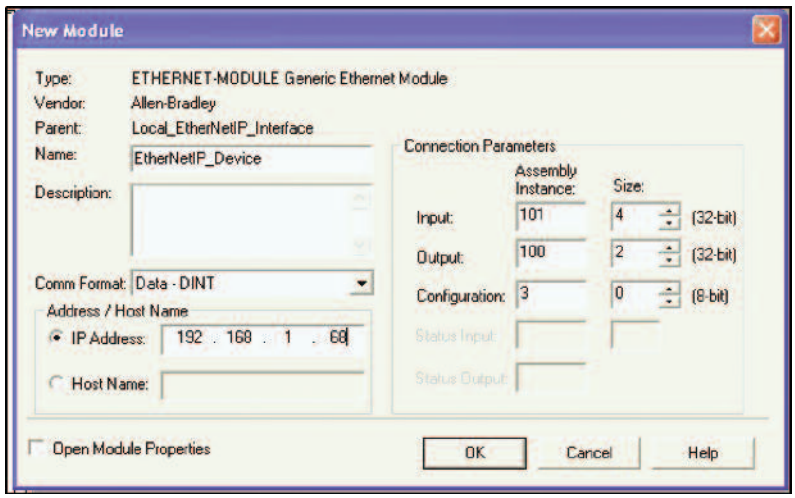

Figure 5 Configuring I/O connections in Logix5000 software

Since RSLogix 5000 software does not read information from device EDS files, the user must choose the heartbeat connection point (assembly object instance) provided by the target device manufacturer. This connection point distinguishes Input Data, Output Data, Input Only and Listen Only connections in instances where the target device supports multiple connection types.

Connection Data Sizes

Connection Data Sizes

The data sizes entered in this screen determine the size of the data in the packets on the wire. However, the ForwardOpen request includes one or two other protocol transport items that are unnoticeable in this view.

To determine the value in bytes of the ForwardOpen, consider the data size specified in the Comm Format field. In Figure 5 above, the data type is DINT, which is 4 bytes long, so the Input data size is (4 * 4) or 16 bytes and the Output data size is (4 * 2) or 8 bytes. All I/O data packets also include a two-byte-long sequence count, which adds two bytes to those totals. Finally, output data contains a 32-bit Run/Idle header that conveys controller mode (Run/Program) to the target. This adds four bytes to the output size. This yields the final values of 18-bytes for Input data and 14-bytes for Output data. These values are used in the ForwardOpen T-O Size and O-T Size fields, respectively.

Configuration Path

Configuration Path

Product manufacturers are strongly encouraged to support a Configuration Assembly in their products, and to accept configuration data in a Data Segment appended to the ForwardOpen service request. Doing this allows devices to be automatically reconfigured when being replaced. Providing automatic device configuration on replacement results in efficient system operation, improved user experience and if a device fails, minimal downtime.

All Comm Format selections include Configuration Instance and Size fields. If the device does not support this feature, enter "zero" in the configuration box. This configuration will have no configuration data at the end of the ForwardOpen Request. If the target device supports receiving configuration data via the ForwardOpen request, then the expected Data Segment size can be entered here. The controller creates a 400-byte tag in user memory, called "C" suffix data, for nonzero size entries. For more information about "C" suffix data, please refer to the Controller Tags section of this document.

Based on information from the device's manual, the user can enter the configuration data in this tag. A Data Segment (Logical Segment type 0x80) added to the end of the ForwardOpen request delivers the data from this tag to the device. Since the Data Segment size specifies the number of bytes in words to follow CIP protocol, it adds an extra byte for odd-sized entries in the module properties screen. A Data Segment can send up to 400 bytes of configuration data.

CIP Routing

CIP Routing

CIP routing information describes the network hops required to get from the originator to the target. For more information, please refer to the Volume 1 of the CIP Networks Library, "Appendix C: Data Management" as this document will not define the format of this. If CIP Routing information is present in a ForwardOpen request, it immediately precedes the Configuration Path, and contains the necessary Port Segments to traverse hops between the controller and the target device.

ETHERNET-Bridge Profile

ETHERNET-Bridge Profile

For modular devices that have a CIP backplane and resemble a remote ENBT in a chassis, use the ETHERNET-BRIDGE profile. These devices contain an adapter (in slot zero) with slots 1-99 available to insert a CIP Generic module. The ETHERNET-BRIDGE profile provides support for modular devices and can achieve multiple connections to a non-modular device if it can simulate a CIP backplane with slots. With the exception of the Version 16 and earlier versions that do not have a "none" communications format, it supports the same connection types as the ETHERNET-MODULE (See the ETHERNET-MODULE Profile above).

Controller Tags Associated with Generic Devices

Controller Tags Associated with Generic Devices

Each ETHERNET-MODULE or ETHERNET-BRIDGE inserted into the I/O tree will result in the creation of tags in the controller's tag database (Figure 6).

Organized according to function (e.g. Input data, Output data and Configuration Data), the tag name is based on the module tag name entered in the Module Properties screen. Using the example above, "Remote_Bridge:0" (the zero refers to the slot in the CIP Bus), is followed by a "I", "O" or "C", and then an array "Data[n]". The number of elements in the array corresponds to the size entered in the Module Properties screen for this device.

Controller Tags Associated with Generic Devices

These tags resemble those created by the ETHERNET-MODULE profile although they do not contain a slot number.

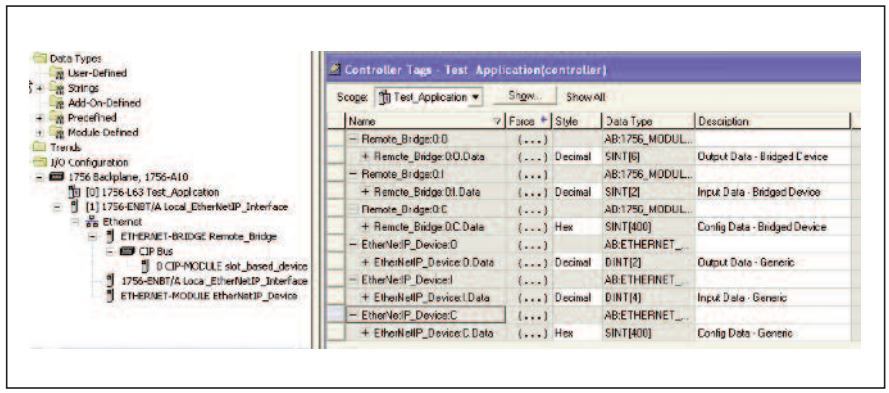

Figure 6 Creation of controller tags in the Logix5000 I/O Tree

I/O Using Tag I/O Data Exchange

I/O Using Tag I/O Data Exchange

Logix controllers provide the ability to exchange data through direct connections to tags, known in Logix parlance as produced/consumed tags. This method of I/O data exchange allows the exchange of tags (symbolic representations of data items) by referencing the symbolic name instead of the assembly object instances, as described earlier. Since no current mechanism can verify this, I/O data exchange assumes that both ends of the data transfer understand the structure and content of the data being transferred.

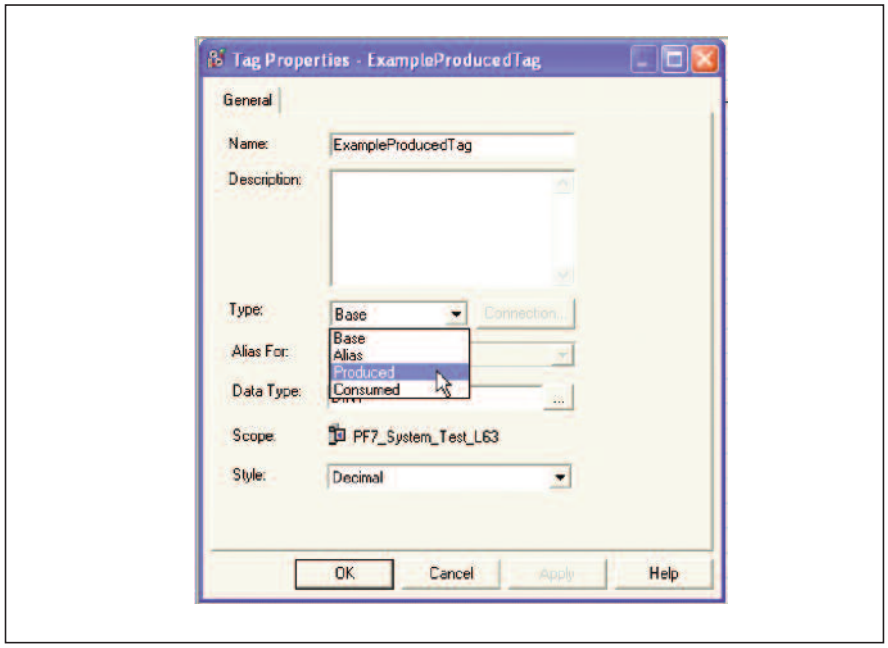

Figure 7 Configuring a Produced Tag in Logix5000

I/O Using Tag I/O Data Exchange

A tag must be controller scope and of type "Produced" so another device can make a connection to it (Figure 8) To request a produced tag connection with the Logix controller, a device does not need to be in the controller's I/O tree. The unidirectional, produced tag connection does not provide controller status information (eg: Run/Program mode).

Similarly, the user denotes a tag as a "Consumed Tag" during tag creation. Consumed Tags require the user to enter the remote tag name and browse to the tag-producing device, as shown in Figure 8. The producing device must be present in the Logix controller's I/O tree to be a consumed tag data source. See the sections above pertaining to ETHERNET-MODULE and/or ETHERNET-BRIDGE for a description of how to add a device to the I/O tree. The unidirectional Consume Tag connection does not expect to receive controller status (e.g. Run/Program mode) from the producer.

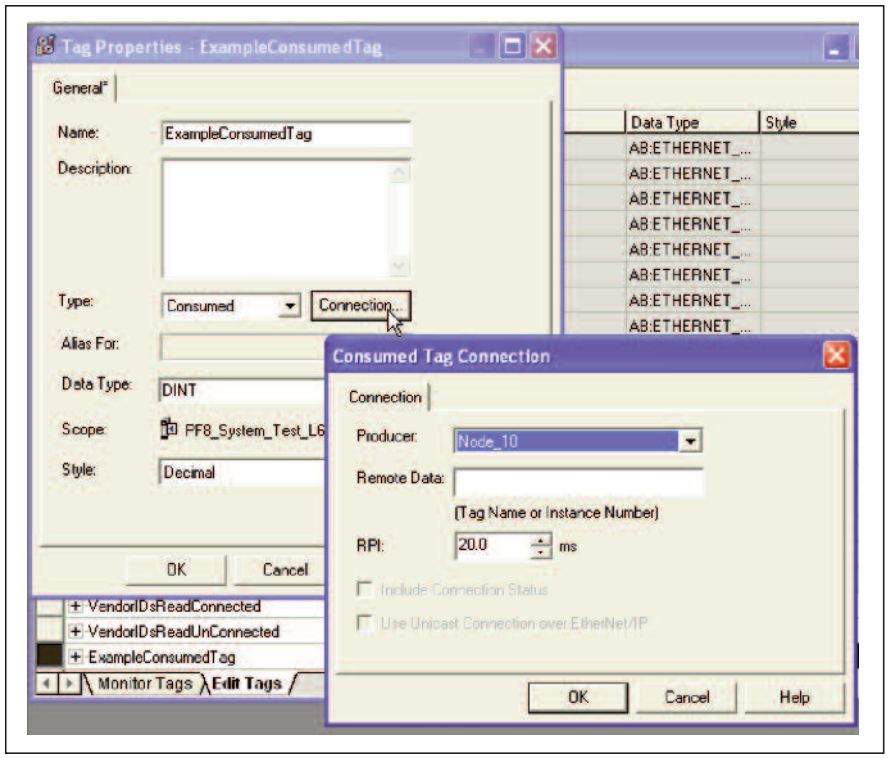

Figure 8 Configuring a Consumed Tag in Logix5000

The event task allows users to implement COS behavior for consumed tags. For more information about this task, please refer to the Rockwell Automation Support Knowledge Base.

The only types of target I/O connections accepted by the Logix system are the produced tag connection, and Input Only connection described in the EDS files of the Logix controllers.

I/O Using Tag I/O Data Exchange

The following connection restrictions apply when Logix acts as the target device:

- Only cyclic connection transport triggers will be accepted. Logix does not support change-of-state and application-triggered connections.
- RPI must be at least 2 ms.
- Logix only accepts connections to a data entity with a tag name, not connection points.
- Connections can be set up with O→T unicast and T→O as multicast or unicast when if the target (producer) is a Logix controller V16, in RSLogix 5000 V16. If it is an ETHERNET-MODULE the unicast option is grayed, in which case define the producer as a Logix controller V16.

I/O Messaging Presumptions Made in Logix

I/O Messaging Presumptions Made in Logix

Several connection settings implied within Logix cannot be changed, despite increased freedoms allowed by CIP specifications. The following settings apply to controller-originated connections such as consumed tag connections and I/O connections made by devices in the I/O configuration tree:

- Only cyclic connections can be created. Logix does not support change-of-state and application-triggered connections, with the exception of the consumed tags mentioned above.
- RPI values are shown in ms with a resolution of 100 μs, even though the wire transmits RPI values in μs.
- For Exclusive Owner connections, it uses the same RPI values for the O→T and T→O directions, even though CIP allows setting them individually per direction.
- It chooses The Connection Timeout Multiplier to yield a resulting timeout of at least 100 ms.
- For Input Only connections (with I/O Tree and I/O tag exchanges), the controller, beginning in Version 16, adjusts the O-T RPI to slow the rate of heartbeat transmissions. The controller selects a RPI and Connection Timeout Multiplier for the O→T direction that yields an approximately two-second connection timeout.
- For Input Only connections, Logix chooses a Connection Timeout Multiplier for the T→O side to establish a resulting timeout of at least 100 ms.
- Logix currently limits the RPI range to1.0-3,200.0 ms.
- By choosing a "... - with Status" connection, the status connection uses the same RPI value as the I/O connection.
- To establish a connection, Logix sends three Connection Points in the ForwardOpen Message, even if it has a zero length configuration size specified in the Module Properties dialog. In this case it does not send configuration data with the Forward_Open request either. Consumed tag connections use a symbolic segment, which the I/O Tag Exchanges section discusses in greater detail.
- When creating a connection, Logix sends an empty Keying Segment that can neither be switched off nor filled with values.
- Data in the O→T direction is sent unicast, while data in the T→O connection is sent multicast for I/O connections. Beginning with Version 17, the user has the ability to select unicast in the T→O direction for Produced and Consumed tags.

CIP does not define the behavior of multiple connections that have the same producing target, although Logix behaves as follows:

- The first arriving connection has the requested RPI value(s).
- If the second connection comes in with a RPI value larger than the existing API value, Logix accepts the connection and adjusts the returned API value for T→O to the existing API.

I/O Messaging Presumptions Made in Logix

- If the second connection comes in with a RPI value smaller than the existing API value, Logix accepts the connection and adjusts the returned API value for T→O to the new, smaller value. Existing connections with a larger RPI will receive the desired data at a faster rate.
- When one of several existing connections goes way, the production rate of the target data slows to the smallest API value of the remaining connections.

Length limitations of CIP data:

- I/O: 500 bytes for input data, 496 bytes for output data. They differ in length because the output tag does not include the four bytes of real-time header. If the input data from an adapter contains a real-time header, Logix treats these four bytes as typical input data.
- Explicit messaging, connected: ~486 bytes sending/receiving
- Explicit messaging, unconnected: 502 bytes. When sending data the unconnected the route path size must be subtracted (minimum 1 word), since EtherNet/IP includes the IP address in ASCII format.
- Configuration data: 400 bytes
- Produced Tags: 500 bytes

Special behavior of CompactLogix™ and FlexLogix™ controllers:
When creating I/O or explicit connections, CompactLogix and FlexLogix controllers accept the same RPI values as other Logix controllers and return the same values in the API fields in the ForwardOpen response. They produce packets at an RPI rate that is an integral power of two milliseconds, with a maximum of up to 1024. Since these controllers use the highest possible value without exceeding the requested RPI, packet rates can attain speeds nearly twice as fast as expected.

For explicit connections, this behavior affects the rate of sending duplicate packets to keep the CIP connection alive. Default settings send duplicate packets every 1024 ms instead of every 7500 ms.

Routing of I/O data:
Most I/O connections should be used within the local subnet. With the exception of produced tags, I/O connections should be set up with a TTL value of "one." These routable, produced tags (RSLogix Version 14 and higher) should have a TTL value as configured for the TCP/IP stack.

Further Design Considerations

Further Design Considerations

Unconnected messaging functionality in the ControlLogix controller is intended primarily to establish communication with other devices. Consequently, the use of unconnected messaging for obtaining device information instead of using connected messaging can result in slower establishment of I/O connections and slower messaging overall. To maximize performance, use connected messaging wherever possible. This requires that devices support connected messaging.

As a general rule, explicit communication requires greater care than I/O data, as discussed in the explicit messaging section of this guide. Extensive use of this technique impacts control performance in larger customer implementations where a controller

Further Design Considerations

communicates with several EtherNet/IP devices. Since transferring information over the I/O channel is easier to use and more efficient, consider including parameters that an end user might need to frequently read or write within an I/O connection. This helps dedicate explicit messaging for infrequent configuration changes.

Planning your EtherNet/IP Implementation for successful integration with RSLogix 5000 software:

- When designing your EtherNet/IP interface, consider the user experience. Given the different mechanisms for communicating with the device, these choices will impact processor performance and how the user utilizes the device.
- Consider how the device may be used in a typical customer implementation. Would the user be expected to configure a single device or multiple devices with a single controller? In the latter scenario, a well-planned implementation ensures that the processor effectively utilizes resources for explicit messaging.
- Consider the nature of the device's data and the frequency of user access. Since users regularly transfer data as required, data transfer should be included in the I/O channel.
- To ease integration and use processor resources most effectively, design the EtherNet/IP interface to minimize explicit messaging during normal operation. Product developers can accomplish this by placing as much information as possible in the I/O channel, when they design their product.
- If the situation requires a significant amount of explicit messaging, consider designing a vendor-specific "user data" CIP object that holds relevant information and can be obtained using a single MSG instruction.
- To minimize the use of unconnected message buffers, implement connected messaging functionality in the device.

To improve user experience and product integration with Logix controllers, consider providing specific logic for EtherNet/IP communications. Logic can be implemented using RSLogix 5000 Add-On Instructions, as well as HMI integration with FactoryTalk® View software. The following pages provide an overview of the benefits and implementation of Add-On Instructions and FactoryTalk View software, illustrated with a case study.

EtherNet/IP Communication with Add-On Instructions

EtherNet/IP Communication with Add-On Instructions

RSLogix 5000 Version 16 introduces the concept of reusable code objects called Add-On Instructions (AOIs) (Figure 9). Designed to improve standardization and code reuse, these instructions enable encapsulation of commonly used logic as sets of reusable instructions.

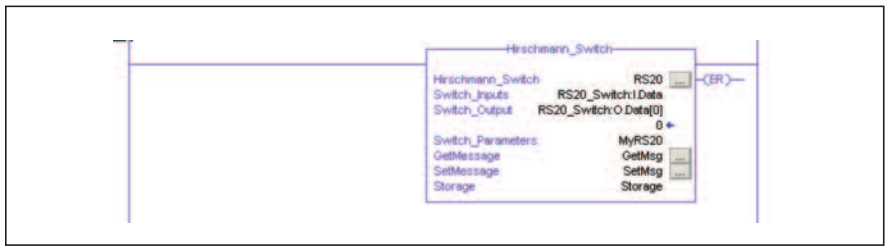

Figure 9 Add-On Instructions in RSLogix 5000 software encapsulates logic to create sets of reusable instructions

Introduction
Why Add-On Instructions?

Introduction – Why Add-on Instructions?

Vendors use Add-On Instructions with EtherNet/IP to set up communications between a Logix Controller and their product with RSLogix 5000 software. By providing Add-On Instructions with commonly used instructions for communications, users can save time and increase consistency, while reducing errors and the need for technical support. Add-On Instructions can also be used to enhance customization in the specific application of products, or across a range of products with different features. Users tend to utilize more features when provided with Add-On Instructions featuring full product capability because it is easier to use.

The creation of a user-defined data type (UDT) that holds data relevant to the device (Figure 10) serves as the basic principle behind this technique. The AOI code populates the UDT. Implicit data from the input channel can be easily and directly copied to the UDT, while placing explicit data in a UDT is more complex. Selecting appropriate element names within the UDT presents device information in a format appropriate for that device. The AOI provides the opportunity to process incoming data from the EtherNet/IP object and present it in a meaningful format to the user. Similarly, output information can be copied from a UDT to the output channel, and parameter changes in the device can be made through the "Set Attribute Single" service.

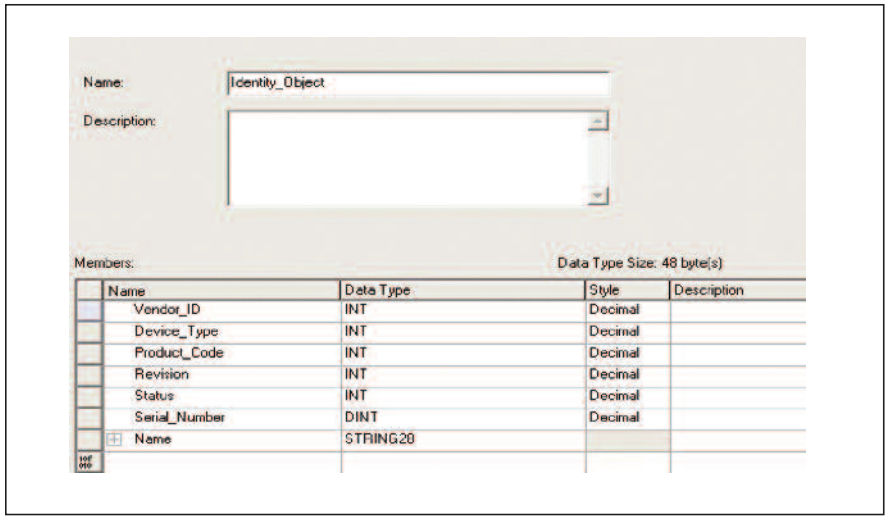

Figure 10 UDT (User Defined Data Type) with Identity Object data

Introduction
Why Add-On Instructions?

Figure 11 shows a vendor authored AOI, with data input and output definition and AOI code (in this case written in ladder logic).

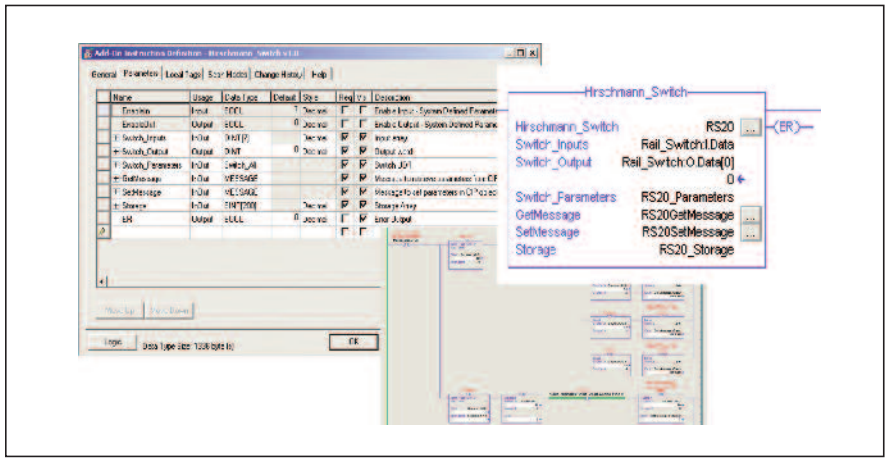

Figure 11 Creating an Add-On Instruction – a vendor-authored instruction for EtherNet/IP with implicit and explicit communication

Generic Benefits of Add-On Instructions

Generic Benefits of Add-On Instructions

- Time Savings:
  Encapsulating commonly used logic as a set of reusable instructions saves time.
- Improved Change Management:
  Add-On Instructions share code throughout the controller and automatically inherit modifications.
- Consistency:
  Reusing instructions promotes consistency and reduces errors.
- Troubleshooting:
  Context views help "visualize" the logic for a specific use and simplify troubleshooting.
- Protection:
  Using the RSLogix 5000 source protection capability limits access to instructions and stops unwanted changes with "view only" or "no access" status.
- Online Documentation:
  Online integration helps distribute documentation instruction as part of the code (Figure 12).

Generic Benefits of Add-On Instructions

Benefits when used in conjunction with EtherNet/IP include:

- Minimal configuration and messaging instruction setup (just Set and Get MSG blocks)
- Removes the user requirement for device CIP implementation knowledge
- User-friendly presentation of explicit and implicit data
- Processing and presenting of data in meaningful format without any further intervention
- The ability to handle large number of variants (such as a product range) with no additional user input
- Ease of replication
- Add-On Instructions may be combined with RS View global objects to provide pre-configured HMI screens or faceplates

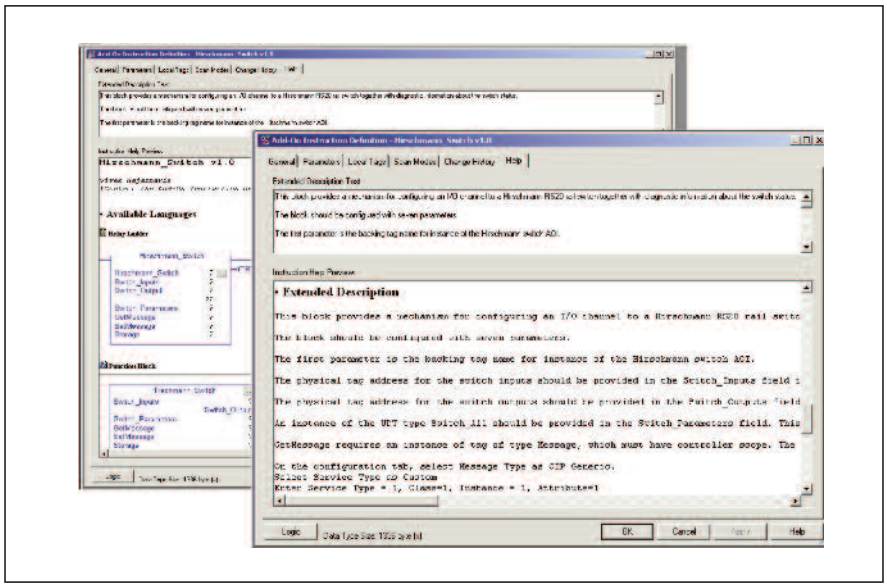

Figure 12 Add-On Instruction help in RSLogix 5000 software

As a basis for object-oriented programming methodology, Add-On Instructions encapsulate the code into pre-validated modules for easy reuse. Add-On Instructions may be created using the Ladder Diagram (Figure 13), Function Block Diagram and Structured Text editors available in RSLogix 5000 software.

Once created, the instructions may then be used in any of the RSLogix 5000 programming languages without additional effort. This feature of RSLogix 5000 does not require additional software or licenses. Users can create and use add-on instructions with RSLogix 5000 Version 16. If users have an earlier version of RSLogix 5000 software that does not support AOI, using sample code achieves similar functionality. As with all sample code, users should exercise care when replicating code for multiple instances of a device.

Generic Benefits of Add-On Instructions

Sometimes, it may be inappropriate or undesirable for end users to add or remove functionality from the CIP interface code or AOI. Since CIP and AOI have a standard application code, anyone can easily modify them without additional programming tools or licenses. RSLogix 5000 security features can prevent the code from being viewed or modified. For more information about RSLogix 5000 security features, please refer to the product user manuals and the Logix security tool help.

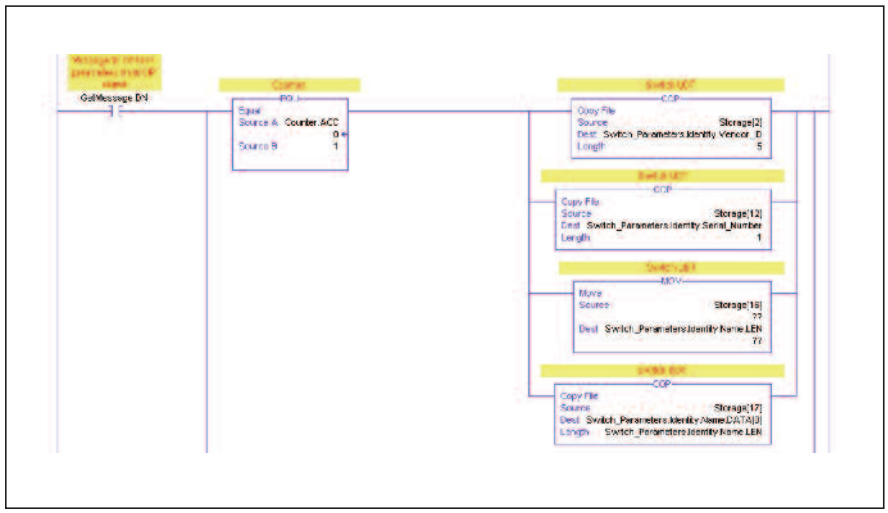

Figure 13 AOI Ladder code used to move data into a UDT

Creating an AOI

Creating an AOI

For integration purposes, an AOI typically performs the following functions: transferring data over the I/O channel and explicit messaging to obtain or set device parameters together with the necessary data processing.

Working with implicit data on the I/O channel requires Copy (COP) or Move (MOV) instructions to transfer data, as shown in Figure 14 and Figure 15.

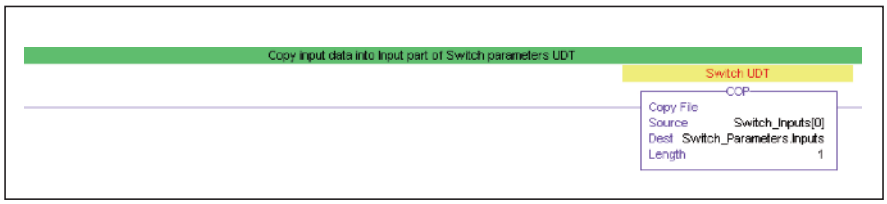

Figure 14 Code to copy data from device input word to UDT

Creating an AOI

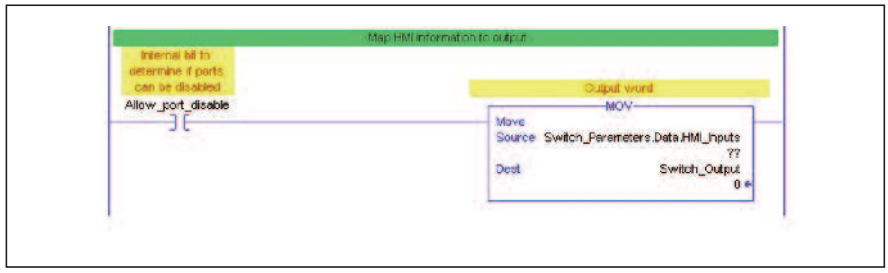

Figure 15 Code to copy data from UDT to device output word

Implicit data from the I/O channel and information from any CIP object can be obtained through explicit messaging using the MSG instruction. Although multiple MSG instructions can be configured to poll individual CIP objects, this approach complicates AOI configuration and use, as well as places heavy demands on the controller's explicit message buffers. Alternatively, an AOI or code can be created to acquire data from several CIP objects with the class, instance and attribute settings for the MSG instruction changed after completion ready for the next scan of the program (Figure 16). An input state, program condition or a timer can trigger CIP object polling. For many continuous operations, the timer is the most appropriate option. While timer-based messaging allows data to be read from several objects, it can also lead to slow cycle times, particularly if several objects need to be interrogated in turn. Polling CIP objects on a "condition based" basis uses controller resources more effectively because it only triggers the MSG instruction when necessary. For example, certain types of information can be obtained by pressing a control panel button, or continuously retrieved by selecting the appropriate screen on an HMI system.

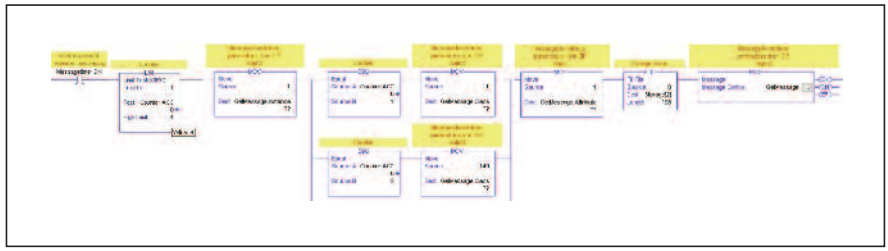

Figure 16 MSG instruction used to obtain data from multiple CIP objects

When designing the device, consider creating a single, vendor-specific CIP object that contains relevant information an end user may require from the device. This allows users to obtain device information through a single MSG instruction, which can contain large quantities of information. This streamlines the data acquisition process while enhancing programmer usability.

Steps to Building an AOI

Steps to Building an AOI

1. **Determine the most relevant data.** Identify which device parameters and functions a user wants to access or configure.

2. **Determine the data's location: which objects and whether the data is implicit or explicit.** If the I/O channel does not contain relevant information, determine which objects contain the parameters of interest.

3. **Create UDTs. Replicate the structure of the input and output channel, as well as relevant objects as necessary (Figure 10).** Combining several UDT instances into a single structure can aid replication and ease integration with visualization tools.
4. **Copy input image into UDT.**

5. **If using explicit messaging, determine whether a time or condition-based polling mechanism should be implemented.** The nature of the device and type of data will determine the most appropriate approach. For example, device-specific information will not change unless the unit is disconnected. A disconnected device has a single poll of the identity object following a status change, which may be sufficient. A time-based polling mechanism may be most appropriate for other parameters, such as the device's continuously changing CPU temperature. As stated previously, avoid time-based polling in order to best utilize processor resources.

6. **Configure message instructions – to ease user configuration, use one message per CIP service.** Using a single MSG instruction for multiple objects (Figure 16) further streamlines configuration.

Since a MSG instruction needs to be configured each time a user creates an AOI, users should utilize as few MSG blocks as possible. To accomplish this, set up one MSG block for each service. For example, configure one MSG with the "Get Attribute All" service and another as "Set Attribute Single." Class, Instance and Attribute values can then be set within the AOI code. Although creating an AOI requires additional programming effort, it simplifies AOIs and eases use.

7. **Copy explicit data into a UDT.** The MSG instructions should be configured to read the data received from the device as a temporary storage array. To move data in this storage array into an instance of the UDT, use MOV and COP instructions (Figure 14 & Figure 15).

8. **Evaluate raw data and provide meaningful information to user.** Data received from the device may not be in the most user-friendly format, so processing may be necessary. For example, users could byte swap devices using the big-endian format or configure a MAC address into a user-readable character string.

9. **Configure second message with "Set Attribute All" service to set necessary device parameters.**

10. **Copy data from UDT to output image.**

11. **Prepare online documentation.** Ensure sufficient information is entered in the online help file for the AOI. The file also can include contact details for support purposes.

Steps to Building an AOI

12. **Set security settings for AOI.** Determine whether the code in the AOI should be invisible, visible but locked, or open to all for viewing and modification.

13. **Make your AOI available to customers.** Approaches include downloading from vendor Web sites and uploading a vendor sample code to the Rockwell Automation sample code site. Sample files for RSLogix 5000 and FactoryTalk® View softwarecan be found here: http://samplecode.rockwell.com.

For more information, videos and tutorials for add-on instructions as well as other features in Version 16, visit the Integrated Architecture™ Tools Web page at www.rockwellautomation.com/go/v16tj.

Integration with FactoryTalk View Software

Integration with FactoryTalk View Software

Creating and populating a UDT allows users to utilize global objects in FactoryTalk View. Instances of these pre-prepared screens can be created within a FactoryTalk View package (Figure 17). Since the RSLogix 5000 tag database has pre-defined links, user configuration is limited to providing a path that indicates the specific device instance within a parameter file. At run-time, the screens are identified by the parameter file and transfer data with the UDT. Like sample code and AOI, users can easily customize these global objects.

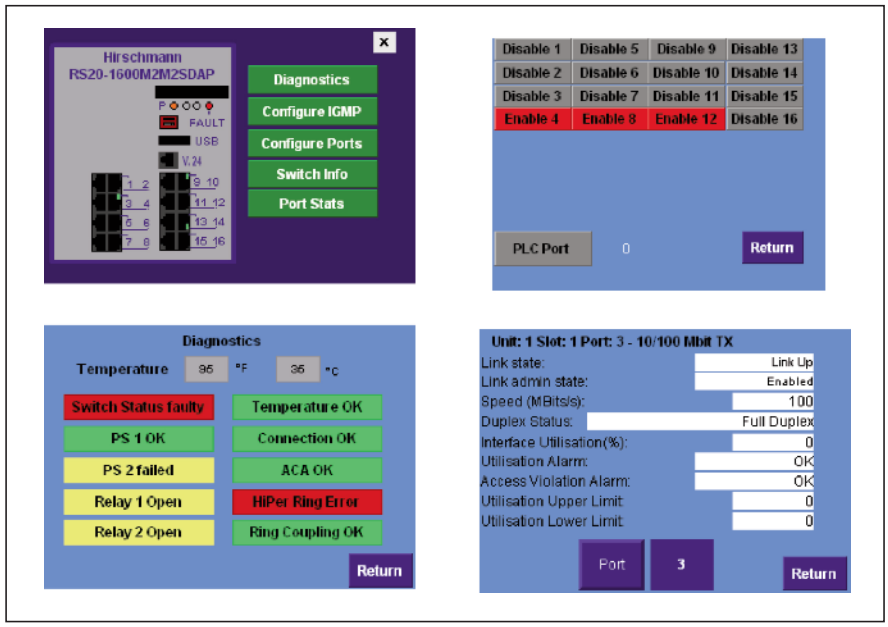

Figure 17 FactoryTalk View global objects provide pre-configured HMI screens

Appendix A

Appendix A – Accessing Tags in a Logix Controller

As mentioned previously, tags are how ControlLogix organizes data. The section "Data Organization in the Controller" discusses the types of tags and resources for where more detailed information can be found.

To access data in the controller CIP services are used with the same symbolic names given to the data in the controller. This is accomplished using one or more ANSI Extended Symbolic Segments to replace the typical CIP address information that uses three Logical Segments (one for Class ID, one for Instance ID and one for Attribute ID). Table 1 shows the encoding of an ANSI Extended Symbolic segment.

Note: For more information on the CIP-defined ANSI Extended Symbolic Segments and all Logical Segments shown here, refer to Appendix C of reference [8].

Table 1 ANSI Extended Symbolic Segment Encoding

Type of Logical Segment	Segment Type	Byte order representation (low byte first)				
ANSI Extended Symbolic		0	1	...	n	n+1
	0x91	length	1st char	...	nth char	*

* optional pad byte (00) when length of string (length) is odd.

When a service references a dimensioned tag (eg: an array), the array members are referenced using a member ID. Table 2 shows the three types of Logical Segments used to to specifiy the array dimensions. There may be one, two or three Member ID segments used to address individual elements of a dimensioned tag; for one, two or three dimension arrays, respectively. Member IDs are also used to reference members of a structured tag.

Table 2 Member ID Logical Segments Used to Access Arrays and Structures in Logix

Type of Logical Segment	Segment Type	Byte order representation (low byte first)				
		0	1	...	n	n+1
8-bit Element ID	0x28	ee	n/a	n/a	n/a	n/a
16-bit Element ID	0x29	00	ee (LSB)	ee (MSB)	n/a	n/a
32-bit Element ID	0x2A	00	ee (LSB)	ee	ee	ee (MSB)

More details of accessing tags can be found in the Logix Data Access Manual, reference [1].

Glossary

Glossary

(Rockwell Automation terms only – for other terms see EtherNet/IP Developers Guide [9])

Add-on Instruction (AOI)
Add-on instructions (or AOIs) are reusable code objects introduced in Rockwell Software RSLogix 5000 Version 16. These instructions enable encapsulation of commonly used logic as sets of reusable instructions. Add-on instructions are created using the Ladder Diagram (Figure 9), Function Block Diagram and Structured Text editors available in RSLogix 5000 software.

Copy File (COP) Instruction
The Copy File (COP) instruction copies the value(s) in the source to the destination. The source remains unchanged.

Message (MSG) Instruction
The message (MSG) instruction asynchronously reads or writes a block of data to another module on a network.

Move (MOV) Instruction
The Move (MOV) instruction copies the source to the destination.

The Source remains unchanged.

User-defined Data Type (UDT)
A UDT is a software structure consisting of different elements and the communication channel.

References

References

[1] Logix5000 Data Access Manual, available from: http://www.rockwellautomation.com/enabled/guides.html

[2] Type Encoding of Logix Structures in CIP Data Table R/W, available from: http://www.rockwellautomation.com/enabled/guides.html

[3] DF1 Protocol and Command Set Reference Manual, available from: http://www.rockwellautomation.com/enabled/guides.html

[4] Delivery of CIP Over RA Serial DF1 Networks, available from: http://www.rockwellautomation.com/enabled/guides.html

[5] Communicating with Rockwell Automation Products Using EtherNet/IP Explicit Messaging, available from: http://www.rockwellautomation.com/enabled/guides.html

[6] I/O Communications with the ControlLogix controller on EtherNet/IP, available from http://www.rockwellautomation.com/enabled/guides.html

[7] Logix5000 Controllers I/O and Tag Data, Publication: 1756-PM004A-EN-P available from: http://literature.rockwellautomation.com/idc/groups/literature/documents/pm/1756-pm004_-en-p.pdf

[8] Volume 1 of the CIP Networks Library, "The Common Industrial Protocol Specification", available from ODVA (http://www.odva.org)

[9] Volume 2 of the CIP Networks Library, "The EtherNet/IP Adaptation of CIP", available from ODVA (http://www.odva.org)

[10] CIP eTraining Classroom CD, available from Rockwell Automation Technologies: http://www.rockwellautomation.com/enabled/cipetraining.html

[11] EtherNet/IP Developers Guide, available from ODVA (http://www.odva.org)

References [1] through [6] can be downloaded for free from the Rockwell Automation Web site; reference [8] and [11] are available from ODVA. Reference [10] can be purchased from Rockwell Automation.

References

Developer Resources
Education

ODVA

- Classroom Training http://www.odva.org
- Essential EtherNet/IP Developer Guide http://www.odva.org
- EtherNet/IP Informational Seminars - Global http://www.odva.org
- EtherNet/IP Library (Publication 100, General Recommendations for EtherNet/IP Developers) http://www.odva.org
- EtherNet/IP Implementors Workshops (advanced topics workshop) – North America / Europe http://www.odva.org

Others

- IXXAT – Classroom Training – Europe http://www.ixxat.de/ethernet_ip-seminar_de,812,147.html
- Pyramid - Training http://www.pyramidsolutions.com/network-connectivity-solutions-and-services.html
- Rockwell Automation – eCIP Training CD http://www.rockwellautomation.com/enabled/cipetraining.html

Implementation

Rockwell Automation recognized Value Added Design Partners with Enabling Technologies

- IXXAT Gmbh - Adapter and Scanner Software, Freescale/Coldfire EtherNet/IP, Netsilicon Net+Arm http://www.ixxat.de
- Nippon System Development Ltd. – Adapter and Scanner Software http://www.nsd.co.jp
- Online Development Inc. – Adapter and Scanner Software http://www.oldi.com
- Pyramid Solutions, Inc. - NetStaX (suite of EtherNet/IP developer kits and tools) http://www.pyramidsolutions.com
- SoftDel Systems, Ltd. – Adapter and Scanner Software, DELCIP middleware http://www.softdel.com
- Softing AG. – Adapter Portable Protocol Software and FPGA-based Hardware Integration http://softing.com

References

CIP Safety Design Assistance and Enabling Technology

- Molex - Woodhead Software and Electronics http://www.woodhead.com

Other Enabling Technologies

- Tata Elxsi – CIP stack implementation http://www.talaelxsi.com
- NetModule – EtherNet/IP Stack for Siemens/NEC ERTEC 200 and 400 Profinet controllers http://www.netmodule.com/en/products/ethip/
- Deutschmann Automation – Unidgate IC http://www.deutschmann.de
- Fieldserver – Protocessor Protocol Modules http://www.protocessor.com
- Grid Connect LX and EX chips http://www.gridconnect.com
- Hilscher – NetX http://www.hilscher.com/netx.html
- HMS – Anybus M, Anybus S, Anybus CompactCom http://www.anybus.com
- HMS – Anybus IC http://www.anybus.com/eng/products/products.asp?PID=90&ProductType=AnyBus-IC
- ODVA – Example Source Code (Public) http://www.odva.org
- Real-Time Automation – EtherNet/IP Modbus/TCP Board Level Gateway http://www.rtaautomation.com
- Real-Time Automation – 510 E Embedded Module and Source Code Stacks http://www.rtaautomation.com
- Smart Network Devices – 4NetOS and HyperstoneOS http://www.smartnd.com

Tools

ODVA

- EZ-EDS (EDS authoring tool) http://www.odva.org
- EDS Authenticator Software Tool http://www.odva.org
- Protocol Conformance Test Software Tool (also check device functionality during development) http://www.odva.org

Others

- Frontline Test Equipment – FTS4Control (Protocol Analyzer) http://www.fte.com/products/FTS4Control-01.asp
- IXXAT – EtherNet/IP Scanner Simulation Tool (EIP Scan) http://www.ixxat.de

References

- Pyramid Solutions, Inc. – EtherNet/IP Scanner Simulation Tool (EIPScan) http://www.pyramidsolutions.com
- Pyramid Solutions, Inc. – EtherNet/IP Device Interoperablity Test Tool (EDITT) http://www.pyramidsolutions.com
- Wireshark – Free Ethernet Analyzer http://www.wireshark.org

Testing

Conformance Test Centers

- ODVA (Michigan / North America) http://www.odva.org
- ASTEM RI (Japan / Asia Pacific) http://securesite.jp/ODVA/english
- University of Magdeburg (Germany / Europe) http://lamp.urz.uni-magdeburg.de/~wiaf/lfp/cvs/home_ethernet_eng.php

Other

- Plug Fests (Interoperability Testing) http://www.odva.org

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

November 2008