Routing of messages between DeviceNet networks and into other CIP networks

Viktor Schiffer, Rockwell Automation Germany

The routing of messages across CIP™[1] networks (CIP = Common Industrial Protocol) is one of the most interesting features of CIP since it allows seamless transition of messages (connected and unconnected) between a DeviceNet™[2] network and other networks of the CIP family, a feature that is not found in any other industrial communication networks.

This contribution explains the following details of the mechanics of the routing process for both connected and unconnected messages:

- **Explanation of the general principle of the routing process**
- **The port object and port segments**
- **Details of unconnected message routing**
- **Details of connected message routing**
- **Execution of the routing**
- **Object addressing and object visibility within routers as seen from multiple networks**
- **Route browsing through multiple networks**
- **Details of the routing within DeviceNet**
- **Route representation with Electronic Data Sheets (EDS)**
- **Bridging into non-CIP networks (last hop)**
- **Real-world example: Full details of a trace of explicit messaging**

The advantage of the described method is the fact that there is only one routing method within all CIP networks (DeviceNet, ControlNet™[3], EtherNet/IP™[4]) and therefore, only minimal translation of messages is required.

Introduction:

CIP has defined mechanisms that allow the transmission of messages across multiple networks, provided the linking devices between the various networks are equipped with a suitable set of capabilities (objects and support of services). This unique feature makes a combination of CIP networks appear as if they were one big network. This paper describes the general principle behind these mechanisms within CIP, how connected and unconnected messaging is set up across multiple networks and what requirements need to be fulfilled within the linking devices. Examples of connected and unconnected messaging are shown and the benefits of CIP routing are discussed. Finally, messaging into non-CIP networks (last hop only) is explained.

General principle:

When messages are transported from one network to another (see Figure 1) linking devices are required between these networks that serve a dual purpose: They carry out the transport between the networks and they may reformat the messages to adapt them to a different data link layer if required. When the linking device connects parts of a network (subnets) that share the same node address space, e.g. DeviceNet to DeviceNet, it is called a bridge. All it has to do is to store the message and forward it on the other side using the bus access

[1] CIP™ is a trademark of ODVA
[2] DeviceNet™ is a trademark of ODVA
[3] ControlNet™ is a trademark of ControlNet International
[4] EtherNet/IP™ is a trademark of ControlNet International under license by ODVA

mechanism of the other network. The message as such does not need any change at all. However, when the networks are dissimilar or when they do not share the same node address range, the linking device may also have to reformat it so that it fits into the transport and network layers of the other network. The linking device is then called a router.

Bridges are not discussed within the scope of this paper. These devices are “invisible” to all CIP messages and therefore do not and cannot modify any of them. From now on, this paper will talk about routers only.

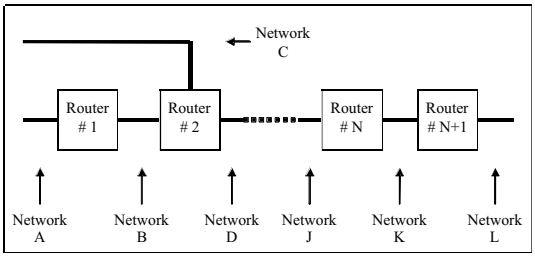

Figure 1: Routing between Networks

Routers may connect multiple networks as shown in Figure 1. Router #2 has three “ports” that link networks B, C and D. Appropriate inclusion of the correct port information into the message that establishes the message transfer is therefore required.

CIP distinguishes between two messaging principles: Connected and unconnected. When routing connected messages, a “highway” is built from the originating node through all routers to the destination node. This “highway” is then used for all messages across this connection for as long as it exists. Due to the permanent and fixed route, messages across this connection can be routed without any overhead and without any path information within the actual message. However, such connections tie up resources within the routers. On the other hand, when routing unconnected messages, only a temporary “road” is built from the originator to the target. This “road” is erected while the message travels from the originator to the target and it is “dismantled” when the response travels back on the return path. Since this “road” only exists until the message has completed, no permanent resources are tied up within the routers. However, the “road” has to be built and taken down with every message sent and this is more time consuming and it also means that routing information must be provided with every message.

Port Objects and Port Segments:

A central detail of all routing within CIP is described through the Port Object. Most CIP devices do not have a need for a Port Object, all it would do is describe the one and only port it has and this would result in largely redundant information. However, as soon as other CIP or non-CIP ports are added to the device, the Port Object provides a standardized way of describing the port information:

- The class attributes specify how many ports there are and they provide a list of their principle characteristics.
- Every port of the device is represented by an instance of the Port Object and the instance attributes give a full description of the properties of the port associated with the instance. The most important details for the routing process are the port type (DeviceNet, ControlNet, EtherNet/IP or other) and the port number.

Navigation through the ports of a device is done with the help of Port Segments. The segment encoding method described in Appendix C of the CIP Specification [1] is a comprehensive addressing method that allows many different types of addresses one of them being the Port Segment described here. A Port Segment is recognized by the three most significant bits of the first segment byte being set to “0”. What a Port Segment essentially does is to describe a “door” (the port) through which to “leave” a device and then where to go next. Therefore all port segments contain a port identifier and a link address, see Figure 2.

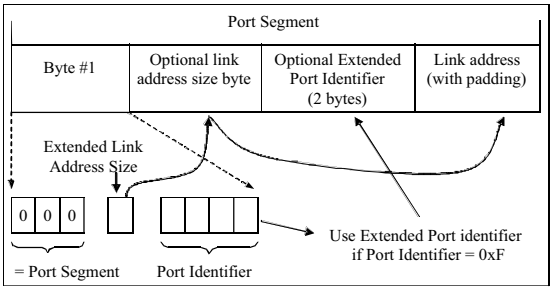

Figure 2: Port Segment Structure

In most cases on DeviceNet and ControlNet, the complete Port Segment will comprise only two bytes with the first byte indicating the port number and the second byte indicating the link address on the network, e.g. [02] [06] means: Go to port #2 and on the network associated with that port go to the device with the node number "6". The Extended Port Identifier is only used when more than 14 ports are available on a device. EtherNet/IP addresses need an extended link address description and this is done by setting the Extended Link Address Size bit, specifying the number of bytes for the link address and specifying the link address through its ASCII representation.

Going to the EtherNet/IP port #3 and then to the device with the IP address 10.71.1.1 would therefore expressed as

[13]…… Port segment, extended link address size, port #3
[09]…… 9 bytes of link address
[31] [30] [2E]
[37] [31] [2E]
[31] [2E] [31]… IP address 10.71.1.1
[00]…… pad byte

If multiple hops are to be described by port segments, the individual segments are simply concatenated. Therefore,

"[13] [09] [31] [30] [2E] [37] [31] [2E] [31] [2E] [31] [00] [02] [06] [05] [20]"

means:

"Go to port #3, then to node number (IP address) 10.71.1.1; within that device, go to port #2, then to node number 6, within that device, go to port #5, and finally go to node number 32.

Messaging details, connected:

Connections across multiple routers are established in pretty much the same way as it is common between two devices on EtherNet/IP and ControlNet; a Forward_Open request (Service Code 0x54) is sent from the originator node to the target node. The only difference is that the Connection_Path information now contains routing information through one or several routers. For full information on the Forward_Open service, see chapter 3-5.5.2 of the CIP Specification [1]. Only the last field of the Forward_Open request has to be extended, all others stay as before. However, it may be necessary to adjust the timing information defined in the first two bytes of the message to accommodate the additional delay of passing through multiple routers and networks.

Figure 3 shows typical data found within the Connection_Path field of a direct Forward_Open message in ControlNet or EtherNet/IP. There is nothing but the connection data for the target node, i.e. segments to describe the location of the configuration assembly and connection points for the I/O data.

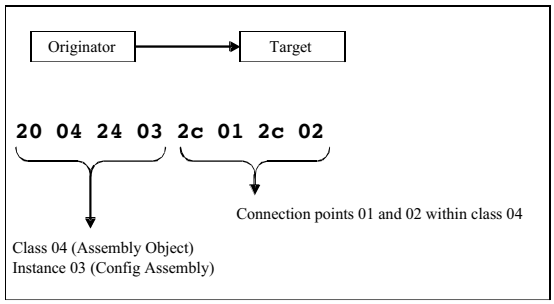

Figure 3: Typical Path Information for a Direct Connection

When a connection is to be established into another network, routing information must be included within the Connection_Path. Therefore, Figure 4 shows the same Connection_Path, but with the inclusion of a Port Segment as described in Figure 2 above. This example

shows a Port Segment for a link with single byte addresses, e.g. DeviceNet or ControlNet.

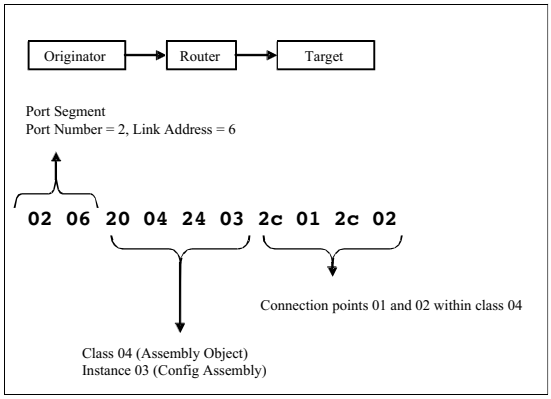

Figure 4: Typical Path Information for a Connection across a Router

When a connection is to be opened across multiple hops, a concatenation of multiple Port Segments must be used, the first Port Segment indicating the first hop. When a Forward_Open request then propagates along the projected path, each router strips off the “consumed” Port Segment until the target network is reached. The number of hops is theoretically unlimited, but in reality it is limited by the amount of data that can be carried in a single Forward_Open request.

A connection established with this method can be used for either Explicit Messaging or I/O Messaging in the same way as it is done within a single network. All connections created with the Forward_Open request exist until they are taken down again with a Forward_Close request (Service Code 0x4E) or are deleted due to other mechanisms, e.g. timeouts. The Vendor ID, Connection Serial Number and Originator Serial Number must match the information used in the Forward_Open request.

Messaging details, unconnected:

Unconnected messaging (for Explicit Messages only) across routers makes use of a somewhat different concept: The transport mechanism is separated from the execution of the actual Explicit Message. This is similar to sending a request in a letter through registered mail. The mail service sends the object (unopened!) across the system, from one postal office to the next until the destination town has been reached where the letter is delivered to the addressee. The addressee (equivalent to the target network) opens the letter and does whatever the content tells him to do. The mailman records the (successful or unsuccessful) delivery and returns a delivery (or non-delivery) receipt plus any other response to the sender.

The service used for this mechanism within CIP is the Unconnected_Send request (Service Code 0x52). An Unconnected_Send request therefore acts like an envelope for the Explicit Message to be delivered and, apart from size indicators and possible padding, contains nothing but timing information and Route_Path information on the “outside” of the envelope. Timing information on the “envelope” is required since every router needs some time to process the request so that the timing for the next hop needs to be adjusted. This process is identical to the timing information processing that is done with Forward_Open messages. The complete structure of an Unconnected_Send request can be found in chapter 3-5.5.4 of the CIP Specification [1].

In contrast to the Forward_Open request, the Route_Path field of an Unconnected_Send request only contains one or several Port Segments and no further addressing segments. As with Forward_Open requests, the number of hops is theoretically unlimited, but in reality it is the overall message length including path segments and request data that will allow a limited number of hops only.

Execution of the routing, connected:

Every Forward_Open request is directed towards an instance (typically #1) of the Connection Manager Object. This object – as its name indicates – is responsible for the connection management. If the object does not find a Port Segment, it executes the Forward_Open service in the local device as requested. If it finds one or several port segments, however, then the

object deletes the first Port Segment and forwards the Forward_Open message to the port/device combination indicated in this segment. Everything else in the Forward_Open request remains unchanged except for the Originator-to-Target Connection ID which may have to be adapted to the characteristics of the network connected to the forwarding port. If an error occurs along the path of the Forward_Open request, an unsuccessful Forward_Open response is returned to the originator. When the complete connection establishment process is successful, a successful Forward_Open response is returned and the connection is then ready to be used. During this process, Originator-to-Target and Target-to-Originator Connection IDs have been created and all routers along the routing path remember these Connection IDs. All the routers then have to do is listening to messages with the established Connection IDs, forward them to the appropriate port and produce them on the other network with the Connection ID assigned for that network. Thus all messages can be transmitted in a most economical fashion.

Execution of the routing, unconnected:

Every Unconnected_Send request is directed towards an instance (typically #1) of the Connection Manager Object. This object – as its name indicates – is responsible for the connection management. If the receiving Connection Manager Object finds only one port segment in the request, it transmits the Unconnected_Send request to the port specified in the port segment and there the embedded Explicit Message is executed with the node indicated in the Port Segment and a response is returned. If it encounters a network that does not support unconnected messaging, e.g. a DeviceNet network, the router creates an Explicit Messaging Connection before executing the embedded Explicit Message. However, if the receiving Connection Manager Object finds two or more Port Segments, then the object deletes the first Port Segment, forwards the Unconnected_Send message to the port/device combination indicated in this segment and remembers the path details for the return message. It also processes and updates the timing parameters. If an error occurs along the path of the Unconnected_Send request, an unsuccessful Unconnected_Send response is returned to the originator. Once the forwarding process is complete and a return message has been generated, the router can release all resources associated with the routed message. It is important to note in this context that the transport mechanism may have been successful in forwarding the message and returning the response, but the response could still contain an indication that the desired service could not be performed successfully in the target device.

Object addressing within routers:

Routers have two or more network interfaces (ports) represented by individual instances of the Port Object (the port numbers do not necessarily align with the instance numbers). All objects that are associated with a network port exist at least once per port. This is so because, from the network point of view, every port is independent and does not have any immediate relationship to the other port(s).

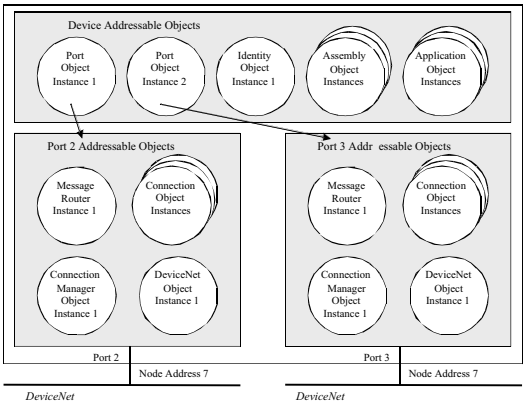

Figure 5: Object Address Space within a CIP Router (DeviceNet to DeviceNet)

As shown in Figure 5, all network related objects are directly visible only on the associated port while the other objects are visible from any port. Any network related objects associated with the other port(s) must therefore be addressed using the Unconnected_Send mechanism with an

appropriate path in place. In order to access instance 1 of the DeviceNet Object in object space of port 3, the Unconnected_Send service of the Connection Manager Object for port 2 needs to be used. The Route_Path parameter within the Unconnected_Send service would contain 03 07 indicating that the request should be routed to port #3 within the device, node #7 on that network. The router must detect that it is node #7 on the network with which port #3 is connected, process the request accordingly, and send a response.

Route browsing:

All routers contain enough information within their Port Objects to allow a tool to find its way through the router into another network. This is so because the class attribute 9 contains a data array of the most important properties of all ports instantiated in the device. Every one of these ports is described in full detail within an instance of the Port Object associated with the port it describes. Using this information, a browser tool can then interrogate the devices on the other network(s) for further information, e.g. to find further CIP routers. Once the tool has identified a particular device on a network that can be reached through the router, it can then use the path information determined through this browsing process and insert it into the Unconnected_Send messages that are required to work with the devices of the target network(s).

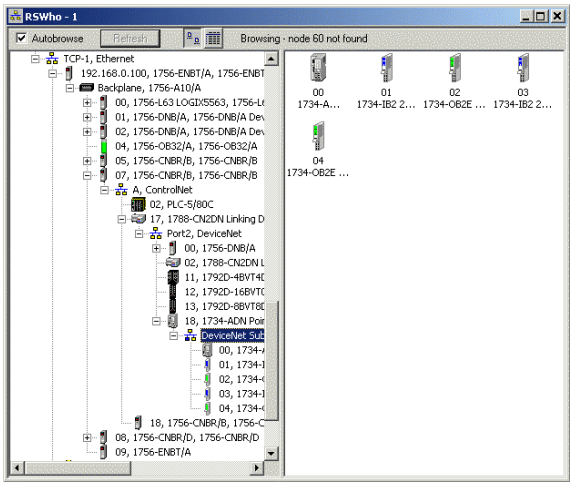

Figure 6: Network Browsing with RSLinx

Figure 6 illustrates the browsing process and how this is displayed through a graphical user interface. The navigation is similar to the Windows Explorer. In this example, the browse tool (RSLinx, Rockwell Automation) first has browsed an EtherNet/IP network and found – among other devices – a Rockwell Automation 1756-ENBT module at IP address 192.168.0.100. Within this device, there is a second port (port #1, “Backplane”), the backplane port of this device. Within the backplane, RSLinx has found a set of devices for possible further browsing, including 1756-DNBs (for DeviceNet) and 1756-CNBRs (for ControlNet). The device through which the tool entered the chassis is also shown (slot #9), but without the “+” sign indicating that there is nothing else to browse. After clicking on the “+” sign of the 1756-CNBR in slot #7, port “A” opens up and indicates the presence of a ControlNet network. After browsing this network, another routing device (node #17, 1788-CN2DN) is found that supports a DeviceNet network on the other side. After browsing this DeviceNet network, the possible target nodes (1734-xx) on this network are identified. The path to these 1734-xx nodes now contains a port segment for the backplane (to node #7), another segment into the ControlNet network (to node #17), then one into the DeviceNet network (to node #18) and finally one to the target device (to node numbers 1 through 4).

Routers and device types:

Being a router does not imply being a device with a particular device type although most of them would likely use the Communications Adapter device profile, in other words, saying you are a CIP router really means you support the CIP routing function. Router devices only have one identity, independent of the port they are accessed through.

Adaptations for DeviceNet:

DeviceNet nodes typically do not contain the Connection Manager Object, but since CIP routing requires this object, all routers on DeviceNet must have the Connection

Manager Object. Due to the somewhat modified Explicit Messaging structure on DeviceNet (abbreviated object addressing) some translation has to take place inside the router. However, the most important adaptation required is the prepending of a Transaction ID field to allow distinction between multiple outstanding Explicit Messages; normal DeviceNet Explicit Messages allow distinction between only two outstanding Explicit Messages. This additional Transaction ID field occupies the first two bytes of the service specific data of the Forward_Open and Unconnected_Send message. By the way, both messages are sent across DeviceNet Explicit Messaging Connections even though "Unconnected_Send" seems to indicate an unconnected message. Due to the length of these two message types, both messages are sent as fragmented messages although the examples in this paper and in the CIP Specification always show them as one entity for clarity reasons.

Another detail on DeviceNet is also worth mentioning: Most DeviceNet devices do not support peer-to-peer I/O Messaging and DeviceNet in general does not support Unconnected Explicit Messaging. Therefore, to accommodate I/O Message routing, a CIP-to-DeviceNet router will first try to use peer-to-peer I/O Messaging, but if that is not successful, I/O Messaging through the Master/Slave Connection Set will be tried. In a similar way, a router will create an Explicit Messaging Connection using the UCCM_Open service or the Master/Slave Connection set or reuse an existing connection for any Unconnected_Send requests to be transmitted across DeviceNet.

A full description of all adaptations necessary for DeviceNet can be found in the DeviceNet Specification [2].

Routing representation in EDSs:

Figure 7 shows the `[Port]` section of a typical router EDS. Routers that have more ports need as many `PortN` entries as there are ports in the device. This example is taken from the Rockwell Automation 1788-CN2DN device shown in Figure 6 above. The port name (Port2) and the port type (DeviceNet) are direct equivalents to what can be found in attributes #4 and #1 of the Port Object instances. The "`N`" of the `PortN` entry corresponds to the instance number of the Port Object. In the case of the 1788-CN2DN, the Port Object has the instances #1 and #2.

```
[Port]
   Port1 =   ControlNet_Redundant,   $ Port type
             "Port1",                $ Port name
             "20 F0 24 01",          $ Path to Port object, The CNet object
             3;                      $ Internal Port Number

   Port2 =   DeviceNet,              $ Port type
             "Port2",                $ Port name
             "20 03 24 01",          $ Path to Port object, The DNet object
             2;                      $ Internal Port Number
```

Figure 7: Typical `[Port]` Section in a Router EDS

Non-CIP network on the last hop:

When the last hop goes into a non-CIP network, routing of I/O and Explicit Messages is still possible, but whether these functions can be supported or not is largely dependant on the characteristics of the target network. The following details have to be considered:

- The linking device between the two networks should be capable of performing the routing of all messaging requests coming from the CIP side. These messages are forwarded to the individual devices on the target networks. Using this concept, the request for a certain piece of data within a target device will be answered by this device, not by a proxy object within the router that may contain old data or that still needs to forward the data to the target device.
- With a suitable router, devices on the non-CIP target network will look and behave pretty much like CIP devices, but they may not support the minimum CIP object model due to different data structures in the non-CIP target devices.
- Whether such a router can also route requests from the non-CIP network into the CIP world will largely depend on what services and capabilities exist in the target network; they may not be as universal and powerful as those within CIP.

- While node addressing mechanisms within CIP are flexible enough to even accommodate addresses as complex as an IP address, object addressing (or whatever equivalent the target network uses) may have to be translated in the router. The popular method of using an index (16 bit) + subindex (8 bit) addressing (e.g. on CANopen and Interbus-S) is not a problem, this can easily be represented as a vendor specific object that is addressed in a target device and the router performs the translation into the native target network addressing scheme.
- CIP Service Codes can be reused as long as the target network has services that are a 100% equivalent of a CIP service. Other services of the target network have to be represented by vendor-specific service codes that the router translates into the native target network services.
- To what extent both I/O and Explicit Message routing can be accomplished, depends on the message type structure of the target network. In many cases, a proper routing will only be possible for configuration data and diagnostic parameters through Explicit Messaging.

A typical example of how non-CIP message routing can be accomplished is the work that is currently being done in the JSIG (Joint Special Interest Group) “Distributed Motion” of the ODVA that is working on defining an open CIP-to-SERCOS routing functionality. Once their work is finished, this can be used as a “template” for other non-CIP networks.

Advantages of the CIP routing principles:

The advantage of the described method is that no router needs to know anything about the messages to be routed ahead of time. Any configuration that is required comes either with the Forward_Open or the Unconnected_Send requests. Therefore, no router must be configured ahead of time and as a consequence, no configuration is required at any time, not even when a defective router must be replaced. This is why it is called seamless routing. The message originator only has to know the system network topology to specify the proper hops. From a user’s point of view, this is very simple to do since he can determine the desired paths through route browsing without the need to manually enter the path.

APPENDIX: Real world example: Explicit messaging from EtherNet/IP to a DeviceNet node

The following example shows the messages that are exchanged between an EtherNet/IP messaging client (RSNetworx for DeviceNet) and a DeviceNet target device reached through an EtherNet/IP-to-EtherNet/IP router, an EtherNet/IP-to-DeviceNet router and a DeviceNet-to-DeviceNet router. The EtherNet/IP-to-DeviceNet router is made up of two devices connected via a backplane; therefore, another Port Segment is required. The following addresses are used:

EtherNet/IP messaging client	10.71.130.117
EtherNet/IP-to-Backplane router, EtherNet/IP side	192.168.0.100
EtherNet/IP-to-Backplane router, Backplane side:	9
Backplane-to-DeviceNet router, Backplane side:	1
Backplane-to-DeviceNet router, DeviceNet side:	0
DeviceNet-to-DeviceNet router, primary side:	5
DeviceNet-to-DeviceNet router, secondary side:	0
DeviceNet target device:	1

The example shows only the relevant messages that are exchanged, any messages required to create connections are dropped for clarity. Processing of the timeout information within the Unconnected_Send message is not discussed in this context.

The Unconnected_Send message of the EtherNet/IP messaging client is sent as SendRRData request (command code 0x006F) across a previously established EtherNet/IP session. The details of this EtherNet frame are explained in Figure A 1. Full details of the EtherNet/IP message encoding can be found in the EtherNet/IP Specification [3].

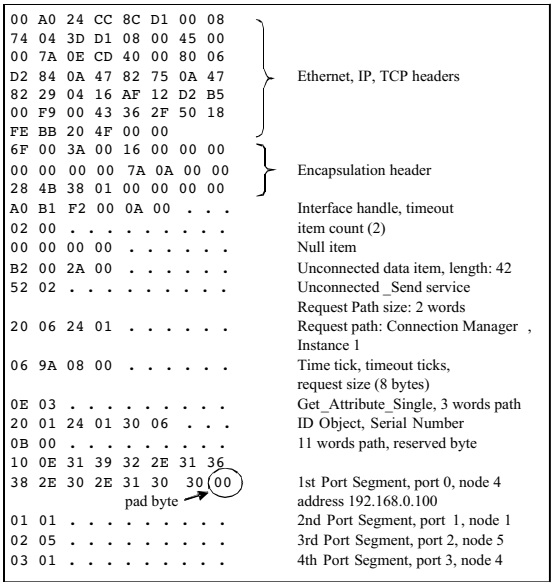

Figure A 1: Unconnected_Send Request across EtherNet/IP

The message shown in Figure A 1 is what is sent by the requesting application. It first passes through the EtherNet/IP-to-EtherNet/IP router (the next EtherNet/IP node is on a different network) before it is delivered to the EtherNet/IP-to-DeviceNet router. This routing process only removes the first Port Segment and updates the timing information.

This message is then translated by the EtherNet/IP-to-DeviceNet router as follows:

- Since the EtherNet/IP-to-DeviceNet router consists of two routers (EtherNet/IP-to-backplane and backplane-to-DeviceNet), it “consumes” the next two Port Segments.
- The message is also translated into the DeviceNet format.

The resulting message on DeviceNet is shown in Figure A 2; the message header, the fragmentation and fragmentation acknowledgements are omitted for clarity.

52	Unconnected_ Send service
06 00 01	Path to Connection Manager, instance 1 (format 16/8)
01 00	Transaction ID
06 90 08 00	Time tick (unchanged), timeout ticks (reduc ed from 0x9A to 0x90), request size
0E 03	Get_Attribute_Single, 3 words path
20 01 24 01 30 06 .	ID Object, Serial N umber
01 00	1 word path , reserved byte
03 01	remaining Port Segment

Figure A 2: Unconnected_Send Request on DeviceNet

Finally, this message is translated once again in the DeviceNet-to-DeviceNet router. This router recognizes that the message has now reached the target network and so it executes the explicit message "wrapped" into the Unconnected_Send message and the result is the well familiar Get_Attribute_Single request/response messages on DeviceNet as shown in Figure A 3 (message headers left intact here, messages using the Master/Slave Connection Set).

00 0E 01 00 01 06 . .	Get_Attribute_Single request from ID O bject, instance 1, attribute 6 (Serial Number), 16/8 format
00 8E B7 52 0A 1A . .	Get_Attribute_Single success response with Serial Number 0x1A0A52B7

Figure A 3: Explicit Messaging Request/Response on DeviceNet Target Network

The response now travels back through the DeviceNet-to-DeviceNet router and takes the shape of an Unconnected_Send response now as shown in Figure A 4.

D2	Unconnected _Send response
01 00	DeviceNet Transaction ID
00 00	General Status (0 = success), reserved byte
B7 52 0A 1A	Serial Number (0x1A0A52B7)

Figure A 4: Unconnected_Send Response across DeviceNet

Back at the originating device, after traveling back through the EtherNet/IP-to-DeviceNet router and the EtherNet/IP-to-EtherNet/IP router it finally takes the shape of the message that is shown in Figure A 5.

00 08 74 04 3D D1 00 A0 24 CC 8C D1 08 00 45 00 00 58 51 B8 40 00 80 06 8F BB 0A 47 82 29 0A 47 82 75 AF 12 04 16 00 43 36 2F D2 B5 01 4B 50 18 1C A6 9C B6 00 00	Ethernet, IP, TCP headers
6F 00 18 00 16 00 00 00 00 00 00 00 7A 0A 00 00 28 4B 38 01 00 00 00 00	Encapsulation header
A0 B1 F2 00 0A 00 . . .	Interface handle, timeout
02 00	Item count (2)
00 00 00 00	Null item
B2 00 08 00	Unconnected data item, length: 8
8E	Get_Attribute_Single response
00 00 00	Reserved byte, general status (0 = success), reserved byte
B7 52 0A 1A	Serial Number (0x1A0A52B7)

Figure A 5: Unconnected_Send Response across EtherNet/IP

In this example, the router has chosen to return "8E" (Get_Attribute_Single response) as service code. The desired data (Serial Number 0x1A0A52B7) is then displayed to the user of RSNetworx for DeviceNet and the process has completed successfully.

REFERENCES:

[1] CIP Common Specification, Edition 2.0, December 15, 2003, ODVA.

[2] DeviceNet Adaptation of CIP Specification, Edition 1.0, December 15, 2003, ODVA.

[3] EtherNet/IP Adaptation of CIP Specification, Edition 1.0, June 5, 2001, ODVA.

Viktor Schiffer
Rockwell Automation Germany
Düsselberger Str. 15
42781 Haan
Germany
Phone: +49-2104-960-193
Fax: +49-2104-960-197
Email: vschiffer@ra.rockwell.com
Website:
http://www.automation.rockwell.com