Delivery of CIP Over RA Serial DF1 Links

(Rev. 1.1, 12-Oct-06)

Delivery of CIP Over RA Serial DF1 Links

Purpose

This document describes the use of PCCC commands 0Ah and 0Bh to support CIP connected and unconnected explicit messaging over a Serial DF1 link (e.g RS-232 or RS-485). It also describes the PCCC Fragmentation protocol used to transfer messages larger than the PCCC 244 byte limit.

Note: **“CIP”** refers to the **“Common Industrial Protocol”** shared between ControlNet, DeviceNet, and EtherNet/IP, and is documented in the **"CIP Standard" (**CIP Networks Library (Volume 1: Common Industrial Protocol). Also, the term "IOI" and "EPATH" are interchangeable.

Note: **“PCCC”** refers to the **“Programmable Controller Communication Commands”** as described in the **"DF1 Protocol and Command Set Manual”**, publication 1770-6.5.16, and the **“Logix Data Access Reference Manual”**, publication 1756-RM005A-EN-E. Both are available for download from the www.ab.com website. When Rockwell Automation (RA) defined its DF1 serial protocol, it was limited to the PCCC application protocol. Therefore, when RA developed the CIP application protocol, it encapsulated this in PCCC.

Appendices

CIP Communication Architecture

The Open System Interconnect (OSI) reference model from the International Standards Organization (ISO) is used to help structure this document. The table below shows how the layers from the model are discussed in this document.

Table 1 Open System Interconnect (OSI) Layers

This OSI layer:	with this OSI name:	is discussed under:	Function
7	Application	Messaging	Message format and meaning
6	Presentation		
5	Session		not used in CIP
4	Transport	Transport	End to end data integrity
3	Network	Routing	Determining the path from originator to destination
2	Data Link	not discussed	Rules for accessing a single link
1	Physical	not discussed	Rules for encoding bits on the media

The CIP networks share a common understanding of the layers above the Data Link Layer.

Messaging

RA Logix products have two application layer (messaging) protocols. These are:

- CIP Connected Messages (refer to “Messaging”). This is the preferred messaging system for CIP products.

- PCCC. This protocol eases Logix connection to RA legacy networks (DH, DH+, Serial DF1 and DH-485) and continues as a Logix Serial DF1interface standard. The CIP extensions to PCCC are specific to Rockwell Automation and are not part of the “open” CIP Standard.

These are both request/response protocols, with exactly one response expected for each request.

Transport

CIP defines a set of transport protocol classes. Each of these classes defines a header for the PDU and appropriate state machines. Each class provides a different capability. The application chooses which class to use for that application. Refer to CIP Standard for standard transport protocols.

Routing

CIP generally uses connections for moving data and messages, although unconnected messages are also supported. Connected messages use a Connection ID (implicit address) and unconnected messages use an IOI (Internal Object Identifier, or EPATH) to explicitly identify the route and target object. These connections are opened by one end–point of the connection. The connection originator specifies the route the connection must take. These connections can be:

- point–to–point or multicast;

- unidirectional or bi-directional.

Interface modules, which connect to non-CIP networks, must know how to route messages from other network types. This information is presented in a “routing table”. The ControlLogix chassis holds communication interfaces to each network type. Each communication module is a “half–gateway” which translates between the native network and the ControlLogix backplane protocol. This allows each communication module to have a small number of translations rather than trying to support *n* translations, one for each other network type.

Delivery of CIP in PCCC

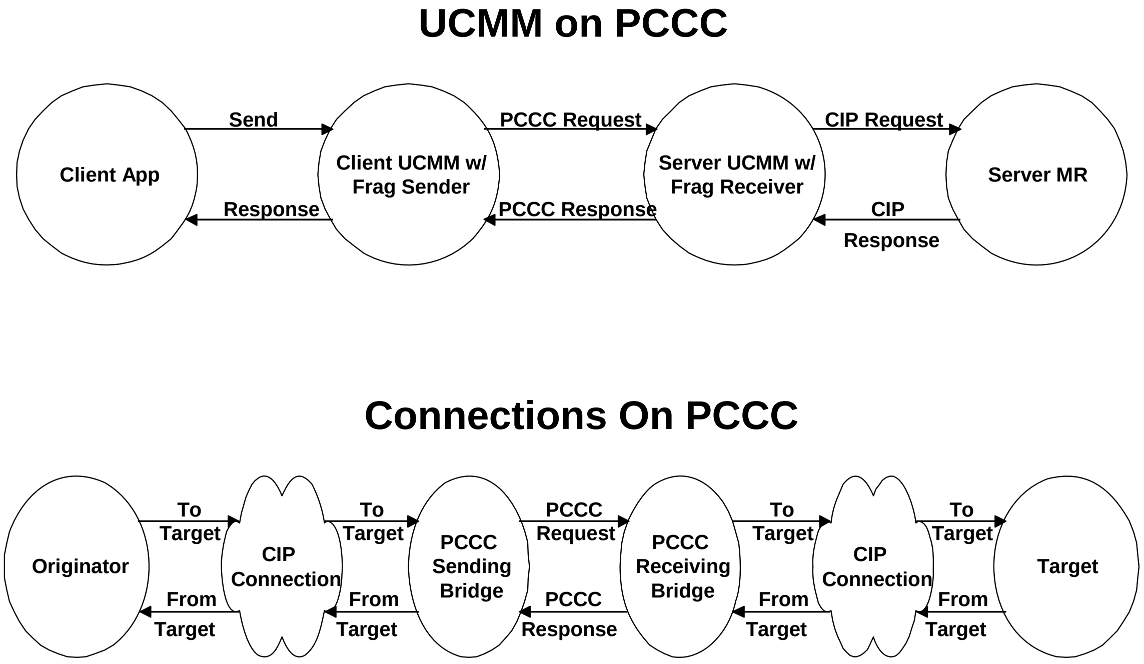
The delivery of CIP messages across existing PCCC-based link types encapsulates CIP information inside of PCCC messages. The PCCC messaging system is used to provide two different components of the CIP communication structure:

- UCMM – The CIP UnConnected Messaging Manager
- Network Connection Pair

Connected messages have the advantage of reserving a "channel" (some buffer space) for communications, where delivery of unconnected messages may depend more on the network traffic to that node.

These are illustrated in the following diagram.

Figure 1 CIP within PCCC



PCCC as an UnConnected Messaging Manager (UCMM)

PCCC Command Code $0B_{hex}$ provides CIP UCMM behavior. The CIP Standard defines the CIP UCMM. The UCMM is directly attached to the Message Router. Therefore, the contents of this PCCC message is a CIP Service request. The CIP Service response is returned via the UCMM, which is the PCCC response. Uses of the embedded CIP would be

a) Unconnected native communication to a CIP device (e.g. read/write ControlLogix Data Table Tags)
b) Make a connection request to a CIP device (for connected communication)
c) deliver an embedded PCCC command (e.g. Type R/W of "mapped" data in ControlLogix); in this case a PCCC command contains a CIP command which in turn contains another PCCC command.

The following table defines the fields for this PCCC message:

Table 2 PCCC Command 0B Structure

Name	Type	Description of Request Parameter	Semantics of Values
CMD	USINT	Command = $0B_{hex}$	
STS	USINT	Status (0 in request)	
TNSW	UINT	Used to match response with request	
FNC	USINT	Fragmentation protocol function	* Fragmentation Protocol
Extra	USINT	Additional information for fragmentation protocol	* Fragmentation Protocol
Service*****	USINT	CIP Service Code	Refer to CIP Standard
Size of IOI***** (EPATH)	USINT	Number of UINT's in IOI	Refer to CIP Standard
IOI***** (EPATH)	Array of UINT	Internal Object Identifier or Path	Refer to CIP Standard
Parameters	Object and Service specific	Parameters for this service for this object	Refer to CIP Standard

***** These only appear in the First_Req., First_Response and Only messages.of Fragmentation Protocol

The size of the message, from the FNC to the Parameters, inclusive, is limited by the PCCC specification and must be less than or equal to 244 bytes. This is enforced by the UCMM implementation. If an application tries to send a message over the UCMM larger than 244 bytes, an error is returned to the local application.

The UCMM requires definition of the following items:

- Retry Time
- Transaction ID
- Request/Response
- Datagram/ReqRsp
- Priority

These are specified in the PCCC implementation as follows:

- **Retry Time** – This is provided by the PCCC delivery subsystem.
- **Transaction ID –** This is provided by the PCCC TNSW.
- **Request/Response** – This is provided by bit 6 of the CMD field of the PCCC message.
- **Datagram/ReqRsp** – This is **not** provided by this PCCC implementation. This UCMM always provides Request/Response messaging.
- **Priority** – This is provided by bit 5 of the CMD field of the PCCC message.

See ”Formal model”, unconnected sender and unconnected receiver, for the formal specification of the fragmentation protocol provided for CMD $0B_{hex}$.

PCCC as Network Connection Pair

PCCC Command Code $0A_{hex}$ provides CIP Network Connection behavior. See “Network Connections” for the definition of CIP Network Connections. A CIP Network Connection provides data movement in one direction across a link. Connection would be established by sending the 0B command first. In many cases, a pair of Network Connections is opened via the Connection Manager with one OPEN request. The PCCC request and response provides this pair of Network Connections. The following table shows the fields used:

Table 3 PCCC Command 0A Structure

Name	Type	Description of Request Parameter	Semantics of Values
CMD	USINT	Command = $0A_{hex}$	See DF1 publication
STS	USINT	Status (0 in request)	See DF1 publication
TNSW	UINT	Transaction sequence number	See DF1 publication
FNC	USINT	Fragmentation protocol function	* Fragmentation Protocol
Extra	USINT	Additional information for fragmentation protocol	* Fragmentation Protocol
CID*****	UINT	Connection ID	Refer to CIP Standard
Trans. Header*****	Transport Class Specific	Transport Header	Refer to CIP Standard
Data	ARRAY of USINT	Data transferred via this connection	Refer to CIP Standard
*****These only appear in the First_req., First_response, and Only messages			

This Connection ID may have a different value than the Connection ID of a different segment along the path taken by the connection. The Connection ID is the same for **all** messages over this CIP Network Connection. The contents of the data field are defined by the object, which is the target of the connection.

The target application of this protocol is to support Transport Class 3 connections to the CIP Message router. It is improbable that the defined protocol will work for other transport classes, since PCCC has a strict single request to single response mapping. Therefore, those transport classes that do not have exactly one response for each request cannot be supported. The PCCC “Request” will be used to deliver data from the Client side to the Server side of the connection, and the PCCC “Response” will be used to deliver data from the Server side to the Client side of the connection.

The PCCC status in the PCCC response indicates the success or failure of the PCCC system to deliver the data across the PCCC link. It does **not** indicate the success or failure of the CIP service that might have been requested. The data field will contain the CIP service status.

The following three diagrams review how CIP connection segments are used and the structure of Originators and Targets.

Keeping a Connection Alive

"Keep-alive" messages are sent over a connection to keep it from timing out during idle periods. If an originator has sporadic use of a connection, it should send "keep-alive" messages if it wants the connection to stay alive during the time when it does want to use it. Normally this involves re-sending the last message sent with the **same** sequence count. This is a "less expensive" use of CPU cycles, since the target will not reprocess the whole message; it just resends the same response already sent (it looks to the responder like a retry due to a lost response). The connection timeout should be short enough that recovery from noise or a temporary disconnect is relatively quick. It is better to scale the timeout to something reasonable for noise recovery (20 seconds?) and then use keep-alive messages.

Figure 2 How CIP connection segments are used.

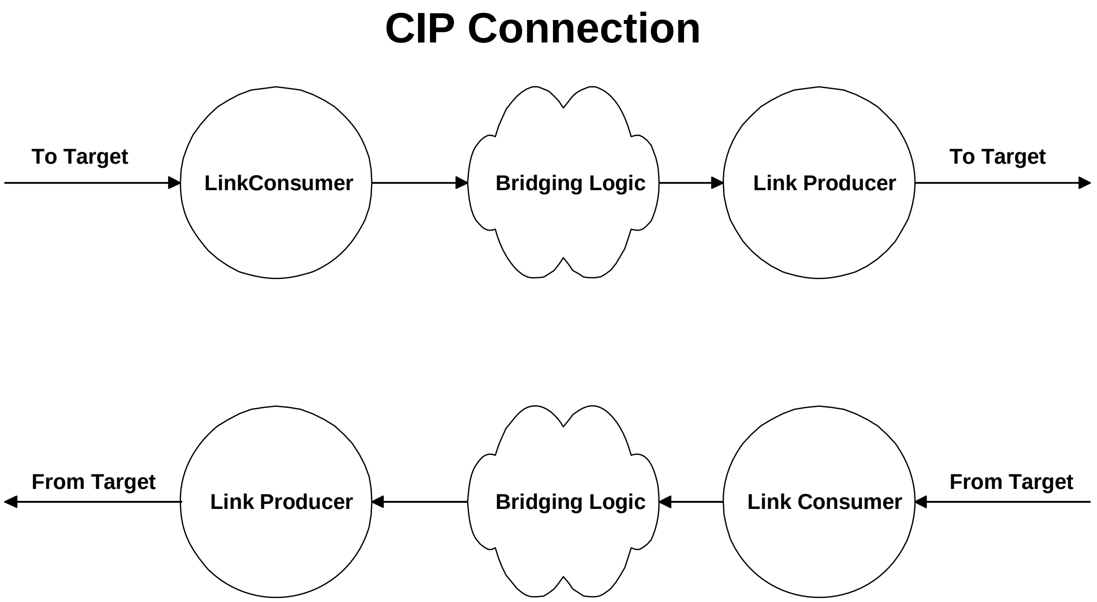


Figure 3 Structure of Originator

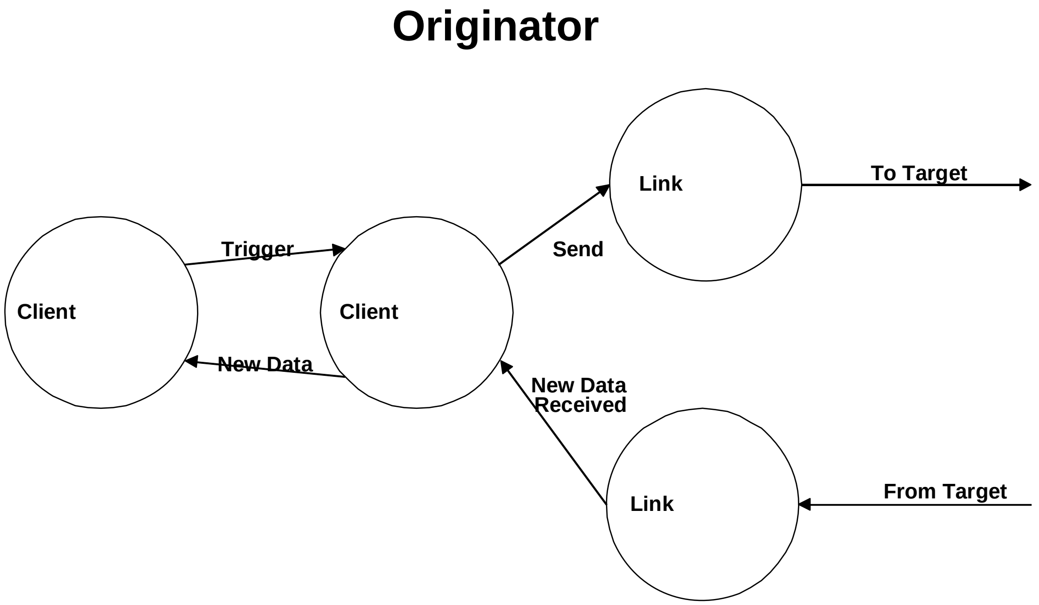
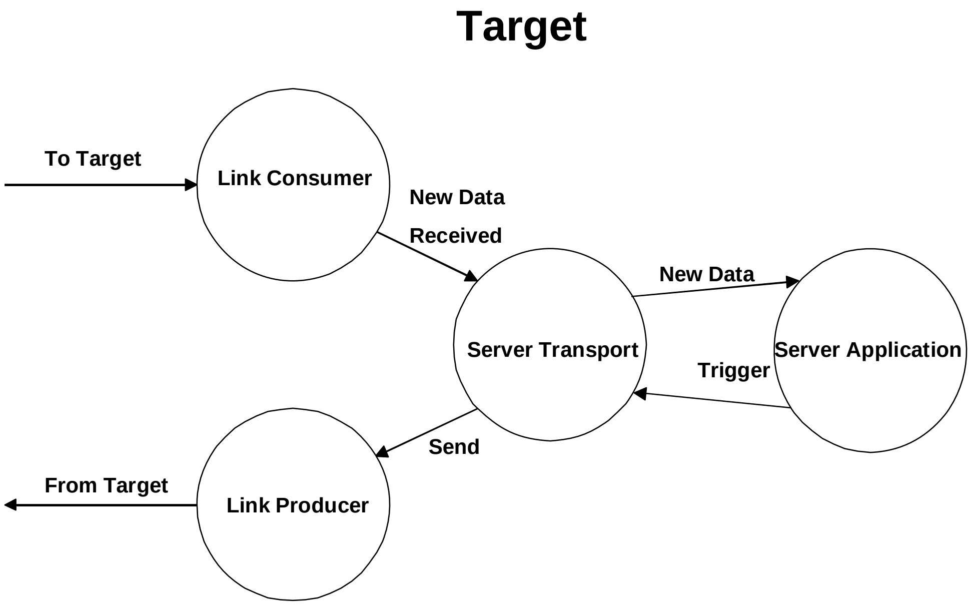


Figure 4 Structure of Target



The next two diagrams show the structure of a CIP Bridge that has a PCCC link attached to it. In particular note where the fragmentation state machines fit the structure of a bridge; they are between the CIP ”link” components (Producer and Consumer) and the PCCC link. This allows the bridge logic to follow the normal CIP behavior, and localizes the impact of the PCCC link specifics to the link components.

Figure 5 Structure of CIP Sending Bridge with a PCCC link attached

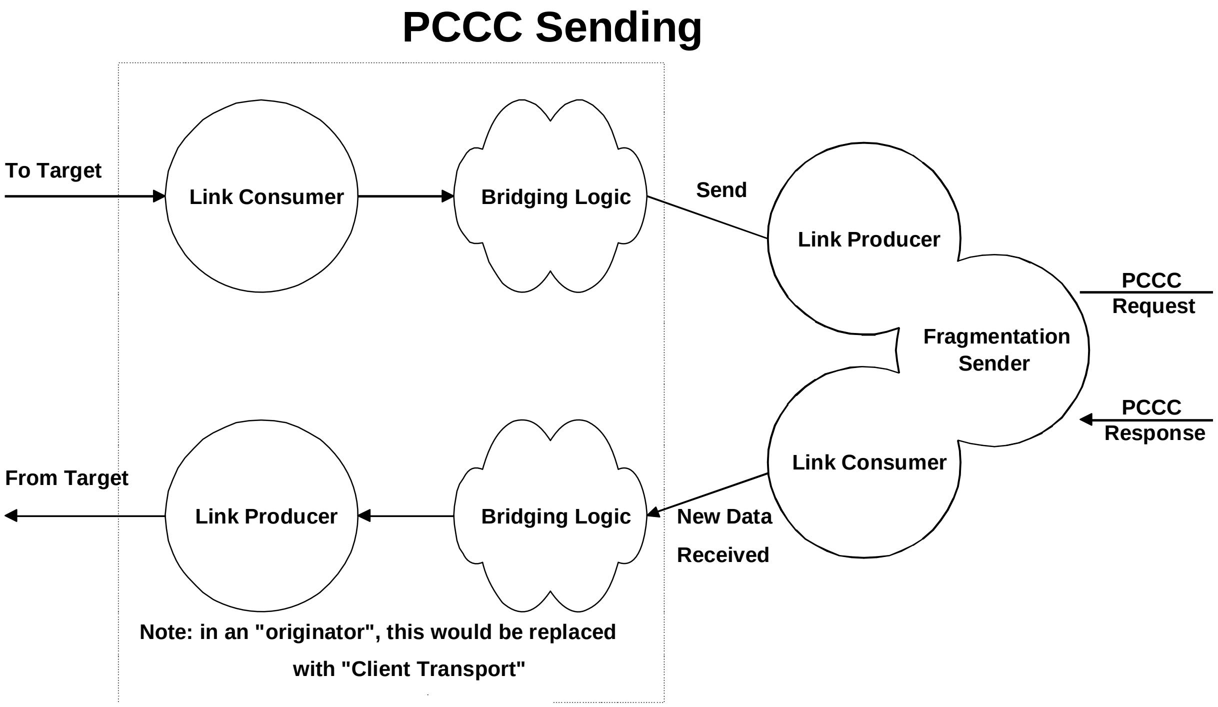
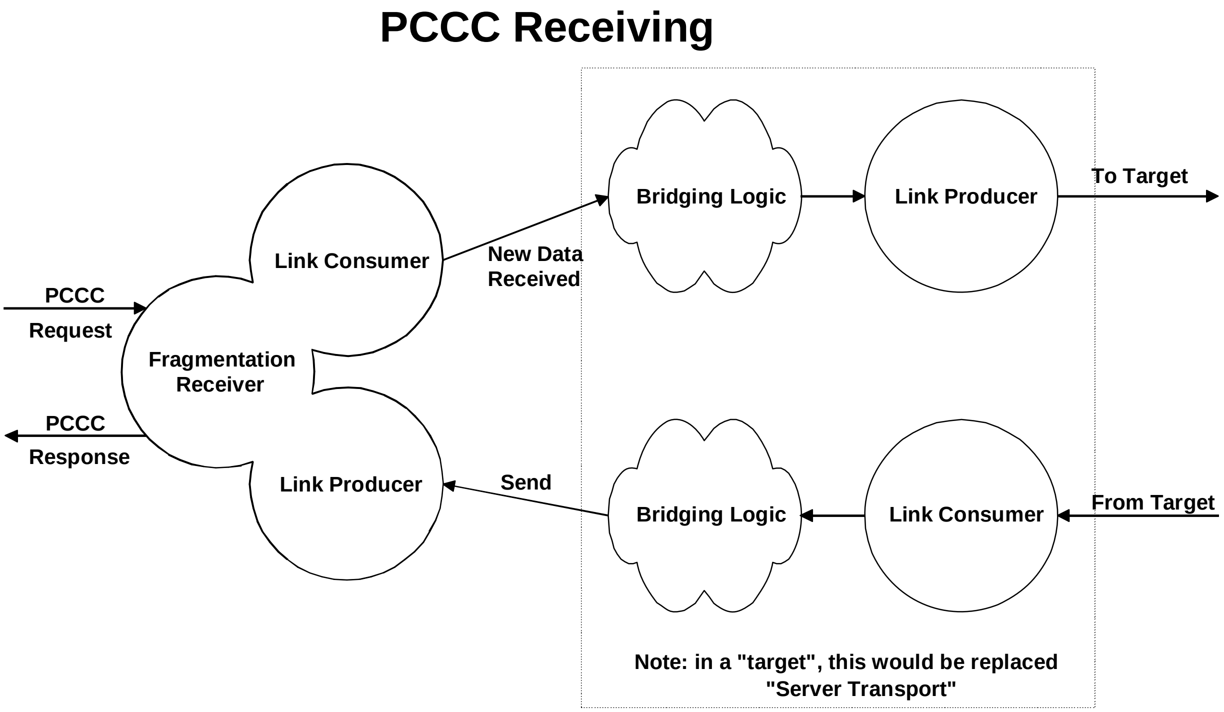


Figure 6 Structure of CIP Receiving Bridge with a PCCC link attached.



The following is an example of the fields for a CIP Messaging connection using a Class 3 Transport encapsulated in PCCC using Serial DF1 Full Duplex:

Table 4 Example of CIP in PCCC on DF 1

Name	Type	Description of Request Parameter	Semantics of Values
DLE	USINT	ASCII escape character	See DF1 Publication
STX	USINT	”Start of message”	See DF1 Publication
DST	USINT	Address of destination	See DF1 Publication
SRC	USINT	Address of source	See DF1 Publication
CMD	USINT	Command = $0A_{hex}$ (CIP Connected message)	See DF1 Publication
STS	USINT	Status (0 in request)	See DF1 Publication
TNSW	UINT	Transaction sequence number	See DF1 Publication
FNC	USINT	Fragmentation protocol function	Fragmentation Protocol
Extra	USINT	Additional information for fragmentation protocol	Fragmentation Protocol
CID	UINT	Connection ID for delivering CIP Responses	See CIP Standard
Trans. Header	UINT	Class 3 Transport Header: 16 bit Sequence Count	See CIP Standard
Service	USINT	CIPService Code	See CIP Standard
Size of IOI (EPATH)	USINT	Number of USINT’s in IOI	See CIP Standard
IOI (EPATH)	Array of USINT	Internal Object Identifier or Path	See CIP Standard
Parameters	Object and Service specific	Parameters for this service for this object	See CIP Standard
DLE	USINT	ASCII escape character	See DF1 Publication
ETX	USINT	“End of message”	See DF1 Publication
BCC or CRC	USINT or UINT	Block Check Character or Cyclic Redundancy Check	See DF1 Publication

Delivery of PCCC in CIP

It’s also possible for the PCCC messaging protocol to be delivered across the CIP system by establishing a connection to the Message Router object and using an IOI to specify the PCCC object (67h). The PCCC command is processed by the "Execute PCCC" (4Bh) service.

Table 5 PCCC Object Services

Size	Function
Variable (typically 8 or 9 bytes)	Requestor Identification
4 to 248 bytes	PCCC command (starting with CMD byte)

The following table shows the structure of the Requestor Identification for the Execute PCCC service:

Table 6 Requestor ID for Execute PCCC Service

Name	Type	Description of Request Parameter	Semantics of Values
Length	USINT	Length of Requestor Identification	In USINT’s, includes Length, Vendor, Ser. No., and Other
Vendor	UINT	Vendor number of requestor	See CIP Standard
Serial Number	UDINT	CIP Serial number of requestor	See CIP Standard
Other	ARRAY of USINT	Identifier of user, task, etc. on the requestor	Requestor specific, typically 2 USINT’s

Example: Execute_PCCC service over CIP Class 3 Transport

Following is an example of the bytes for the Execute_PCCC service request over (within) a CIP Class 3 transport. It does not include the link layer bytes, since they vary from one link type to another.

Table 7 Execute PCCC Over Class 3 Transport

Name	Type	Description of Request Parameter	Semantics of Values
Trans. Header	UINT	Class 3 Transport Header: 16 bit Sequence Count	See CIP Standard
Service	USINT	CIP Service Code for Execute_PCCC service	4Bh
Size of IOI (EPATH)	USINT	Number of UINT's in IOI	See Identity object specification in CIP Standard

IOI (EPATH)	Array of USINT	Internal Object Identifier:
		Class = PCCC Object (67h)
		Instance = 1
			See Identity object specification in CIP Standard

Length	USINT	Length of Requestor Identification	In USINT's, includes Length, Vendor, Ser. No., and Other
Vendor	UINT	Vendor number of requestor	See Identity object specification in CIP Standard.
Serial Number	UDINT	CIP Serial number of requestor	See Identity object specification, in CIP Standard
Other	ARRAY of USINT	Identifier of user, task, etc. on the requestor	Requestor specific, typically 2 USINT's
CMD	USINT	Command code	See DF1 publication
STS	USINT	Status (0 in request)	See DF1 publication
TNSW	UINT	Transaction word	See DF1 publication
PCCC Data	ARRAY of USINT	Parameters for this service	See DF1 publication

Appendix A: Summary of PCCC Commands 0Ah and OBh.

CIP Connected Message (PCCC CMD 0Ah)

- CMD Byte - 0Ah
- Command Parameters -
    - Fragmentation Header - Two bytes, Function first, "extra" second
    - Connection ID - Two bytes, low byte first
    - Transport Header - <size depends on transport class>

      For Transport classes 0-3, this is a two byte sequence count, low byte first.
    - Data - <variable size>
- Possible Responses -
    - STS = 00H - Success, with the following:
        - Fragmentation Header - Two bytes, Function first, "extra" second
        - Connection ID - Two bytes, low byte first
        - Transport Header - <size depends on transport class>

          For Transport classes 0-3, this is a two byte sequence count, low byte first.
        - Response data - <variable size>
    - NOTE: Only STS is supported; there is NOT any EXT STS for this command.
- Operation -

This command is used to provide CIP Network Connection behavior. A CIP Network Connection provides data movement in one direction across a series of links. In many cases, a pair of Network Connections are opened via the Connection Manager with one OPEN request. The PCCC request and response provides this pair of Network Connections.

- Command format diagram

```
        COMMAND BLOCK                                RESPONSE BLOCK

-------------------------             -------------------------
|  Frag Function Code   |             |   Frag Function Code  |
-------------------------             -------------------------
|"Extra" depends on func|             |"Extra" depends on func|
-------------------------             -------------------------
|  Connection ID (low)  |             |  Connection ID (low)  |
-------------------------             -------------------------
|  Connection ID (high) |             |  Connection ID (high) |
-------------------------             -------------------------
|Transport Header (1st) |             |Transport Header (1st) |
-------------------------             -------------------------
|          :            |             |          :            |
-------------------------             -------------------------
|Transport Header (last)|             |Transport Header (last)|
-------------------------             -------------------------
| Connection Data (1st) |             | Connection Data (1st) |
-------------------------             -------------------------
|          :            |             |          :            |
-------------------------             -------------------------
| Connection Data (last)|             | Connection Data (last)|
-------------------------             -------------------------
```

```
Example using a class 3 Transport connection to the CIP Message Router.

        COMMAND BLOCK                                RESPONSE BLOCK
-------------------------                     -------------------------
| 00 (means "Only")     |                     | 00 (means "Only")     |
-------------------------                     -------------------------
| 00 (Only has no extra)|                     | 00 (Only has no extra)|
-------------------------                     -------------------------
|  Connection ID (low)  |                     |  Connection ID (low)  |
-------------------------                     -------------------------
|  Connection ID (high) |                     |  Connection ID (high) |
-------------------------                     -------------------------
| Trans Seq Count (low) |                     | Trans Seq Count (low) |
-------------------------                     -------------------------
| Trans Seq Count (hi)  |                     | Trans Seq Count (hi)  |
-------------------------                     -------------------------
|     Service           |                     | Service w/Reply bit   |
-------------------------                     -------------------------
|     Size of IOI       |                     |          0            |
-------------------------                     -------------------------
|    IOI (1st Byte)     |                     | CIP Status            |
-------------------------                     -------------------------
|          :            |                     | Size of CIP Ext status|
-------------------------                     -------------------------
|    IOI (Last Byte)    |                     | CIP Ext status (1st)  |
-------------------------                     -------------------------
| Parameters (1st byte) |                     |          :            |
-------------------------                     -------------------------
|          :            |                     | CIP Ext status (last) |
-------------------------                     -------------------------
| Parameters (last byte)|                     | Response Data (1st)   |
-------------------------                     -------------------------
                                              |          :            |
                                              -------------------------
                                              | Response Data (last)  |
                                              -------------------------
```

CIP Unconnected Message (PCCC CMD 0BH)

- CMD Byte - 0BH
- Command Parameters -
    - Fragmentation Header - Two bytes, Function first, "extra" second
    - Service - one byte
    - Size of IOI - in words, one byte
    - IOI - an even number of words; zero words is valid.
    - CIP Service Parameters - as defined by the object/service
- Possible Responses -
    - STS = 00H - Success, with the following:
        - Fragmentation Header - Two bytes, Function first, "extra" second
        - Service - one byte, with "response" bit now set
        - CIP Status - one byte
        - Pad byte - one byte,
        - Size of extended status - one byte, in words
        - Extended status - "size of extended status" number of words
        - CIP response data
    - NOTE: Only STS is supported; there is NOT any EXT STS for this command.
- Operation -

This command is used to provide CIP Unconnected Message Manager (UCMM) behavior. (See the CIP Standard for a definition of the UCMM.) The CIP UCMM is directly attached to the Message Router Object. Therefore, the content of this PCCC message is a CIP Service request. The CIP Service response is returned via the UCMM (which is the PCCC response).

- Command format diagram

```
        COMMAND BLOCK                        RESPONSE BLOCK

-------------------------             -------------------------
|  Frag Function Code   |             |   Frag Function Code  |
-------------------------             -------------------------
|"Extra" depends on func|             |"Extra" depends on func|
-------------------------             -------------------------
|     Service           |             | Service w/Reply bit   |
-------------------------             -------------------------
|     Size of IOI       |             |          0            |
-------------------------             -------------------------
|    IOI (1st Byte)     |             | CIP Status            |
-------------------------             -------------------------
|          :            |             | Size of CIP Ext status|
-------------------------             -------------------------
|    IOI (Last Byte)    |             | CIP Ext status (1st)  |
-------------------------             -------------------------
| Parameters (1st byte) |             |          :            |
-------------------------             -------------------------
|          :            |             | CIP Ext status (last) |
-------------------------             -------------------------
| Parameters (last byte)|             | Response Data (1st)   |
-------------------------             -------------------------
                                      |          :            |
                                      -------------------------
                                      | Response Data (last)  |
                                      -------------------------
```

Appendix B: Fragmentation Protocol

PCCC has an inherent format of 244 bytes of application data. CIP messages may be up to 510 bytes. Therefore, a fragmentation protocol is used to allow CIP messages to pass transparently over PCCC based communication channels. This protocol is very similar for the two uses (connected and unconnected), but there are some subtle differences which result in two sets of state machines.

See ”Formal model”, connected sender and connected receiver, for the formal specification of the fragmentation protocol provided for CMD $0A_{hex}$.

Terms

- Sender

  This term is used to indicate the partner of the DH+ or DF1 transaction that is assigning the TNSW.

- Receiver

  This term is used to indicate the partner of the DH+ or DF1 transaction that is returning the TNSW value received.

- Function

  A function is used to communicate between the Sender and Receiver the reason this packet is being sent.

- Handle

  A handle is a means of associating multiple PCCC request/response pairs together into one transaction. It is required that the Receiver use only one handle value for the whole transaction; this will permit the Sender to Abort a transaction without having to wait for the receiver's next response.

Overview

This fragmentation protocol uses a **function field** identifying which fragment of a message is being transferred will be used. Each fragment is individually acknowledged before the next fragment is sent. The fragmentation protocol in PCCC also uses a **handle**. This allows multiple messages to be fragmented and reassembled in parallel between the devices using PCCC. The handle is assigned by the Receiver for each fragment. The Sender provides it back to the Receiver for each fragment.

Application

This protocol is available for PCCC commands 0Ah and 0Bh, which are used for transferring CIP connected and unconnected messages.

Functions

The protocol uses two bytes in each direction. The Function is the first byte. The second byte contains “extra” information, which depends on the value of the Function byte.

The following Functions are defined:

Table 8 Fragmentation Functions

Value	Functions sent by Sender	Functions sent by Receiver	Meaning of “Extra”
$00_{(hex)}$	Only	Only	0, unused, ignored
$01_{(hex)}$	First Request		Size
$02_{(hex)}$	Middle	Middle	Handle
$03_{(hex)}$	Last	Last	Handle
$04_{(hex)}$		First_Response	Handle
$05_{(hex)}$	Send_More		Handle
$06_{(hex)}$	Abort		Handle
$07_{(hex)}$		ACK	Handle
$08_{(hex)}$		Nak	Reason code

The text below describes the intended use of each function. The word “typically” refers to the normal expected successful operation. Obviously, in the real world, many error conditions can occur, including never receiving the “next expected” message. Robust implementations must account for all conditions. See the State Event Matrices later in this document for the complete formal specification of how these functions are used.

ONLY Function

This Function is used to send an unfragmented message, whether from the Sender or from the Receiver. The “extra” byte should be set to zero; it should be ignored by either party when it receives this Function.

The Receiver will typically respond with an ONLY (if the response is not fragmented) or FIRST_Resp (if the response is fragmented).

When this is sent from the Receiver to the Sender, it means that the Sender now has the complete response and can process it.

FIRST_Req Function

This Function is only sent from the Sender to the Receiver. It is sent with the data for the first fragment of a fragmented request. That is, it is only used for messages that are two or more fragments long. The “extra” data specifies the size of the buffer the Receiver should allocate to hold the complete message. It indicates the number of 256 byte blocks (minus 1) the Receiver should allocate for the complete application message. That is, application messages of size 0 to 256 bytes will have the “extra” data field set to 0; 257 to 512, to 1; and so on up to 65536 bytes. This is **not** the same as the number of fragments that will be sent.

The Receiver will typically respond with an ACK (which contains the handle).

For purposes of calculating the size, the first byte of the Unconnected message is the "Service code" of the CIP message. The first byte of the Connected message is the first byte of the CID. In both cases, this is the first byte after the "Extra" byte following the FIRST_Req FNC.

FIRST_Resp Function

This Function is only sent from the Receiver to the Sender. It is sent with the data for the first fragment of a fragmented response. That is, it is only used for messages that are two or more fragments long. The “extra” data specifies the handle that will be used for this transaction.

There is no size field. Therefore, the Sender should allocate a maximum size buffer for this response (or be prepared to dynamically allocate memory as the response fragments are received).

The Sender should issue a SEND_MORE Function with this handle to solicit the next part of the response. When the Sender receives the LAST Function, it knows that it has all the fragments.

MIDDLE Function

This Function is sent with the data for all fragments except the first and the last. That is, it is only used for messages that are three or more fragments long. The “extra” data specifies the Handle that was given to the Sender by the Receiver in the FIRST_Resp (for responses) or in the ACK to the FIRST_Req (for requests).

While it *is* permissible for the Receiver to change the value of the handle during the delivery of the fragments of a request or response, it is *highly recommended* that it does not do this, because it makes it more difficult to debug.

The Sender should use the value of the handle it receives in the MIDDLE Function (for responses) or in the ACK to the MIDDLE Function (for requests) for the next SEND_MORE function.

When the MIDDLE Function is sent from the Sender to the Receiver, the Receiver should respond with an ACK (which contains the handle) or a NAK (which contains an error code).

When the MIDDLE Function is sent from the Receiver to the Sender, it is the result of the SEND_MORE Function from the Sender. The handle to indicate which message the next fragment is requested for was given to the Receiver with the SEND_MORE.

LAST Function

This Function is sent with the data for the last fragment. The "extra" data specifies the Handle that was given to the Sender by the Receiver in the SEND_MORE (for responses) or in the ACK to the preceding request.

When this is sent from the Sender to the Receiver, the Receiver will typically respond with an ONLY (if the response is not fragmented) or FIRST_Resp (if the response is fragmented).

When this is sent from the Receiver to the Sender, it means that the Sender now has the complete response and can process it.

Send More Function

This is only used in the Sender to Receiver direction. It means the Sender is ready to accept another fragment of a fragmented response. It gives the handle supplied in the preceding response back to the Receiver to identify the transaction.

Abort Function

This is only used in the Sender to Receiver direction. It means the Sender wishes to discontinue the transaction specified by the handle. The Receiver should return an ACK to this.

ACK Function

This is only used in the Receiver to Sender direction. It means that the Receiver has accepted the fragment (delivered in the FIRST_Req or MIDDLE functions) and is ready to receive another one. It provides the handle to identify the transaction.

NAK Function

This is used to indicate to the Sender that the Receiver is unable to accept the Function sent. An error code is provided to indicate the reason. The error codes are:

0 - (Not used for errors; reserved to indicate SUCCESS.)

1 - means "Unrecognized Command Value"

2 - means "Sequence Error", for example a LAST after an ONLY.

3 - means "Not Enough Memory". As a response to a FIRST_Req or ONLY, indicates the receiver or server was unable to allocate the needed memory. As a response to a MIDDLE or LAST, indicates the receiver received more data than was allocated by the FIRST_Req.

4 - means "Unknown Handle".

5 - means "Unable to support requested memory size". As a response to a FIRST_Req indicates the receiver or server does not support the size requested.

6 - means "Connection Busy".

Formal Model

This is the formal model used for the specification of the PCCC Fragmentation Protocol. A compliant implementation acts, when viewed from outside the implementation, as if it has these exact parts implemented as specified below. However, the real implementation can be structured as deemed necessary by the implementer.

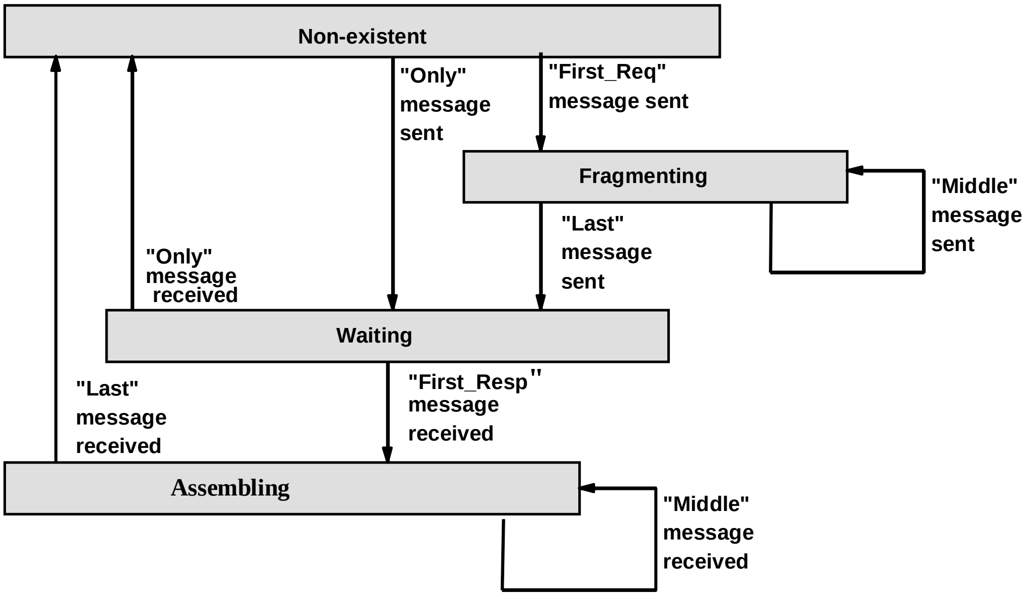
The formal model used for the specification of the PCCC Fragmentation Protocol has two major parts: Sender and Receiver. These are modeled with a State Event Matrix (SEM). These have a lifetime of a CIP message/transaction.

There is a separate sender and receiver definition for the UCMM and the Network Connection Pair.

Unconnected Sender

The Sender state machine starts up an instance each time it has a CIP message to send over PCCC. It uses the TNSW to match a response from the Receiver with this Sender. The following state transition diagram (STD) is intended to be informative in understanding the SEM; the SEM is the specification. The STD only shows the “normal” events.

Figure 7 State Transition Diagram - Unconnected Sender



The states for this SM are:

- Non-existent
- Fragmenting
- Waiting
- Assembling

The following events can happen.

- *Reset* Event
- *Timeout* Event

  This can either be the time specified by the application as part of the "UCMM request timeout" OR it can be the same network specific hard-coded time value specified for the Unconnected Receiver.

- *CIP Request Arrived* Event

- *PCCC Response Arrived* Event, FNC= ACK
- *PCCC Response Arrived* Event, FNC= NAK
- *PCCC Response Arrived* Event, FNC= Only
- *PCCC Response Arrived* Event, FNC= First_Resp
- *PCCC Response Arrived* Event, FNC= Middle
- *PCCC Response Arrived* Event, FNC= Last
- *PCCC Response Arrived* Event, FNC= Illegal command

The following internal storage is assumed:

- Buffer Pointer
- Room Left

Table 9 State Event Matrix - Unconnected Sender

State → **Event** ↓	**Non-existent**	**Fragmenting**	**Waiting**	**Assembling**
Reset or Timeout	Stay in Non-existent	Free memory previously allocated. Return a CIP "unable to deliver" error. Go to Non-existent. See Note.		
CIP Request Arrived	If the CIP message fits in one PCCC message, send the Only message and go to Waiting. Otherwise, initialize Buffer Pointer and Room Left, copy what fits into a First_Req message, send it, and go to Fragmenting	Each CIP message should start up a new state machine. If a CIP message "associated" with this SM is received, drop it and stay in the same state.		

<table>
<tr><th>State →
Event ↓</th><th>Non-existent</th><th>Fragmenting</th><th>Waiting</th><th>Assembling</th></tr>
<tr><td>PCCC Response Arrived with FNC=ACK</td><td>Stay in Non-existent</td><td>If Room Left fits in one PCCC message, send the Last message using Handle from the ACK and go to Waiting. Otherwise, update Buffer Pointer and Room Left, copy what fits into a Middle message using Handle from the ACK, send it, and stay in Fragmenting.</td><td colspan="2">(Assume the Receiver is confused.) Send an Abort (with the Handle provided in the ACK) to the Receiver. Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.</td></tr>
<tr><td>PCCC Response Arrived with FNC=NAK</td><td>Stay in Non-existent</td><td colspan="3">Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.</td></tr>
<tr><td>PCCC Response Arrived with FNC=Only</td><td>Stay in Non-existent</td><td>(Assume the Receiver is confused.) Send an Abort (with the Handle provided in the response) to the Receiver. Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.</td><td>Copy the response data into the CIP response. Send the CIP response. Go to Non-existent.</td><td>(Assume the Receiver is confused.) Send an Abort (with the Handle provided in the response) to the Receiver. Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.</td></tr>
<tr><td>PCCC Response Arrived with FNC=First_Resp</td><td>Stay in Non-existent</td><td>(Assume the Receiver is confused.) Send an Abort (with the Handle provided in the response) to the Receiver. Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.</td><td>Allocate a buffer to handle the maximum CIP response, initialize Buffer Pointer, Room Left, and Handle, copy the PCCC response data into the CIP response. Generate and send a Send_More message with the Handle. Go to Assembling.</td><td>(Assume the Receiver is confused.) Send an Abort (with the Handle provided in the response) to the Receiver. Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.</td></tr>
</table>

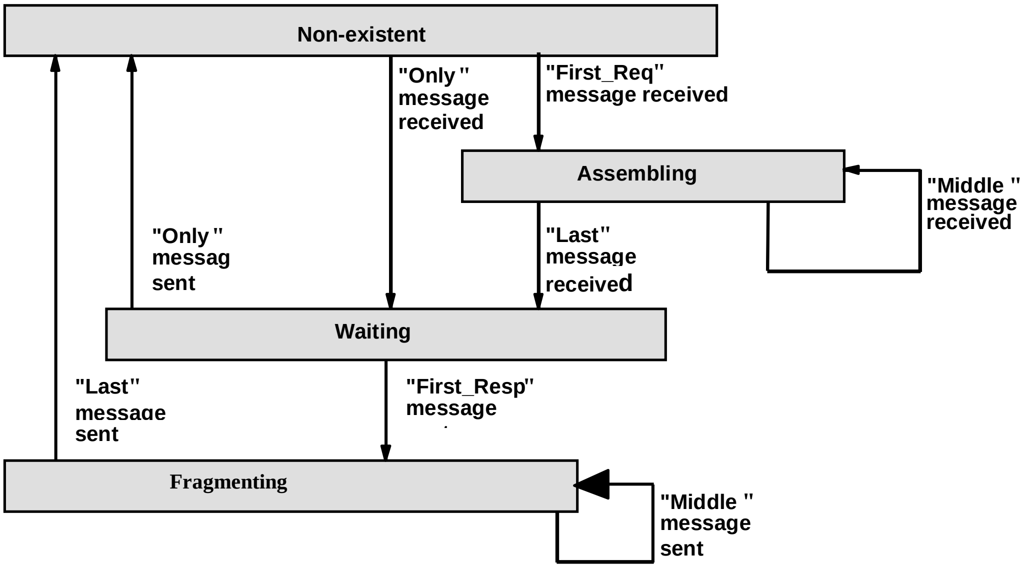
State → **Event ↓**	**Non-existent**	**Fragmenting**	**Waiting**	**Assembling**
PCCC Response Arrived with FNC=Middle	Stay in Non-existent	(Assume the Receiver is confused.) Send an Abort (with the Handle provided in the response) to the Receiver. Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.		If size of data is less than Room Left, (or if the additional memory can be dynamically allocated) then: Copy data to buffer at Buffer Pointer, update Buffer Pointer and Room Left, send a Send_More message with the Handle from the Middle, stay in Assembling. else: Free memory previously allocated, [send an Abort?] return a CIP “unable to deliver” error, go to Non-existent
PCCC Response Arrived with FNC=Last	Stay in Non-existent	(Assume the Receiver is confused.) Send an Abort (with the Handle provided in the response) to the Receiver. Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.		If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, deliver the CIP response, and go to Non-existent. else: Free memory previously allocated, [send an Abort?] return a CIP “unable to deliver” error, go to Non-existent
PCCC Response Arrived with FNC=Illegal value	Stay in Non-existent	(Assume the Receiver is confused.) (Do *not* send an Abort, since we don’t even know if there is a handle with this illegal valued message.) Free memory previously allocated. Return a CIP “unable to deliver” error. Go to Non-existent.		

1. The Sender is permitted (but not required) to send an Abort if it has a Handle for the transaction. This will help the Receiver to free up resources more quickly.

Unconnected Receiver

An instance of the Receiver State Machine starts up when it receives the first fragment of a CIP message via PCCC and ceases to exist with the delivery of the last fragment of the CIP response. It assigns and uses the Handle in the messages (after the first one) to match the Sender's request with the appropriate instance of one of the Receiver state machines. The following state transition diagram (STD) is intended to be informative in understanding the SEM; the SEM is the specification. The STD only shows the "normal" events.

Figure 8 State Transition Diagram -Unconnected Receiver



The following events can happen.

- *Reset* Event
- *Timeout* Event

  This is a "long" timer, used to free up resources, which the Sender evidently has ceased to want to use. It should be 3 minutes for DF1/RS232 links.

- *CIP Response Arrived* Event
- *PCCC Request Arrive*d Event, FNC= Only
- *PCCC Request Arrived* Event, FNC= First_Req
- *PCCC Request Arrived* Event, FNC= Middle
- *PCCC Request Arrived* Event, FNC= Last
- *PCCC Request Arrived* Event, FNC= Send_More

- *PCCC Request Arrived* Event, FNC= Abort
- *PCCC Request Arrived* Event, FNC= Illegal command

The following internal storage is assumed:

- Buffer Pointer
- Room Left

Table 10 State Event Matrix - Unconnected Receiver

State → **Event ↓**	**Non-existent**	**Assembling**	**Waiting**	**Fragmenting**
Reset or Timeout	Stay in Non-existent	Free memory previously allocated. Go to Non-existent. Stop Timer.		
CIP Response Arrived	Each CIP message should have its own state machine. If a CIP message "associated" with this SM is received, drop it and stay in the same state.		If the CIP message fits in one PCCC message, send the Only message, free up any resources, and go to Non-existent. Stop Timer. Otherwise, initialize Buffer Pointer and Room Left, copy what fits into a First_Resp message, send it, and go to Fragmenting. Start Timer. See Note.	Each CIP message should start up its own state machine. If a CIP message "associated" with this SM is received, drop it and stay in the same state.
PCCC Request Arrived with FNC=Only	Allocate resources needed; assign a Handle to identify the resources. Copy the data into the CIP request. Send the CIP request. Go to Waiting.	Not applicable. Since there is no handle in the Only request, it cannot specify an existing transaction. Therefore, the only thing that can be done is to create a new instance as described in the text for the Non-existent state.		

State → Event ↓	Non-existent	Assembling	Waiting	Fragmenting
PCCC Request Arrived with FNC=First_Req	Allocate resources needed to handle the size specified, assign a Handle to identify the resources. Copy the data into the CIP request buffer. Initialize the Buffer Pointer and Room Left. Send an ACK with the Handle assigned. Go to Assembling. Start Timer	Not applicable. Since there is no handle in the First_Req request, it cannot specify an existing transaction. Therefore, the only thing that can be done is to create a new instance as described in the text for the Non-existent state.		
PCCC Request Arrived with FNC=Middle	(Assume the Sender is confused.) Return a PCCC response of NAK (sequence error). Stay in Non-existent	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, update Buffer Pointer and Room Left, send an ACK message with the Handle, stay in Assembling. Else: Free memory allocated, return a PCCC response of NAK (no memory), go to Non-existent. Stop Timer.	(Assume the Sender is confused.) Free memory previously allocated. Return a PCCC response of NAK (sequence error). Go to Non-existent. Stop Timer.	

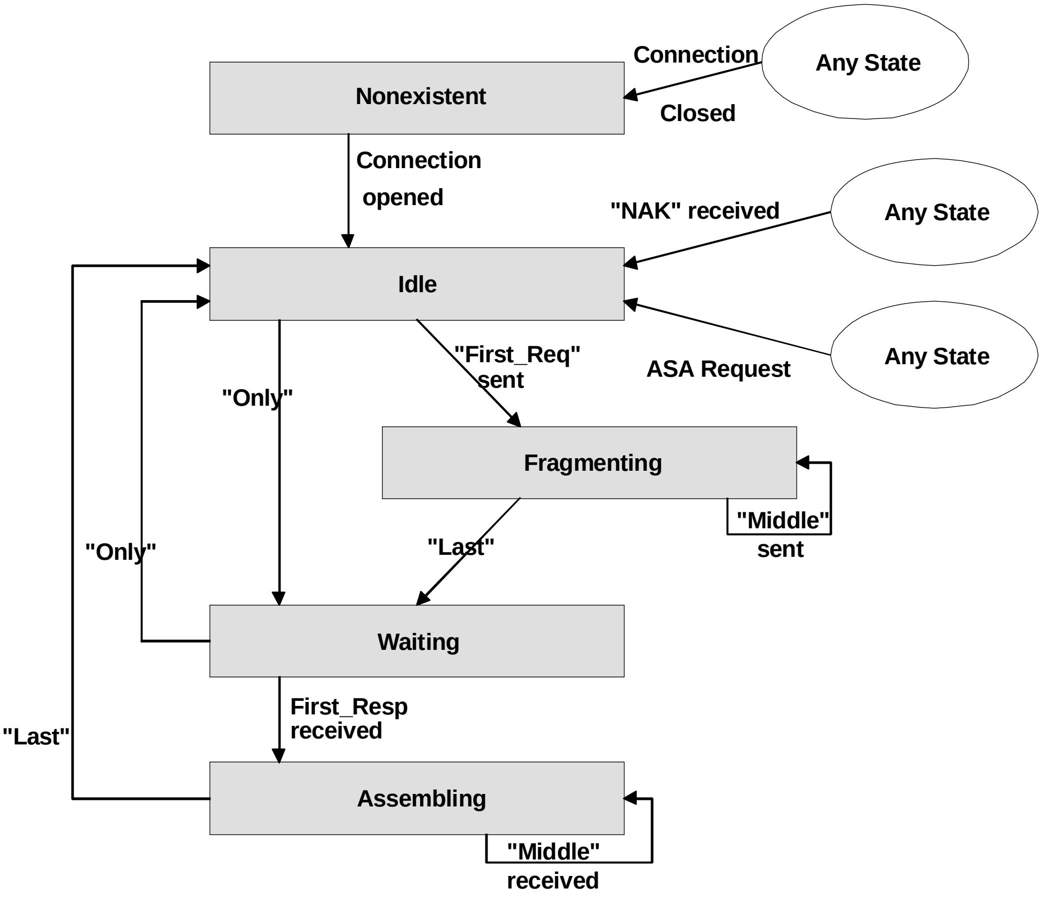
<table>
<tr><th>State →
Event ↓</th><th>Non-existent</th><th>Assembling</th><th>Waiting</th><th>Fragmenting</th></tr>
<tr><td>PCCC Request Arrived with FNC=Last</td><td>(Assume the Sender is confused.) Return a PCCC response of NAK (sequence error). Stay in Non-existent</td><td>If size of data is less than Room Left, then:
Copy data to buffer at Buffer Pointer, deliver the CIP request, and go to Waiting.
Else:
Free memory allocated, return a PCCC response of NAK (no memory), go to Non-existent. Stop Timer.</td><td colspan="2">(Assume the Sender is confused.) Free memory previously allocated. Return a PCCC response of NAK (sequence error). Go to Non-existent. Stop Timer.</td></tr>
<tr><td>PCCC Request Arrived with FNC=Send_More</td><td>(Assume the Sender is confused.) Return a PCCC response of NAK (sequence error). Stay in Non-existent</td><td colspan="2">(Assume the Sender is confused.) Free memory previously allocated. Return a PCCC response of NAK (sequence error). Go to Non-existent. Stop Timer (when done)</td><td>If Room Left fits in one PCCC message, send the Last message using Handle and go to Non-existent. Stop Timer.
Otherwise, update Buffer Pointer and Room Left, copy what fits into a Middle message using Handle, send it, and stay in Fragmenting.</td></tr>
<tr><td>PCCC Request Arrived with FNC=Abort</td><td colspan="4">Free memory previously allocated, if any. Return a PCCC ACK response. Go to (or stay in) Non-existent.</td></tr>
<tr><td>PCCC Request Arrived with FNC=
Illegal value</td><td colspan="4">Return a PCCC response of NAK (unrecognized command value). It is not possible to reliably determine which handle is associated with this request, therefore, do not change the state of anything.</td></tr>
</table>

1. In the case where the message request is being forwarded to another device via the Connection Manager, it is required for the local Connection Manager to generate an "CIP Response" when it does not receive a reply to its forwarded request. So in no case should a device NOT exit the Waiting state.

Connected Sender

The Sender state machine comes into existence when a connection is opened. It uses the TNSW to match a response from the Receiver with this Sender. The following state transition diagram (STD) is intended to be informative in understanding the SEM; the SEM is the specification. The STD only shows the “normal” events.

Figure 9 State Transition Diagram - Connected Fragmentation Sender



The states for this SM are:

- Non-existent
- Idle
- Fragmenting
- Waiting
- Assembling

The following events can happen:

- *Reset* Event
- *Connection Closed* Event
- *PCCC Timeout* Event
- *CIP Request Arrived* Event
- *PCCC Response Arrived* Event, FNC= ACK
- *PCCC Response Arrived* Event, FNC= NAK
- *PCCC Response Arrived* Event, FNC= Only
- *PCCC Response Arrived* Event, FNC= First_Resp
- *PCCC Response Arrived* Event, FNC= Middle
- *PCCC Response Arrived* Event, FNC= Last
- *PCCC Response Arrived* Event, FNC= Illegal command

NOTE: All "*PCCC Response Arrived* Events" assume matching TNSW. A PCCC response with an unexpected TNSW should be dropped with no change in state.

The following internal storage is assumed:

- Buffer Pointer
- Room Left
- TNSW this is incremented for each distinct PCCC message. (Link layer retries will reuse the same TNSW.)

Note: CIP Connections do not return an "unable to deliver" status, they just let the return path time out.

Note: At Connection Establishment time, the memory is allocated according to the parameters in the Forward_Open request. This memory is deallocated when the connection is closed.

Note: To make this table easier to understand, the "Non-Existent" state is not shown. It transitions to the "Idle" state when the connection is open, and back to the Non-Existent state from any other state when the connection is closed.

Table 11 State Event Matrix - Connected Sender

State → **Event ↓**	**Idle**	**Waiting**	**Fragmenting**	**Assembling**
Reset or Connection Closed	Go to Non-Existent.			
PCCC Timeout	Stay in Idle.	(Assume Receiver is temporarily disconnected or busy) Go to Idle.		
CIP Request Arrived	Increment TNSW. If the CIP message fits in one PCCC message, send the Only message and go to Waiting. Otherwise, initialize Buffer Pointer and Room Left, copy what fits into a First_Req message, send it, and go to Fragmenting.	(Assume Client got tired of waiting.) Increment TNSW. If the CIP message fits in one PCCC message, send the Only message and go to (or stay in) Waiting. Otherwise, initialize Buffer Pointer and Room Left, copy what fits into a First_Req message, send it, and go to (or stay in) Fragmenting.		
PCCC Response Arrived with FNC=ACK	Stay in Idle.	(Assume the Receiver is confused.) Go to Idle.	Increment TNSW. If Room Left fits in one PCCC message, send the Last message using Handle from the ACK and go to Waiting. Otherwise, update Buffer Pointer and Room Left, copy what fits into a Middle message using Handle from the ACK, send it, and stay in Fragmenting.	(Assume the Receiver is confused.) Go to Idle.
PCCC Response Arrived with FNC=NAK	Stay in (or go to) Idle.			

State → **Event** ↓	**Idle**	**Waiting**	**Fragmenting**	**Assembling**
PCCC Response Arrived with FNC=Only	Stay in Idle.	Copy the response data into the CIP response. Send the CIP response. Go to Idle.	(Assume the Receiver was confused, but is now in the Idle state.) Go to Idle.	(Assume the Receiver was confused, but is now in the Idle state.) Go to Idle.
PCCC Response Arrived with FNC=First_Resp	Stay in Idle.	Initialize Buffer Pointer to start, Room Left to connection size, and save Handle. Increment TNSW. If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, update Buffer Pointer and Room Left, send a Send_More message with the Handle, go to Assembling. Else: Drop the response, go to idle.	(Assume the Receiver is confused.) Sender's next First_Req or Only will resynchronize the Receiver. Go to Idle.	(Assume the Receiver is confused.) Sender's next First_Req or Only will resynchronize the Receiver. Go to Idle.

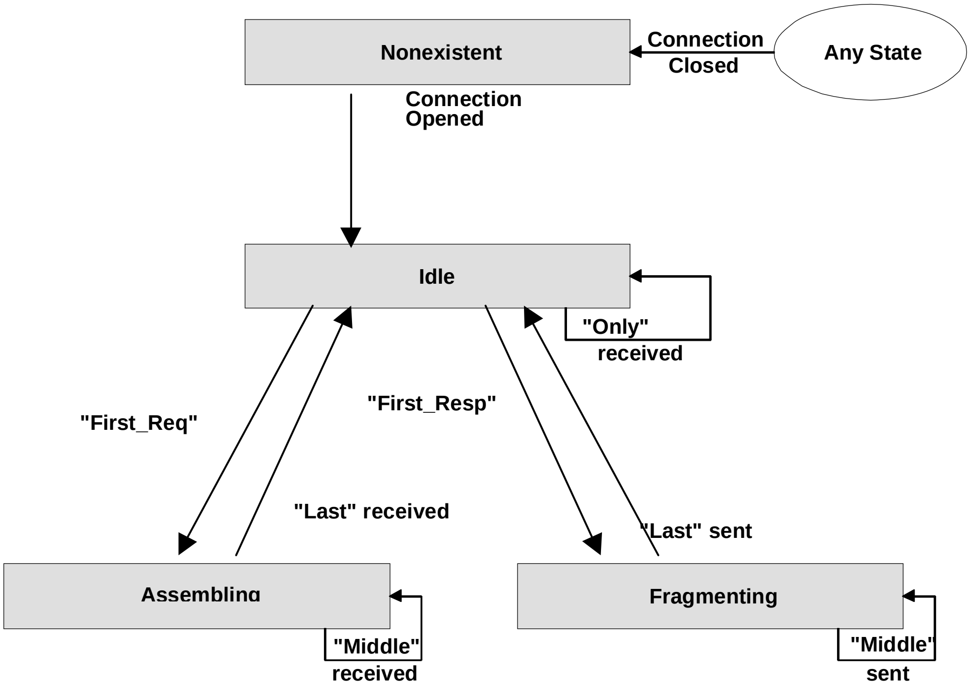
State → Event ↓	Idle	Waiting	Fragmenting	Assembling
PCCC Response Arrived with FNC=Middle	Stay in Idle.	(Assume the Receiver is confused.) Sender's next First_Req or Only will resynchronize the Receiver. Go to Idle.	(Assume the Receiver is confused.) Sender's next First_Req or Only will resynchronize the Receiver. Go to Idle.	Increment TNSW. If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, update Buffer Pointer and Room Left, send a Send_More message with the Handle from the Middle, stay in Assembling. Else: Drop the Response. Go to Idle.
PCCC Response Arrived with FNC=Last	Stay in Idle.	(Assume the Receiver was confused, but is now in the Idle state.) Go to Idle.	(Assume the Receiver was confused, but is now in the Idle state.) Go to Idle.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, deliver the CIP response, and go to Idle. else: Go to Idle
PCCC Response Arrived with FNC=Illegal value	Stay in Idle.	(Assume the Receiver is very confused.) (Do *not* send an Abort, since we don't even know if there is a handle with this illegal valued message.) Go to Idle.		

There are several occasions where the Sender sends an Abort to get the Receiver unconfused. This is not really necessary, since the next CIP Request will get it unstuck, but it is a courtesy that will allow it to return to normal quicker. It is not necessary to wait for the ACK, since ACKs received in the Idle state are ignored. Also, if a new request goes out before the ACK is received, this sender will no longer be expecting the TNSW of the Abort and it will be dropped.

Connected Receiver

An instance of the Receiver State Machine starts up when a connection is opened and ceases to exist when the connection is closed. It assigns and uses the Handle in the messages (after the first one) to match the Sender's request with the appropriate instance of one of the Receiver state machines. The following state transition diagram (STD) is intended to be informative in understanding the SEM; the SEM is the specification. The STD only shows the "normal" events.

Figure 10 State Transition Diagram - Connected Fragmentation Receiver



The states for this SM are:

- Non-existent
- Idle
- Fragmenting
- Assembling

The following events can happen.

- *Reset* Event
- *Connection Closure* Event

- *CIP Response Arrived* Event
- *PCCC Request Arrived* Event, FNC= Only
- *PCCC Request Arrived* Event, FNC= First_Req
- *PCCC Request Arrived* Event, FNC= Middle
- *PCCC Request Arrived* Event, FNC= Last
- *PCCC Request Arrived* Event, FNC= Send_More
- *PCCC Request Arrived* Event, FNC= Abort
- *PCCC Request Arrived* Event, FNC= Illegal command

The following internal storage is assumed:

- Buffer Pointer
- Room Left
- TNSW This is the most recently received TNSW in a request. Note that this state machine might deliver multiple responses with the same TNSW under some conditions (E.g., multiple CIP responses received)

Table 12 State Event Matrix - Connected Receiver

State → **Event ↓**	**Idle**	**Assembling**	**Fragmenting**
Reset or Connection closure	Go to Non-existent.		
CIP Response Arrived	If the CIP message fits in one PCCC message, send the Only message, stay in Idle. Otherwise, initialize Buffer Pointer and Room Left, copy what fits into a First_Resp message, send it, and go to Fragmenting.	The sender is not waiting for a response, so drop it and stay in the same state.	The sender is not waiting for a response, so drop it and stay in the same state.
PCCC Request Arrived with FNC=Only	Assign a Handle to identify the resources. Copy the data into the CIP request. Send the CIP request. Stay in Idle.	(Assume the sender quit the old message and wants to start a new sequence.) Assign a Handle to identify the resources. Copy the data into the CIP request. Send the CIP request. Go to Idle.	

State → **Event** ↓	**Idle**	**Assembling**	**Fragmenting**
PCCC Request Arrived with FNC=First_Req	Check that the resources allocated to the connection are sufficient for the size specified, assign a Handle to identify the resources; if they are not, send a NAK (Not Enough Memory) and stay in idle. Copy the data into the CIP request buffer. Initialize the Buffer Pointer and Room Left. Send an ACK with the Handle assigned. Go to Assembling.	(Assume the sender quit the old message and wants to start a new sequence.) Check that the resources allocated to the connection are sufficient for the size specified, assign a Handle to identify the resources; if they are not, send a NAK (Not Enough Memory) and go to idle. Copy the data into the CIP request buffer. Initialize the Buffer Pointer and Room Left. Send an ACK with the Handle assigned. Go to Assembling.	
PCCC Request Arrived with FNC=Middle	(Assume the Sender is confused.) Return a PCCC response of NAK (Unknown Handle). Stay in Idle.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, update Buffer Pointer and Room Left, send an ACK message with the Handle, stay in Assembling. Else: Return a PCCC response of NAK (Not Enough Memory), go to Idle.	(Assume the Sender is confused.) Return a PCCC response of NAK (sequence error). Go to Idle.
PCCC Request Arrived with FNC=Last	(Assume the Sender is confused.) Return a PCCC response of NAK (Unknown Handle). Stay in Idle.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, deliver the CIP request, and go to Idle. Else: Return a PCCC response of NAK (Not Enough Memory), go to Idle.	(Assume the Sender is confused.) Return a PCCC response of NAK (sequence error). Go to Idle.

State → **Event** ↓	**Idle**	**Assembling**	**Fragmenting**
PCCC Request Arrived with FNC=Send_More	(Assume the Sender is confused.) Return a PCCC response of NAK (Unknown Handle). Stay in Idle.	(Assume the Sender is confused.) Return a PCCC response of NAK (sequence error). Go to Idle.	If Room Left fits in one PCCC message, send the Last message using Handle and go to Idle. Otherwise, update Buffer Pointer and Room Left, copy what fits into a Middle message using Handle, send it, and stay in Fragmenting.
PCCC Request Arrived with FNC=Abort	Return a PCCC ACK response. Go to (or stay in) Idle.		
PCCC Request Arrived with FNC= Illegal value	Return a PCCC response of NAK (unrecognized command value). It is not possible to reliably determine which connection or handle is associated with this request, therefore, do not change the state of anything.		

Example: Time Sequence Diagrams

In the following diagrams the arrow pointing to the right indicates a message going from the Sender to the Receiver. The arrow pointing to the left indicates a message going from the Receiver to the Sender. Note the strict alternation of Sender and Receiver messages. Only the PCCC messages are shown; the link layer ACKs are not shown.

Table 13 Short Request, Short Response

Sender	Function	Receiver	Extra
	Only	→	0
(Wait for application response.)			
←	Only		0

Table 14 Two Fragment Request, Short Response

Sender	Function	Receiver	Extra
	First	→	Size
←	ACK		Handle A, picked by Receiver
←	Last		Handle A
(Wait for application response.)			
←	Only		0

Table 15 Three Fragment Request, Short Response

Sender	Function	Receiver	Extra
	First	→	Size
←	ACK		Handle A, picked by Receiver
	Middle	→	Handle A
←	ACK		Handle A
	Last	→	Handle A
(Wait for application response.)			
←	Only		0

Table 16 Long Request, Short Response

Sender	Function	Receiver	Extra
	First	→	Size
←	ACK		Handle A, picked by Receiver
	Middle	→	Handle A
←	ACK		Handle A
	Middle	→	Handle A
←	ACK		Handle A
	…		
	Last	→	Handle A
	(Wait for application response.)		
←	Only		0

Table 17 Short Request, Two Fragment Response

Sender	Function	Receiver	Extra
	Only	→	0
	(Wait for application response.)		
←	First_Resp		Handle A, picked by Receiver
	Send_More	→	Handle A
←	Last		Handle A

Table 18 Two Fragment Request, Two Fragment Response

Sender	Function	Receiver	Extra
	First_Req	→	Size
←	ACK		Handle_A, picked by Receiver
	Last	→	Handle_A
(Wait for application response.)			
←	First_Resp		Handle_A
	Send_More	→	Handle_A
←	Last		Handle_A

Table 19 Two Fragment Request, Long Response

Sender	Function	Receiver	Extra
	First_Req	→	Size
←	ACK		Handle_A, picked by Receiver
	Last	→	Handle_A
(Wait for application response.)			
←	First_Resp		Handle_A
	Send_More	→	Handle_A
←	Middle		Handle_A
	Send_More	→	Handle_A
←	Middle		Handle_A
	...		
	Send_More	→	Handle_A
←	Last		Handle_A

Example: Byte Patterns

These are examples of PCCC messages that show the byte ordering intended by the above specifications. The “Connected” examples assume a Class 3 Transport connection to the Message Router. These examples all start with the first byte specified by the PCCC specification, that is, CMD. The lines that are **bold** are those controlled by this specification; other lines are controlled by other specifications.

Unconnected, Short

The unconnected CIP message and response can each be up to 242 bytes, starting with the “CIP Service”, before fragmentation is needed.

Table 20 Unconnected Short Request

Byte	**Value (in Hex)**
CMD	0B (PCCC request: CIP Unconnected)
STS	00
TNSW-1	1
TNSW-2	00
FNC	**0** (Only**)**
Extra (unused)	**0**
CIP Service	01 (Get Attributes All, Request)
Size of IOI	02
IOI, first byte	20 (Class)
....	(rest of CIP message)

Table 21 Unconnected Short Response

Byte	**Value (in Hex)**
CMD	4B (PCCC response: CIP Unconnected)
STS	00
TNSW-1	1
TNSW-2	00
FNC	**0 (**Only**)**
Extra (unused)	**0**
CIP Service	81 (Get Attributes All, Response)
Unused	00
CIP General Status	00
Size of CIP Extended Status	00
....	(rest of CIP response)

Connected, Short

The Connected CIP message and response can each be up to 238 bytes, starting with the “CIP Service”, before fragmentation is needed.

Table 22 Connected Short Request

Byte	**Value (in Hex)**
CMD	0A (PCCC Request: CIP Connected)
STS	00
TNSW-1	2
TNSW-2	00
FNC	**0 (**Only**)**
Extra (unused)	**0**
Connection ID, Low byte	14
Connection ID, High byte	27
Transport Header, Low byte	56
Transport Header, High byte	12
CIP Service	01 (Get Attributes All, Request)
Size of IOI	02
IOI, first byte	20 (Class)
....	(rest of CIP message)

Table 23 Connected Short Response

Byte	**Value (in Hex)**
CMD	4A (PCCC Response: CIP Connected)
STS	00
TNSW-1	2
TNSW-2	00
FNC	**0 (**Only**)**
Extra (unused)	**0**
Connection ID, Low byte	14
Connection ID, High byte	27
Transport Header, Low byte	56
Transport Header, High byte	12
CIP Service	81 (Get Attributes All, Response)
Unused	00
CIP General Status	00
Size of CIP Extended Status	00
....	(rest of CIP response)

Unconnected, Long

This shows the bytes for a "Two Fragment Request, Two Fragment Response" when sending an unconnected CIP Message. The extension to 3 or more fragments in either direction ought to be obvious. Note that the CIP Service, IOI, etc. are only sent once in each direction.

Table 24 Unconnected "First" Request

Byte	**Value (in Hex)**
CMD	0B (PCCC Request: CIP Unconnected)
STS	00
TNSW-1	3
TNSW-2	00
FNC	**01 (**First_Req**)**
Extra (Size)	**01 (meaning "Allocate 512 bytes")**
CIP Service	01 (Get Attributes All, Request)
Size of IOI	02
IOI, first byte	20 (Class)
....	(part of CIP message)

Table 25 Unconnected Response

Byte	**Value (in Hex)**
CMD	4B (PCCC Response: CIP Unconnected)
STS	00
TNSW-1	3
TNSW-2	00
FNC	**7 (**ACK**)**
Extra (Handle)	**7**

Table 26 Unconnected “Last” Request

Byte	**Value (in Hex)**
CMD	0B (PCCC Request: CIP Unconnected)
STS	00
TNSW-1	4
TNSW-2	00
FNC	**3 (**Last**)**
Extra (Handle)	**7**
....	(rest of CIP message)

Table 27 Unconnected “First” Response

Byte	**Value (in Hex)**
CMD	4B (PCCC Response: CIP Unconnected)
STS	00
TNSW-1	4
TNSW-2	00
FNC	**4 (**First_Resp**)**
Extra (Response Handle)	**7**
CIP Service	81 (Get Attributes All, Response)
Unused	00
CIP General Status	00
Size of CIP Extended Status	00
....	(part of CIP response)

Table 28 Unconnected Request for Another Fragment

Byte	**Value (in Hex)**
CMD	0B (PCCC Request: CIP Unconnected)
STS	00
TNSW-1	5
TNSW-2	00
FNC	**5 (**Send_More**)**
Extra (Response Handle)	**7**

Table 29 Unconnected "Last" Response

Byte	**Value (in Hex)**
CMD	4B (PCCC Response: CIP Unconnected)
STS	00
TNSW-1	5
TNSW-2	00
FNC	**3 (**Last**)**
Extra (Response Handle)	**7**
....	(Rest of CIP response)

Connected, Long

This shows the bytes for a "Two Fragment Request, Two Fragment Response" when sending a connected CIP Message. The extension to 3 or more fragments in either direction ought to be obvious. Note that the Connection ID, Transport header, CIP Service, IOI, etc., are only sent once in each direction.

Table 30 Connected "First" Request

Byte	**Value (in Hex)**
CMD	0A (PCCC Request: CIP Connected)
STS	00
TNSW-1	6
TNSW-2	00
FNC	**1 (**First_Req**)**
Extra (Size)	**1 (**meaning "Allocate 512 bytes**")**
Connection ID, Low byte	14
Connection ID, High byte	27
Transport Header, Low byte	57
Transport Header, High byte	12
CIP Service	01 (Get Attributes All, Request)
Size of IOI	02
IOI, first byte	20 (Class)
....	(part of CIP message)

Table 31 Connected Response

Byte	Value (in Hex)
CMD	4A (PCCC Response: CIP Connected)
STS	00
TNSW-1	6
TNSW-2	00
FNC	**7 (**ACK**)**
Extra (Handle)	**16**

Table 32 Connected "Last" Request

Byte	Value (in Hex)
CMD	0A (PCCC Request: CIP Connected)
STS	00
TNSW-1	7
TNSW-2	00
FNC	**3 (**Last**)**
Extra (Handle)	**16**
....	(rest of CIP message)

Table 33 Connected "First" Response

Byte	**Value (in Hex)**
CMD	4A (PCCC Response: CIP Connected)
STS	00
TNSW-1	7
TNSW-2	00
FNC	**4** (First_Resp**)**
Extra (Handle)	**16**
Connection ID, Low byte	14
Connection ID, High byte	27
Transport Header, Low byte	57
Transport Header, High byte	12
CIP Service	81 (Get Attributes All, Response)
Unused	00
CIP General Status	00
Size of CIP Extended Status	00
....	(Part of CIP response)

Table 34 Connected Request for Another Fragment

Byte	**Value (in Hex)**
CMD	0A (PCCC Request: CIP Connected)
STS	00
TNSW-1	8
TNSW-2	00
FNC	**05** (Send_More**)**
Extra (Response Handle)	**16**

Table 35 Connected “Last” Response

Byte	**Value (in Hex)**
CMD	4A (PCCC Response: CIP Connected)
STS	00
TNSW-1	8
TNSW-2	00
FNC	**3 (**Last**)**
Extra (Response Handle)	**16**
....	(Rest of CIP response)