iCC 1994

1st international CAN Conference

in Mainz (Germany)

Sponsored by

Allen Bradley
National Semiconductor
Philips Semiconductors

Organized by

CAN in Automation (CiA)
international users and manufacturers group
Am Weichselgarten 26
D-91058 Erlangen
Phone +49-9131-69086-0
Fax +49-9131-69086-79
Email:headquarters@can-cia.de
URL: http://www.can-cia.de

DeviceNet™ Application Protocol

Authors: Dan Noonen, Stuart Siegel, Pat Maloney

Allen-Bradley 747 Alpha Drive Highland Heights, Ohio 44143 (USA)
(216) 646-5000

Abstract

DeviceNet is an open network developed by Allen-Bradley based on CAN that is designed to allow low cost industrial control devices to communicate with each other. DeviceNet is defined in terms of an abstract object model which presents the suite of communication services available and describes the externally visible behavior of a DeviceNet node. The DeviceNet Model is application independent. DeviceNet provides the communication services needed by various types of applications. Many of today's lower level industrial control devices must retain their low cost/low resource characteristics even when directly connected to a network. DeviceNet takes this into consideration by defining a specific instance of the Model for communications typically seen in a Master/Slave application. This is referred to as the Predefined Master/Slave Connection Set. This paper describes the DeviceNet Communication Model and presents the Predefined Master/Slave Connection Set.

DeviceNet Abstract Object Model

A DeviceNet compliant device is abstractly modeled as a collection of objects. An object provides a logical interface to the internal components of a device (e.g. logic, memory, etc.) and defines how that device reacts to particular events. The object model includes both communication specific objects and application specific objects (see figure 1). The communication objects manage and provide for the run-time exchange of messages across DeviceNet, while the application objects implement the product specific features and provide a logical interface to product specific information. Although the definition and implementation of the application objects are just as important as the communication objects, this paper discusses the communication model.

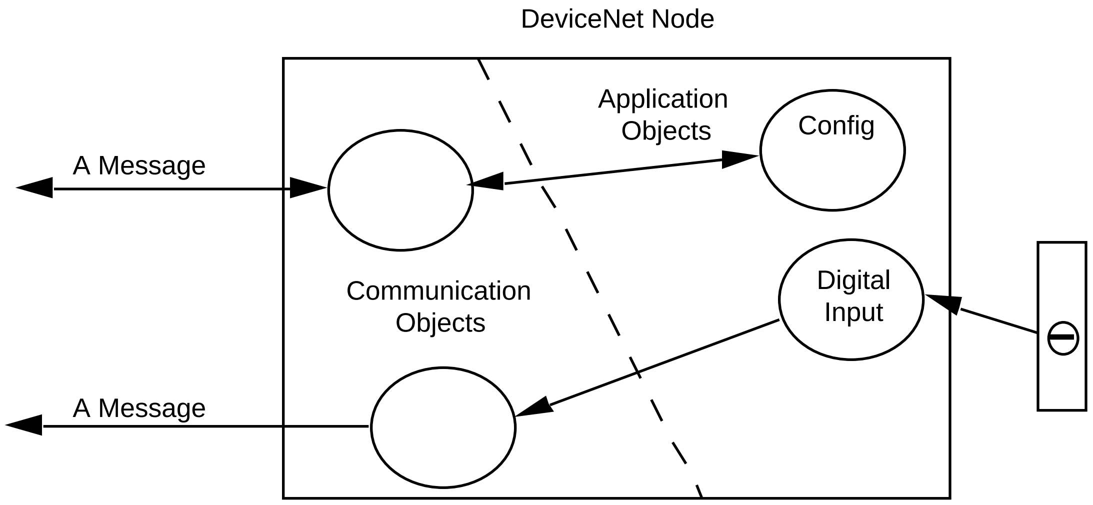


Figure 1 - DeviceNet Abstract Model

DeviceNet Addressing

In order to access internal components/logic within a device, the DeviceNet Communication Model defines an addressing scheme that provides access to objects within a device. This information is physically represented within the DeviceNet protocol (See figure 2). The object model addressing information includes:

Device Address - Referred to as the Media Access Control Identifier (MAC ID). This is an integer identification value assigned to each *node* on DeviceNet. This value distinguishes a node among all other nodes on the same link. A test at power-up guarantees the uniqueness of the value on the network (duplicate MAC ID detection).

Class Identifier (Class ID) - The term *Class* refers to a set of objects that represent the same type of system component. The Class ID is an integer identification value assigned to each *Object Class* accessible from the network.

Instance Identifier (Instance ID) - The term *Instance* refers to the actual representation of an object within an Object Class. The Instance ID is an integer identification assigned to an Object Instance that identifies it among all *Instances* of the same *Class* within a particular device.

Attribute Identifier (Attribute ID) - Attributes are parameters associated with an Object Class and/or an Object Instance. Attributes typically provide some type of status information or govern the operation of an object. The Attribute ID is an integer identification value assigned to a Class and/or Instance Attribute.

Service Code - An integer identification value which denotes a particular Object Instance and/or Object Class function.

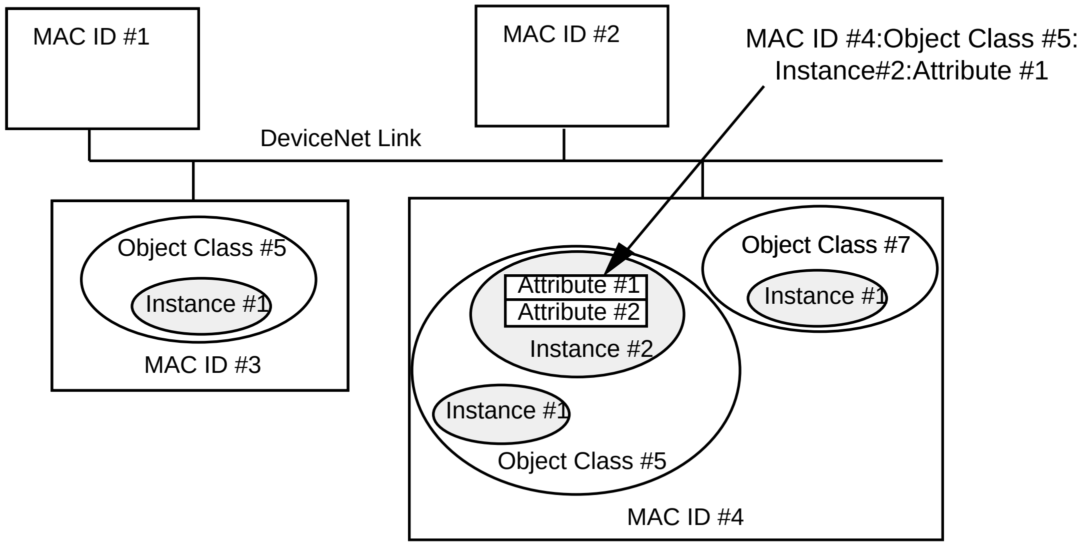


Figure 2 - DeviceNet Addressing Scheme

The use of the addressing scheme allows a simple yet robust device definition that covers a large range of product functionality and cost.

DeviceNet Connection Overview

DeviceNet defines a connection-based scheme to facilitate all application communications. A DeviceNet connection provides a communication path between multiple end-points (such as attributes of objects). A connection is established between two or more end-points prior to exchanging data.

Connection *Objects* model the communication characteristics of a particular Application-to-Application relationship. The Connection Object is one of the Communication Objects within the DeviceNet Object

Model. A Connection Object can be viewed as the container for characteristics associated with a communication relationship being managed by a particular device.

DeviceNet defines two types of Connections:

> I/O Connections - Provide dedicated, special purpose communication paths between a producing application and one or more consuming applications. I/O *Messages* are exchanged across I/O Connections. I/O Messages carry application specific I/O data.
>
> Explicit Messaging Connections - Provide generic, multi-purpose communication paths between two devices. These connections often are referred to as Messaging Connections. Explicit Messages are exchanged across Explicit Messaging Connections. Explicit Messaging provides the typical request/response-oriented network communications used to perform node configuration (e.g. parameter upload/download) and/or problem diagnosis.

An I/O message consists of a Connection ID and associated I/O data. DeviceNet does not define any protocol information resident in the CAN Data Field of an I/O Message whose length is less than or equal to 8 bytes. The meaning of the data within an I/O Message is implied by the associated Connection ID. The connection end-points are assumed to have knowledge of the intended use or meaning of the I/O Message (See figure 3).

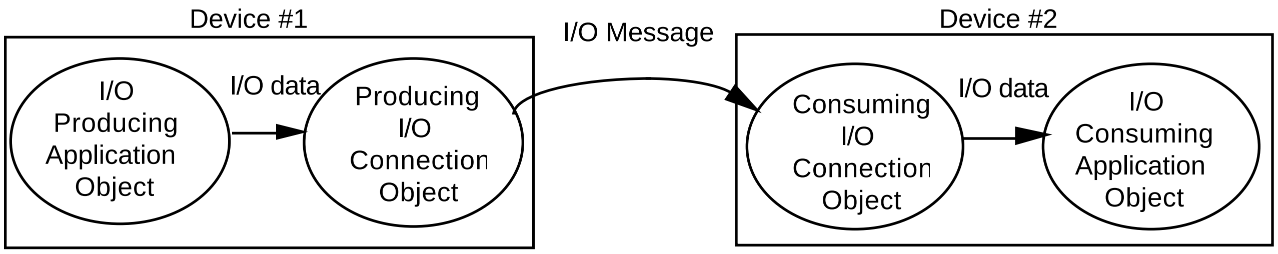

Device #1
I/O Message
Device #2
I/O Producing Application Object
I/O data
Producing I/O Connection Object
Consuming I/O Connection Object
I/O data
I/O Consuming Application Object

Figure 3 - I/O Connection

Explicit Messages are used to command the performance of a particular task and to report the results of performing the task. DeviceNet does define protocol information that is passed in the CAN Data Field of an Explicit Message. An Explicit Message consists of a Connection ID and associated messaging protocol information (See Figure 4).

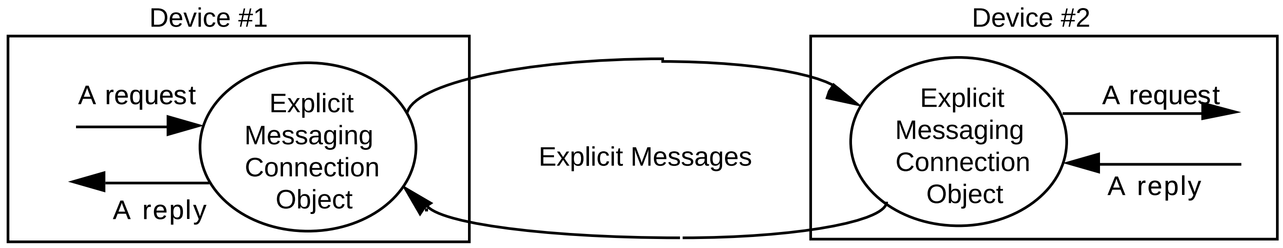

Device #1
Device #2
A request
A reply
Explicit Messaging Connection Object
Explicit Messages
Explicit Messaging Connection Object
A request
A reply

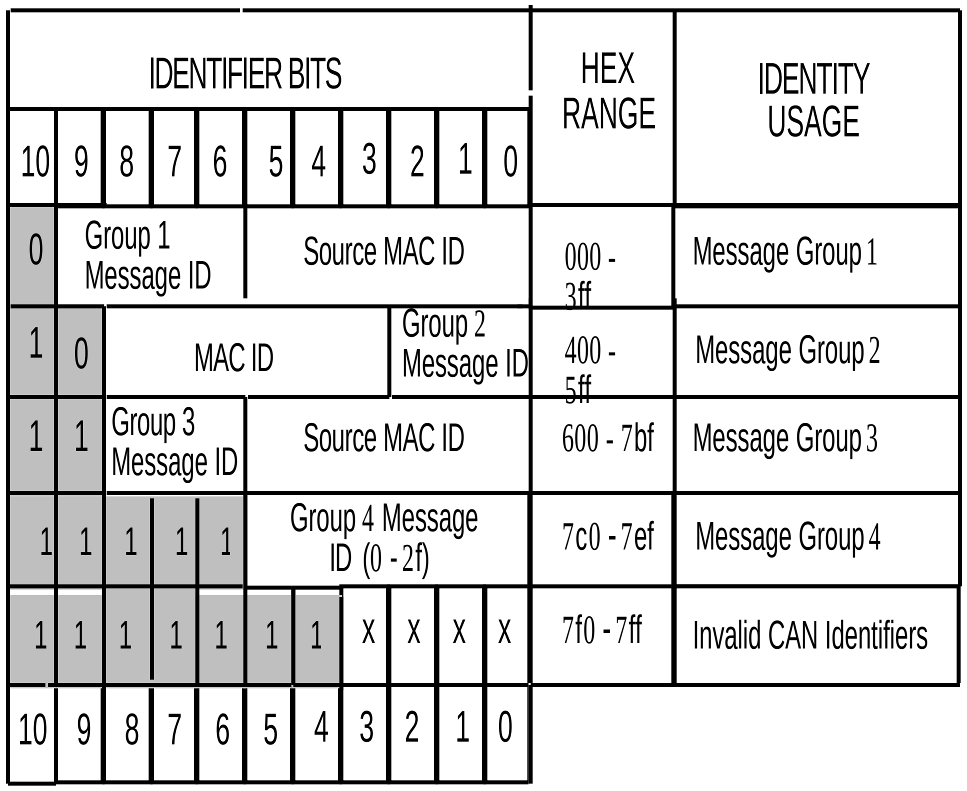
Figure 4 - Explicit Messaging Connection

A node can have multiple Connection Objects (instances of the Connection Object Class) active at any point in time. This enables the management of multiple communication relationships within a single device. Device developers must analyze the associated cost/performance trade-offs during development.

DeviceNet also defines a fragmentation protocol which provides for the exchange of messages (either I/O or Explicit) greater than 8 bytes.

DeviceNet CAN Identifier Field Definition

DeviceNet is based on Part A of the Bosch CAN Specification which defines an 11 bit identifier field. These 11 bits are subdivided into four separate message groups: Group 1, Group 2, Group 3, and Group 4 (See Figure 5).



IDENTIFIER BITS											HEX RANGE	IDENTITY USAGE
10	9	8	7	6	5	4	3	2	1	0		
0	Group 1 Message ID				Source MAC ID						000 - 3ff	Message Group 1
1	0	MAC ID						Group 2 Message ID			400 - 5ff	Message Group 2
1	1	Group 3 Message ID			Source MAC ID						600 - 7bf	Message Group 3
1	1	1	1	1	Group 4 Message ID (0 - 2f)						7c0 - 7ef	Message Group 4
1	1	1	1	1	1	1	X	X	X	X	7f0 - 7ff	Invalid CAN Identifiers
10	9	8	7	6	5	4	3	2	1	0		

Figure 5 - DeviceNet CAN Identifier Field Definition

The CAN Identifier Field on DeviceNet contains the following components:

> Message ID - Identifies a particular message within a particular Message Group. The value is given meaning via dynamic configuration, during product development, or by the DeviceNet Specification itself (Message Groups 2 and 3 reserve certain Message IDs for DeviceNet defined purposes).
>
> Source MAC ID - The MAC ID assigned to the device performing the transmission. Message Groups 1 and 3 require the specification of the Source MAC ID within the CAN Identifier Field.
>
> Destination MAC ID - The MAC ID assigned to the device which to receive this message. Message Group 2 allows the specification of either the Source or Destination within the MAC ID portion of the CAN Identifier Field.

Due to the CAN bus arbitration, the Groups provide for different bus access priorities. The Groups are prioritized so that Group 1 has the highest priority and Group 4 has the lowest priority. Within Groups 1 and Group 3, bus access priority is based on the Message ID and is evenly distributed among all the devices on the network. When two or more Group 1 or Group 3 Messages are arbitrating for CAN bus, the message with the numerically lower Message ID value will win arbitration and gain bus access. Within Group 2, bus access priority is based on the MAC ID. When two or more Group 2 Messages are arbitrating for the CAN bus, the message with the numerically lower MAC ID value will win arbitration and gain bus access. Use of Group 4 is currently reserved/TBD.

Both I/O and Explicit Messaging Connections can be established within any one of the Message Groups. This allows for the appropriate prioritization of information to be exchanged within the system. Although the DeviceNet communication model does not specify the types of connections that can be supported within each group, use of Group 1 for I/O connections, Group 2 for I/O and Explicit Message Connections, and Group 3 for Explicit Message Connections appears to be becoming a defacto standard.

Connection Establishment Overview

The main goal of DeviceNet is to provide for the run-time exchange of input and output information pertinent to a particular application. DeviceNet defines constructs for the full dynamic configuring I/O Connections between devices. It is important to note, though, that DeviceNet does not require any particular level of product support for the full dynamic configuration of connections. A particular product may configure a large percentage of its "connection information" during power-up such that a small amount of run-time configuration is necessary to open up the flow of I/O to/from that device.

The DeviceNet specification pre-defines a set of connections that facilitate the communications typically seen in a Master/Slave relationship. These connections are referred to as the Predefined Master/Slave Connection Set. The Predefined Master/Slave Connection Set removes many of the steps involved in the full, dynamic configuration of connections. This, in turn, presents the means by which a communication environment can be established using less network and device resources.

Dynamic Connection Configuration

Explicit Messaging Connections are dynamically established by the exchange of an *Unconnected* Explicit Request/Response Transaction. Unconnected Explicit Messages are used to perform Connection Management functions and are transmitted within Message Group 3 (See figure 6)

IDENTIFIER BITS											Message ID Meaning
10	9	8	7	6	5	4	3	2	1	0	
1	1	0	0	0	Source MAC ID						Group 3 Message Identifier
1	1	0	0	1	Source MAC ID						
1	1	0	1	0	Source MAC ID						
1	1	0	1	1	Source MAC ID						
1	1	1	0	0	Source MAC ID						
1	1	1	0	1	Source MAC ID						Unconnected Explicit Response Messages
1	1	1	1	0	Source MAC ID						Unconnected Explicit Request Messages
1	1	1	1	1	Source MAC ID						NOT USED WITHIN GROUP 3

Figure 6 - Detailed Group 3

As illustrated above, Message IDs 5 and 6 within Message Group 3 are reserved by DeviceNet. A Group 3 Message whose Message ID is 5 or 6 is processed by the *Unconnected Message Manager (UCMM)* within a device. The Unconnected Message Manager transmits/receives/processes messages that are not sent of a previously established connection.

Assume that the node whose MAC ID is 3 wants to establish an Explicit Messaging Connection with the node whose MAC ID is 4. The illustration below depicts the main components of the transaction executed to established the desired connection.

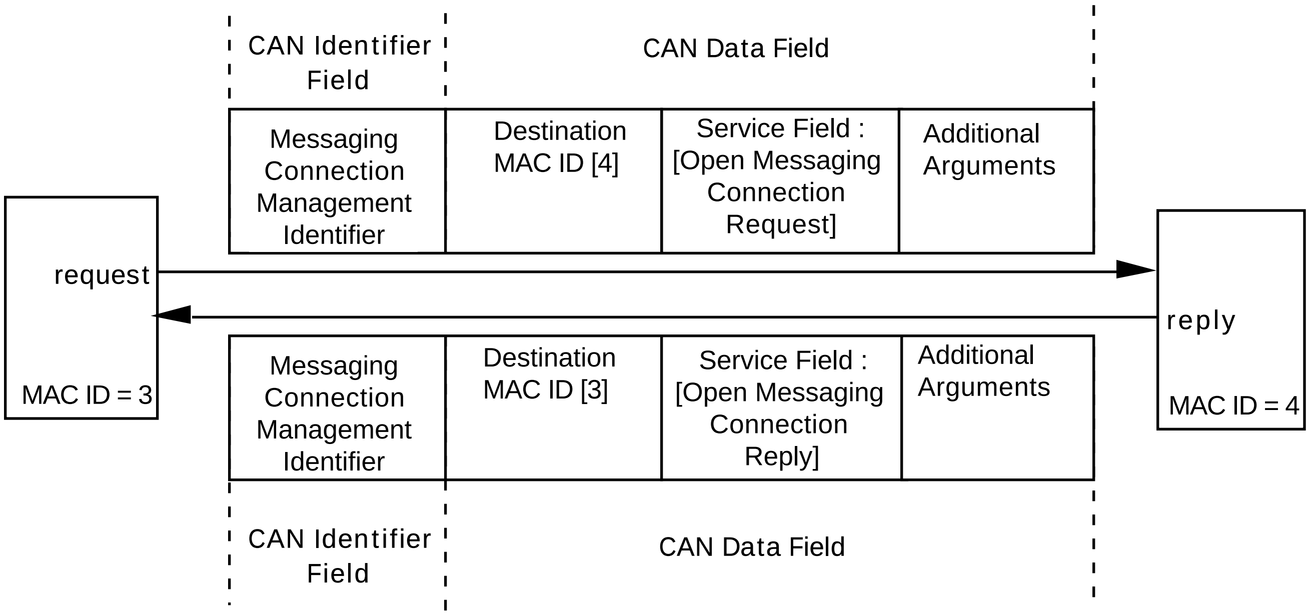

CAN Identifier Field
CAN Data Field
Messaging Connection Management Identifier
Destination MAC ID [4]
Service Field : [Open Messaging Connection Request]
Additional Arguments
request
MAC ID = 3
reply
MAC ID = 4
Messaging Connection Management Identifier
Destination MAC ID [3]
Service Field : [Open Messaging Connection Reply]
Additional Arguments
CAN Identifier Field
CAN Data Field

Figure 7 - Open Connection Request/Reply

Following the successful completion of this transaction, an Explicit Messaging Connection exists between the two devices (see Figure 2). Each device created and configured a Connection Object during this transaction.

I/O Connections are dynamically established by utilizing services available across an Explicit Messaging Connection. The following list presents the tasks necessary to dynamically establish and I/O connection:

1. Establish an Explicit Messaging Connection with one of the intended end-points of the I/O Connection
2. Create an I/O Connection Object by sending a Create Request to the Connection Class across the Explicit Messaging Connection established in step #1.
3. Configure the newly created I/O Connection Object using various Explicit Messaging Services.
4. Apply the configuration parameters loaded into the I/O Connection Object in step #3. This informs the end-point that the I/O Connection Object is now fully configured.
5. Repeat this process within the other end-point(s).

In summary, Explicit Messaging Connections are dynamically established by performing a DeviceNet defined Unconnected Explicit Messaging transaction. Parameters associated with this transaction enable both end-points to fully configure an Explicit Messaging Connection Object that is capable of exchanging Explicit Messages with the other end-point. I/O Connections are dynamically established by utilizing various services available across an Explicit Messaging Connections. Various requests and responses are exchanged across an Explicit Messaging Connection to create the desired I/O Connection Object within an end-point. This process is repeated with all devices required to participate in the I/O Connection.

Connection Object Attributes

A Connection Object provides attributes that control and/or indicate the following:

1. The Object's state.
2. The Object's type (I/O vs. Messaging)
3. The internal Application Object(s) associated with this Connection Object. This tells the node the information that is to be transmitted (produced) and/or where to put information when it is received (consumed).
4. Whether the Connection Object is producing only (transmitter), consuming only (receiver), or both producing and consuming.
5. If the Connection Object is supposed to produce, then the following information is also specified through Connection Object Attribute settings:
   - The event that triggers the production

- The value to place in the CAN Identifier Field (an Identifier Field formatted as defined in figure 5)
- The amount of data to be transmitted

6. If the Connection Object is supposed to consume, then the following information is also specified through Connection Object Attribute settings:
   - How often to expect the consumption and what to do if the message is not consumed in a timely fashion
   - The value that will be specified within the CAN Identifier Field of the message that is to be consumed
   - The amount of data that is to be received

These parameters are configured in establishing an I/O Connection.

Predefined Master/Slave Connection Set:

While DeviceNet provides a powerful Application Layer Protocol that allows for the dynamic configuring of connections between end devices, it has been recognized that some devices will neither have the need nor the resources to utilize this powerful capability. For this reason a set of connection identifiers known as the Predefined Master/Slave Connection Set have been specified to simplify the movement of the I/O and configuration type data typically seen in a Master/Slave architecture (see figure 8)**.**

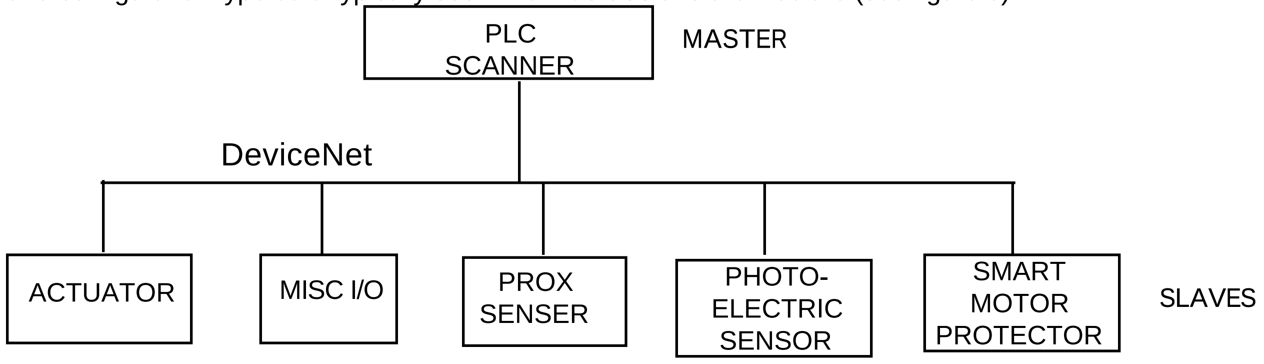


Figure 8 - Typical Master/Slave Architecture

Many, if not most, sensor/actuator devices are designed to perform some predetermined function (pressure sensor, motor starter, etc.) and as such the type and amount of data the device will produce and/or consume is known at power-up.

Typically these devices provide input data or require output data and they require configuration type data. The Predefined Master/Slave Connection set meets these needs by providing connection objects that are almost entirely configured at the time the device powers up. The only remaining step necessary to begin the flow of data is for a master device to "claim ownership of" this predefined connection cet within its slave(s). Figure 9 below reveals the identifiers associated with the predefined connections. From this figure we can see request and response identifiers have been specified for three connection instances.

IDENTIFIER BITS											Description
10	9	8	7	6	5	4	3	2	1	0	
0	Group 1 Message ID				Source MAC ID						Group 1 Messages
1	1	1	1	0	Source MAC ID						Slave's I/O Bit-Strobe Response Message
1	1	1	1	1	Source MAC ID						Slave's I/O Poll Response Message
1	0	MAC ID						Group 2 Message ID			Group 2 Messages
1	0	Source MAC ID						0	0	0	Master's I/O Bit-Strobe Command Message
1	0	Source MAC ID						0	0	1	Reserved for Master's Use -- Use is TBD
1	0	Source MAC ID						0	1	0	Reserved for Master's Use -- Use is TBD
1	0	Source MAC ID						0	1	1	Slave's Explicit Response Messages
1	0	Destination MAC ID						1	0	0	Master's Explicit Request Messages
1	0	Destination MAC ID						1	0	1	Master's I/O Poll Command Message
1	0	Destination MAC ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages
1	0	Destination MAC ID						1	1	1	Duplicate MAC ID Check Messages

Note that Group 2, Message ID=6 is reserved for use
as the Group 2 Only Unconnected Explicit Request message port.

Figure 9 - Predefined Master/Slave Connection Set Identifiers

The predefined connection instances are:

Explicit Messaging Connection: Used to transfer configuration type or other non-periodic data.
Poll I/O Connection: Used to transfer I/O data when polled by the master.
Bit Strobe I/O Connection: Used to transfer I/O data when strobed by the master

A master sends a Poll Message to each slave that supports the Poll I/O Connection. A Master sends 1 Strobe message that is received simultaneously by all slaves. A slave utilizing this predefined connection set must support the Explicit Messaging Connection and at least one of the I/O connections. If a slave supports the Poll I/O Connection it will produce/consume its I/O data when the master sends the Poll Command Message. If the slave supports the Bit Strobe Connection it will consume its output data (1 bit maximum) and produce its data. The fact that the Bit Strobe command message is a multicast message makes it highly efficient for moving small amounts of I/O data to actuator devices while, at the same time, triggering sensor devices to produce their input data.

The object model of a typical slave device is shown in Figure 10. The figure shows the three connection instances of the Predefined Master/Slave Connection Set and their relationship to the other objects contained within the slave's object model. The shaded objects are common to all DeviceNet devices while the non shaded objects will depend upon the application.

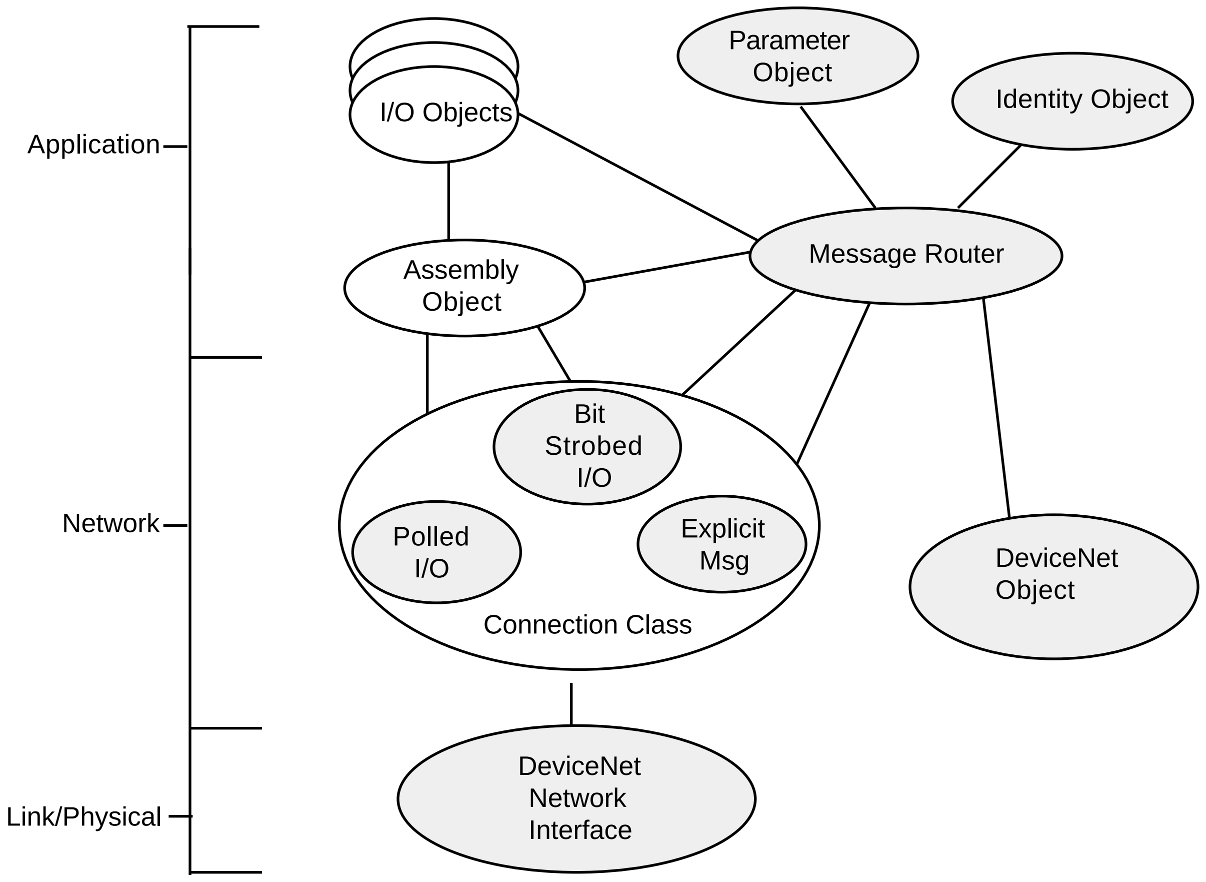


Figure 10 - Slave Device Object Model

As stated previously the whole purpose of the Predefined Master/Slave Connection Set is to provide an efficient, easily implementable version of the DeviceNet protocol. Supporting a bulky protocol may be too much to ask of low end (low cost) microprocessors. A sample implementation of a Slave device using the Predefined Master/Slave Connection Set resulted in the following:

ROM size = 2.5K bytes
RAM usage (depending upon configuration data size) = about 100 bytes
Communication Interrupt Latency = < 160 uSec

With resource requirements like these, even low end microprocessors can bring the power of networked technology to low cost sensor/actuator devices.

Conclusion:

DeviceNet supports a connection-based communication scheme that allows for the transfer of data and messaging traffic over CAN. It is designed to allow the implementation of both simple and complex devices, and supports many common communication paradigms (such as master/slave, multi-master, peer-to-peer). The DeviceNet communication architecture allows users to define the information sent between devices using the connection-based scheme.