DeviceNet™ Technical Overview

What is the DeviceNet?

DeviceNet is a low-cost communications link to connect industrial devices (such as limit switches, photoelectric sensors, valve manifolds, motor starters, process sensors, bar code readers, variable frequency drives, panel displays and operator interfaces) to a network and *eliminate expensive hardwiring.*

The direct connectivity provides improved communication between devices as well as important device-level diagnostics not easily accessible or available through hardwired I/O interfaces.

DeviceNet is a simple, networking solution that reduces the cost and time to wire and install industrial automation devices, while providing interchangeability of "like" components from multiple vendors.

DeviceNet is an *open network standard.* The specification and protocol are open — vendors are not required to purchase hardware, software or licensing rights to connect devices to a system. Anyone may obtain the DeviceNet Specification from the Open DeviceNet Vendor Association, Inc. (ODVA) for a nominal reproduction charge (currently $250 USD + postage). Any company that manufactures (or intends to manufacture) DeviceNet products may join ODVA and participate in technical working groups that are developing enhancements to the DeviceNet Specification.

Buyers of the DeviceNet Specification receive an unlimited, royalty-free license to develop DeviceNet products. Companies looking for assistance may purchase sample code that eases their implementation, development toolkits, and development services from many sources. The key hardware components are available from the largest worldwide suppliers of semiconductors.

Why the DeviceNet Communication Link?
For years the process industry has been attempting to develop a single, open standard to address all kinds of field devices. The original scope of their standards effort was aimed at replacing the 4-20 mA standard with a single digital standard. As the scope increased to address complex and sophisticated services (such as high data rate communications between controllers, time synchronization of large numbers of devices scanning at very high speeds), the development of a single standard became delayed.

At the same time, the cost of communication technology has dropped considerably in recent years, making it cost-effective to connect simple devices never considered for SP50 fieldbus directly to a network. Such a standard for simple devices requires the same level of interchange-ability as exists for 120/220 VAC and 24 VDC discrete, hardwired I/O. DeviceNet allows the interchangeability of simple devices while making interconnectivity of more complex devices possible. In addition to reading the state of discrete devices, DeviceNet provides the capability to report temperatures, to read the load current in a motor starter, to change the deceleration rate of drives, or to count the number of packages that have passed on a conveyor in the previous hour.

Controller Area Network (CAN) is the key to low cost products.
The DeviceNet communication link is based on a broadcast-oriented, communications protocol – the Controller Area Network (CAN). The CAN protocol was originally developed by BOSCH for the European automotive market for replacing expensive, wire harnesses with low-cost network cable on automobiles. As a result, the CAN protocol has fast response and high reliability for applications as demanding as control of anti-lock brakes and air-bags. Chips are available in a variety of packages with high temperature ratings and high noise immunity, attributes well suited for the industrial automation market as well.

But it is consumer and commercial demand for CAN that is the key driver in lowering the price and increasing the performance of CAN chips. In 1994, four suppliers of CAN chips (Intel, Motorola, Philips, Siemens) shipped 4+ million CAN chips. Over 10 million chips are expected to ship in 1996. Whereas other industrial automation networks use custom chips with annual demand varying from 20,000–200,000 per year, DeviceNet products use the same CAN chips as are used in automotive and other consumer/commercial applications. The chips for DeviceNet products are typically 5–10 times less than chips for other networks.

DeviceNet Features and Functionality

Network Size	Up to 64 nodes	
Network Length	Selectable end-to-end network distance varies with speed	
	Baud Rate	**Distance**
	125 Kbps	500 m (1,640 ft)
	250 Kbps	250 m (820 ft)
	500 Kbps	100 m (328 ft)
Data Packets	0-8 bytes	
Bus Topology	Linear (trunkline/dropline); power and signal on the same network cable	
Bus Addressing	Peer-to-Peer with Multi-Cast (one-to-many); Multi-Master and Master/Slave special case; polled or change-of-state (exception-based)	
System Features	Removal and replacement of devices from the network under power	

What is the DeviceNet Specification?

The DeviceNet Specification defines a network communication system for moving data between elements of an industrial control system. The specification is divided into two volumes and defines the following elements:

Volume 1
- DeviceNet Communication Protocol and Application (Layer 7 - Application Layer)
- CAN and its use in DeviceNet (Layer 2 - Data Link Layer)
- DeviceNet Physical Layer and Media (Layer 1 - Physical Layer)

Volume 2
- Device Profiles to obtain interoperability and interchangeability among like products

DeviceNet incorporates CAN (Controller Area Network). CAN defines the syntax or form of the data movement. The DeviceNet application layer defines the semantics or meaning of the data moved.

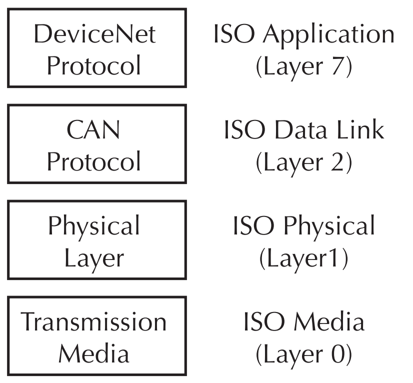

Figure 1. DeviceNet is an Application Layer Protocol (ISO Layer 7)

Communication Protocol Features
- *Peer-to-Peer* data exchange in which any DeviceNet product can produce and consume messages
- *Master/Slave* operation defined as a proper subset of Peer-to-Peer
- A DeviceNet product may behave as a Client or a Server or both
- A DeviceNet network may have up to 64 *Media Access Control Identifiers* or *MAC IDs* (node addresses). Each node can support an infinite number of I/O. Typical I/O counts for pneumatic valve actuators are 16 or 32.

The Object Model
A DeviceNet node is modeled as a collection of *Objects.* An object provides an abstract representation of a particular component within a product. The realization of this abstract object model with a product is implementation dependent .

An *Object Instance* and an *Object Class* have *Attributes* (data), provide *Services* (methods or procedures), and implement *Behaviors*. *Attributes* (1-255), *Instances* (0-6535), *Class* (1-65535) and *Node Address* (0-63) are addressed by number.

DeviceNet Physical Layer and Media

Chapter 9, Volume 1 in the DeviceNet Specifications defines the allowable topologies and components. The variety of topologies that are possible is shown in Figure 2. The specification also deals with system grounding, mixing thick and thin media, termination, and power distribution.
The basic trunkline-dropline topology provides separate twisted pair busses for both signal and power distribution. Thick or thin cable can be used for either trunklines or droplines. End-to-end network distance varies with data rate and cable size (see table on page 4).

Devices can be powered directly from the bus and communicate with each other using the same cable. *Nodes can be removed or inserted from the network without powering-down the network.*

Power taps can be added at any point in the network which makes redundant power supplies possible. The trunkline current rating is 8 amps. An opto-isolated design option allows externally powered devices (e.g. AC Drives starters and solenoid valves) to share the same bus cable. Other CAN-based networks allow only a single power supply (if at all) for the entire network.

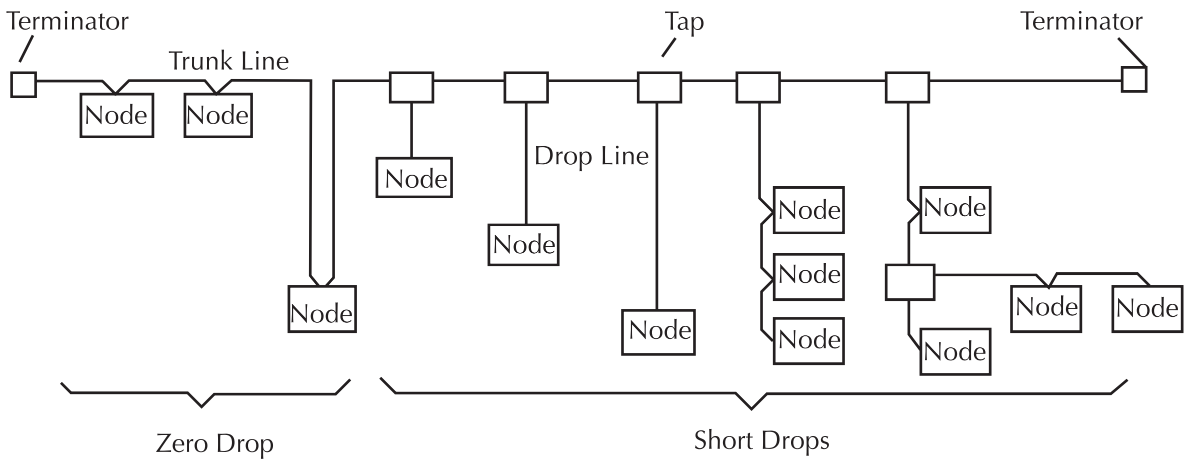

Figure 2. Thick or Thin Cable can be used for either trunklines or droplines

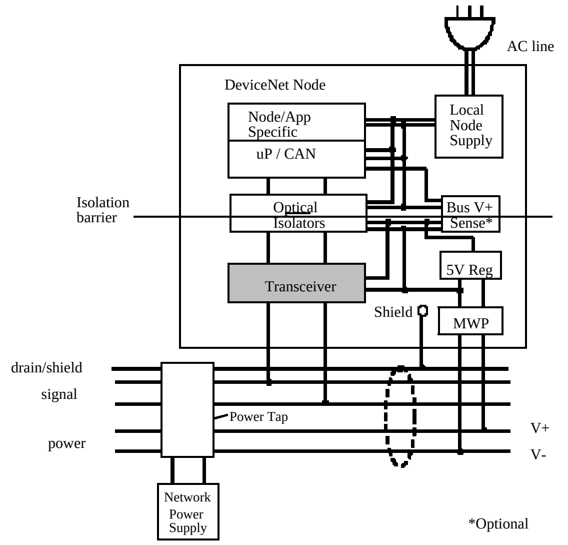

Figure 3. Isolated Physical Layer

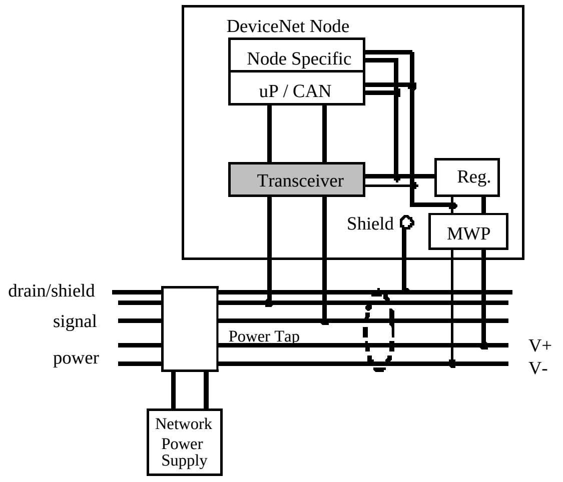

Figure 4. Non-Isolated Physical Layer

A simplified block diagram of transceivers with both isolated and non-isolated physical layers is shown in Figures 3 and 4. The DeviceNet Specifications contain additional information concerning component requirements, protection from mis-wiring, and examples.

Data Rates	125 Kbps	250 Kbps	500 Kbps
Thick Trunk Length	500 m (1,640 ft)	250 m (820 ft)	100 m (328 ft)
Thin Trunk Length	100 m (328 ft)	100 m (328 ft)	100 m (328 ft)
Maximum Drop Length	6 m (20 ft)	6 m (20 ft)	6 m (20 ft)
Cumulative Drop Length	156 m (512 ft)	78 m (256 ft)	39 m (128 ft)

The end-to-end network distance varies with data rate and cable thickness.

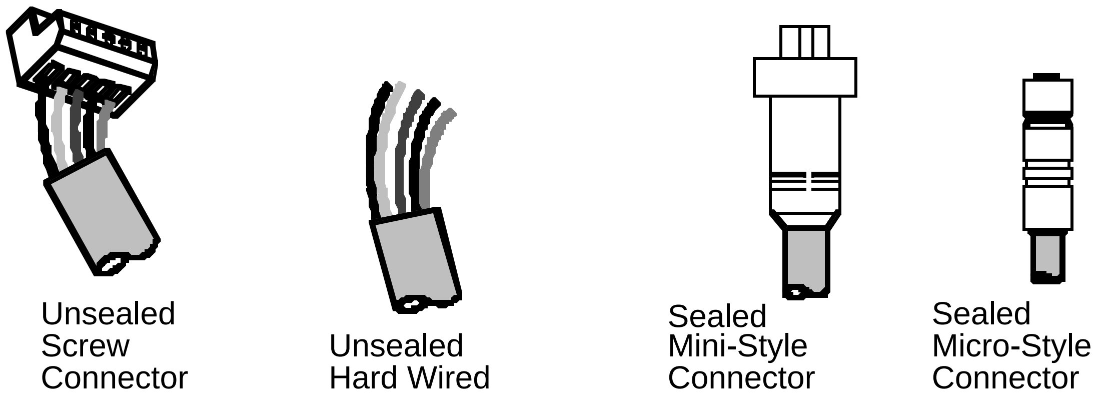

Figure 5. Open and Sealed Connectors are available on DeviceNet

Several different connector types can be used on DeviceNet (see Figure 5). Both sealed and unsealed connectors are available. Large (mini-style) and small (micro-style) sizes of pluggable, sealed connectors are available. For products which do not require sealed connectors, open-style connectors can be used. Screw or clamp connections can be made directly to the cable if a pluggable connection is not required. The DeviceNet Specification also contain information on how to use these cable and connector components to construct single and multi-port taps.

Indicators and Configuration Switches

Although DeviceNet does not require a product to have indicators, if a product does have indicators, it must adhere to the DeviceNet Specification. It is recommended that either a *Module Status LED* and a *Network Status LED,* or the combined *Module Status/Network Status LED* be included.

The indicator(s) consist of bi-color (green/red) LEDs which can have combinations of on, off or flashing. The Module Status LED indicates whether or not the device has power and is operating properly. The Network Status LED indicates the status of the communication link.

CAN and DeviceNet

The Data Link Layer of DeviceNet is completely defined by the CAN specification and by the implementation of CAN Controller chips. The CAN specification defines two bus states called dominant (logic 0) and recessive (logic 1). Any transmitter can drive the bus to a dominant state. The bus can only be in the recessive state when no transmitter is in the dominant state.

Several frame types are defined by CAN:

- data frame
- remote frame
- overload frame
- error frame

Data is moved on DeviceNet using the data frame. The other frames are either not used on DeviceNet or are for exception handling. The data frame format is shown in Figure 6.

Higher priority data gets the right-of-way
DeviceNet is similar to Ethernet in that any DeviceNet node can attempt to transmit if the bus is quiet.

This provides inherent peer-to-peer capability. If two or more nodes try to access the network simultaneously, *a bit-wise non-destructive arbitration mechanism* resolves the potential conflict with no loss of data or bandwidth. By comparison, Ethernet uses collision detectors which result in loss of data and bandwidth as both nodes have to back-off and resend their data.

CAN uses a unique, non-destructive bit-wise arbitration mechanism. This CAN-specific feature allows resolution of collisions (determination of a "winner") without loss of throughput or resending of data by the higher priority node.

CAN uses a *bit-wise arbitration* method of collision resolution. All receivers on a CAN network synchronize to the transition from recessive to dominant represented by a *(Start of Frame)* bit. The identifier and the *RTR* (Remote Transmission Request) bit together form the *Arbitration Field*. The Arbitration Field is used to facilitate media access. Since DeviceNet does not use the RTR bit for any purpose it does not enter into bus access priority consideration. When a device transmits, it also monitors (receives) what it sends to make sure it is the same. This allows detection of simultaneous transmission. If a node transmitting a recessive bit receives a dominant bit while sending the arbitration field, it stops transmitting. The winner of an arbitration between two nodes transmitting simultaneously is the one with the lower numbered 11-bit identifier. CAN also specifies a data frame format with a 29-bit identifier field which is not used by DeviceNet.

The *Control Field* contains two fixed bits and a 4-bit length field. The length may be any number from 0 to 8 representing the number of bytes in the *Data Field*. The 0–8 byte size is ideal for low-end devices with small amounts of I/O data that must be exchanged frequently. And, at eight bytes, there is enough flexibility for simple devices to send diagnostic data, or to send a speed reference and acceleration rate to a drive.

The *CRC* field is a cyclic redundancy check word which is used by CAN controllers to detect frame errors. It is computed from the bits that come before it. A dominant bit in the ACK slot means at least one receiver besides the transmitter heard the transmission.

CAN uses several types of error detection and fault confinement methods including CRC and automatic retries. These methods, which are mostly transparent to the application, prevent a faulty node from disrupting the network.

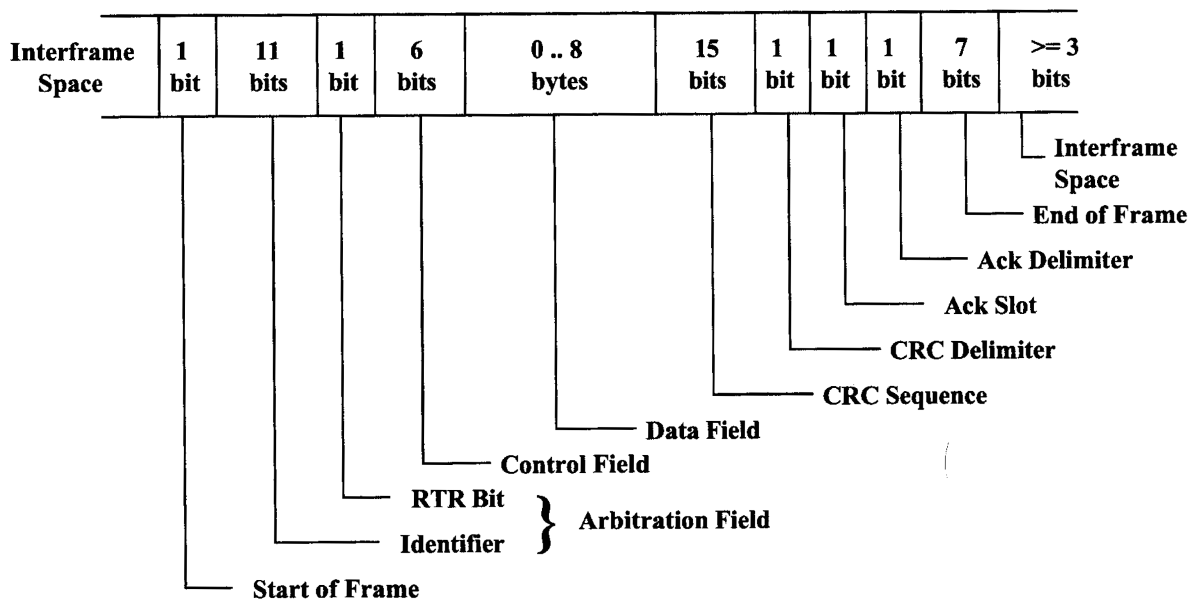

Figure 6. CAN Data Frame

CAN Parts Summary for DeviceNet

Most slaves use Intel 82527 or Motorola 68HC705X4. Most masters/peers use Philips 82C200.

Mfg	Part #	Type	Package	Screeners	Miscellaneous Features	DeviceNet Application
Intel	82527	Peripheral	44PLCC	14 fixed 1 mask/match	2 8-bit I/O ports	Fully Compatible
	8xC196CA	Embedded	68PLCC	14 fixed 1 mask/match	Standard 196 w/CAN; 32K ROM, 1K RAM, 6 A/Ds, 4 Timers	Fully Compatible
Motorola	68HC05X4 68HC705X4	Embedded	28SOIC	1 mask/match	Basic CAN w/6805 core; 4K ROM,176 RAM	Fully Compatible, CPU must filter all IDs unless Group 2 only slave
	68HC705X32 68HC05X32	Embedded	64 QFP	1 mask/match	Basic CAN w/6805 core; 32K ROM, 528 RAM	Fully Compatible CPU must filter all IDs unless Group 2 only slave
	68HC05X16	Embedded	64QFP	1 mask/match	Basic CAN w/6805 core; 16K ROM, 352 RAM	
Philips	SJA1000/N1 SJA1000/TN1	Peripheral Peripheral	28 DIP 28SOTC	1 mask/match full CAN peripheral	Basic CAN w/6805 core; 4K ROM, 176 RAM	Fully Compatible CPU must filter all IDs
	82C250	Transceiver	8 SOIC 8 DIP	n/a	ISO 11898 CAN Physical Layer	Fully Compatible
	82C251	Transceiver	8 SOIC 8 DIP	n/a	ISO 11898 CAN Physical Layer	Fully Compatible 25V overvoltage protection
	8xC592	Embedded	68 PLCC	1 mask/match	CAN; 16K ROM, 512 RAM; 8-ch/10-bit A/D	CPU must filter all IDs unless Group 2 only slave
	8xCE598	Embedded	80 QFP	1 mask/match	8031 core w/basic CAN; 32K ROM, 512 RAM; 8-ch /10-bit A/D	Fully Compatible CPU must filter all IDs unless Group 2 only slave
Siemens	SAB-C167C	Embedded	144 QTP	14 fixed 1 mask/match	Full CAN w/80C167 core; 2*2K RAM; 16 ch/10-bit A/D	Fully Compatible
	SAB-81C90 SAB-81C91	Peripheral	44 PLCC 28 PLCC	16 fixed	Full CAN; C90 has 8-bit I/O ports	Fully Compatible CPU must filter all IDs unless Group 2 only slave
	SAB-C515 8R SAB-C515-L	Embedded	MQFP	14 fixed 1 mask/match	Full CAN w/8031 core; 64K ROM; 2*.25K RAM; 56 I/O; 8 ch/10-bit A/D	Fully Compatible
Unitrode	UC5350	Transceiver	8SOIC		ISO 11898 CAN Physical layer	Cross wire & gound loss protection
Siliconix	si9200DY	Transceiver	8 SOIC	n/a	No slew control	Not compatible
TI	SN65LBC031 SN75LBC031	Transceiver	8 SOIC 8 DIP	n/a	ISO 11898 CAN physical layer	Not compatible
Bosch	CF15B CF15O	Transceiver	8 SOIC	n/a	w/o slew control w/ slew control	Not compatible

CAN References

[1] Anonymous, MC68HC05X4 HCMOS Microcomputer Unit, Motorola LTD, 1992.
[2] Terry, K., Software Driver Routines for the Motorola MC68C05 CAN Module (AN464), Motorola LTD., 1993
[3] Anonymous, 80c51 - Based 8-Bit Microcontrollers, Data Handbook IC20, Philips, 1995.
[4] Anonymous, 82527 Serial Communications Controller Architectural Overview, Intel Corporation, February, 1995, Order Number: 272410-002
[5] Anonymous, 8227 Serial Communications Controller, Controller Area Network Protocol, Intel Corporation, December, 1995, Order Number: 272250-006
[6] Anonymous, 87C196CA/87C196CB Advanced 16-Bit CHMOS Microcontroller with Integrated CAN 2.0, Intel Corporation, October, 1993, Order Number: 272405-002

[7] BOSCH CAN Specification — Version 2.0, Part A. 1991, Robert Bosch GmbH
[8] ISO 11898: 1993 – Road vehicles – Interchange of digital information – Controller area network (CAN) for high–speed communication

Communication Protocol and Application

Chapters 3, 4 and 5, Volume 1 of the DeviceNet Specification defines the DeviceNet Communication Protocol. These chapters deal with connections, messaging protocol, and the communications related objects respectively.

Applications using DeviceNet combine standard or application specific objects together into what is called a *Device Profile.* The Device Profile fully defines the device as viewed from the network. A library of objects is contained in Chapter 6, Volume II of the DeviceNet Specifications. A library of Device Profiles is contained in Chapter 3, Volume II of the DeviceNet Specifications. ODVA coordinates the work of industry experts in the development of both new Object and Device Profile Specifications. This is done through *Special Interest Groups (SIGs).*

DeviceNet supports *strobed, polled, cyclic, change-of-state* and application-triggered data movement. The user can choose *master/slave, multi-master* and *peer-to-peer* or a combination configuration depending on device capability and application requirements. The choice of data movement can significantly speed up system response time. One popular application for DeviceNet is to use a standard, pre-defined set of connections which allow devices to operate in a *Master/Slave Connection Set.*

Connections

The DeviceNet Communication Protocol is based on the idea of connections. You must establish a connection with a device in order to exchange information with that device.

To establish a connection, each DeviceNet product will implement either an *Unconnected Message Manager (UCMM)* or an *Unconnected Port.* Both perform their function by reserving some of the available CAN identifiers. The UCMM is described in detail in Chapter 4, Volume 1 of the DeviceNet Specification, and the Unconnected Port is described in Chapter 7, Volume 1.

When either the UCMM or the Unconnected Port is used to establish an *Explicit Messaging Connection,* that connection is then used to move information from one node to the other, or to establish additional *I/O Connections.* Once connections have been established, I/O data may be moved among devices on the network. At this point, all the protocol of the DeviceNet I/O message is contained within the 11-bit CAN identifier. Everything else is data.

The 11-bit CAN identifier is used to define the connection ID. DeviceNet divides the 11-bit CAN identifier into four groups. The first three defined groups contain two fields, —one 6-bit field for MAC ID and the other for Message ID. The combined fields define the connection ID. Figure 7 shows message group definitions. Group four messages are used for offline communications.

Devices may be *Clients* or *Servers* or *both.* Clients and Servers may be *producers, consumers* or *both.* In a typical *Client device,* its connection would *produce* requests and *consume* responses. In a typical *Server* device, its connections would *consume* requests and *produce* responses. DeviceNet provides for several variations on this model. Some connections in either a *Client* or a *Server may* only *consume* messages. These connections would be the destination for *Cyclic* or *Change-of-State* messages. Similarly, some connections in either a *Client* or *Server* may only *produce* messages. These connections would be the source for *Cyclic* or *Changes-of-State* messages. The use of *Cyclic* and *Change-of-State* connections can substantially reduce bandwidth requirements.

By design, nodes in a DeviceNet system are responsible for managing their own identifiers. These identifiers are interleaved (distributed) throughout the entire range. All nodes have a full range of message priorities available to them regardless of their MAC ID. Through the duplicate MAC ID algorithm, the uniqueness of CAN identifiers is guaranteed without the need for a central tool or record for each network.

A related issue is detection of duplicate nodes. Because DeviceNet uses a device address inside the CAN Identifier Field, it presents a mechanism for detecting duplicate addressed devices. Preventing duplicate addresses is better than trying to locate them after they occur — something not taken into account in other CAN-based networks.

INDENTIFIER 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)						7c0-7ff	Message Group 4
1	1	1	1	1	1	1	x	x	x		7f0-7ff	Invalid CAN Identifiers
10	9	8	7	6	5	4	3	2	1	0		

Figure 7. DeviceNet has 4 defined message groups.

Another key benefit to nodes managing their identifiers is that a user can add and delete nodes and/or add additional peer-to-peer messages among existing nodes at any time without having knowledge of the existing set-up. No centralized record must be located or reconstructed. Since nodes know which IDs are already in use, a tool simply has to request an I/O connection be added between the two devices, specifying priority level, the data path (class, instance, attribute) and the production trigger (cyclic, poll, or change-of-state).

The Object Model

The Object Model provides a template for organizing and implementing *the Attributes* (data), *Services* (methods or procedures) and *Behaviors* of the components of a DeviceNet product (see Figure 8). The model provides an addressing scheme for each Attribute consisting of four numbers. They are the Node Address (MAC ID), the *Object Class Identifier*, the Instance Number, and the *Attribute Number.* This four-level address is used in conjunction with an *Explicit Messaging Connection to* move data from one place to another on a DeviceNet network. The ranges of the four addressing components are shown in the following Table:

Address	Lowest	Highest
Node	0	63
Class	1	65535
Instance	0	65535
Attribute	1	255

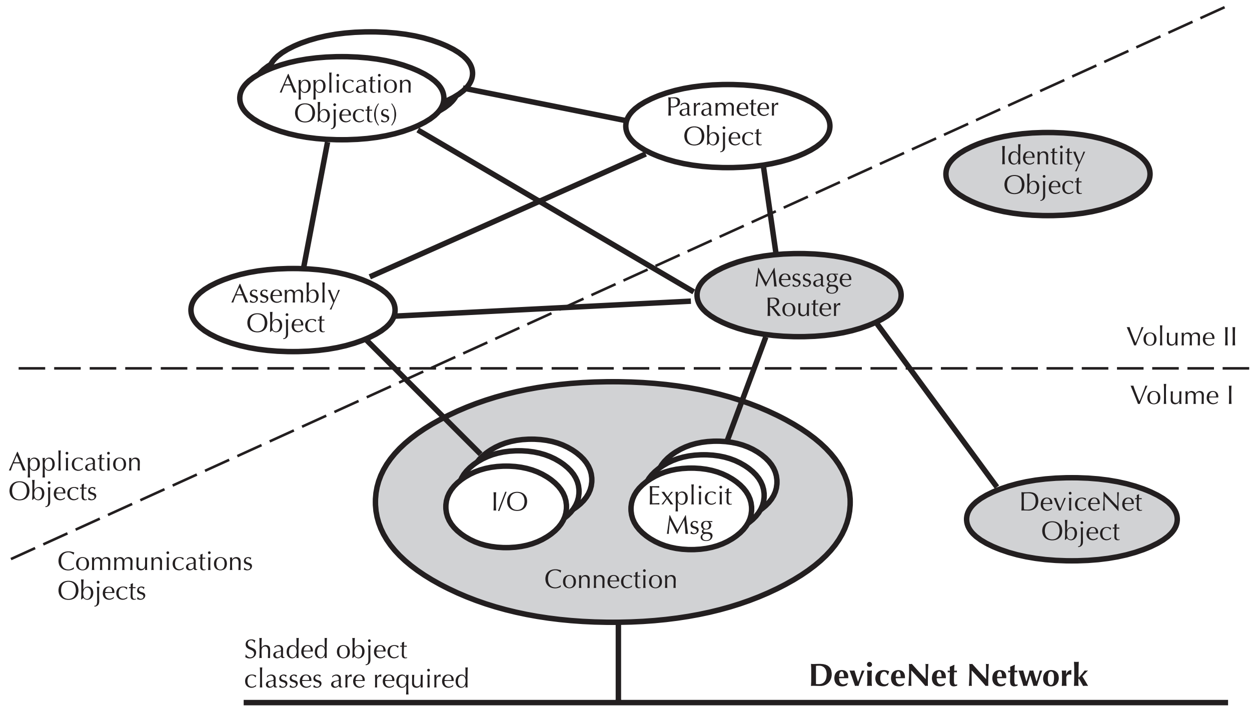

Figure 8. DeviceNet Object Model

Typical Object Classes found in a DeviceNet Product

Object Class Number	Object Class Name	Specification Reference
1	Identity	Vol II, Rel 1.2, page 6-3
2	Message Router	Vol II, Rel 1.2, page 6-17
3	DeviceNet	Vol I, Rel 1.3, page 5-50
4	Assembly	Vol II, Rel 1.2, page 6-25
5	Connection	Vol I, Rel 1.3, page 5-6
6	Parameter	Vol II, Rel 1.2, page 6-95

Identity Object –Vol II, Rel 1.2, page 6-3
A DeviceNet product will typically have a single instance (Instance #1) of the Identity Object. This instance will have as attributes a Vendor ID, a Device Type, a Product code, a revision, a status, a serial number, a product name, and a state. The required services would be *Get_Attribute_Single and a Reset.*

Message Router Object –Vol II, Rel 1.2, page 6-17
A DeviceNet product will typically have a single instance (Instance #1) of the Message Router Object. The Message Router Object is the component of a product that passes Explicit *Messages* to the other Objects. It generally does not have any external visibility over the DeviceNet network.

DeviceNet Object – Vol I, Rel 1.3, page 5-50
A DeviceNet product will typically have a single instance (Instance #1) of the DeviceNet Object. This instance would have as attributes: Node Address or MAC ID, baudrate, Bus-Off action, Bus-Off counter, the allocation choice, and the master's MAC ID. The only required service is *Get_Attribute_Single.*

Assembly Object(s) – Vol II, Rel 1.2, page 6-25
A DeviceNet product will typically have one or more optional Assembly Objects. The primary purpose of these objects is to group different Attributes (data) from different application objects into a single Attribute which can be moved with a single message.

Connection Objects – Vol I, Rel 1.3, page 5-6
A DeviceNet product will typically have at least two connection objects. Each connection object represents one end point of a virtual connection between two nodes on a DeviceNet network. These two types of connections are called Explicit *Messaging and I/O Messaging*. Explicit Messages contain Attribute addressing, Attribute values and a Service Code describing the desired action. I/O messages contain nothing but data. In an I/O message, all the information about what to do with the data is contained in the Connection Object associated with that I/O message.

Parameter Object – Vol II, Rel 1.2, page 6-95
The optional Parameter object would be used in devices with configurable parameters. One instance would be present for each configurable parameter. The parameter object provides a standard way for a configuration tool to access all parameters. Configuration options which are attributes of the Parameter object could include values, ranges, text strings, and limits.

Application Objects
Usually at least one application object besides those from the Assembly or Parameter Class will be present in a device. There are a number of standard objects in the DeviceNet Object Library in Chapter 6, Volume II.

Messaging

The DeviceNet application layer defines how identifiers are assigned (thus controlling priorities), and how the CAN data field is used to specify services, move data, and determine its meaning.

The way information flows on a communication network is critical. Older communication technology consisted of messages that were constructed with a specific source and destination. Instead of a traditional source-destination approach, DeviceNet uses a more efficient *Producer-Consumer Model* which requires packets to have identifier fields for the data. The identifier provides the means for multiple priority levels (used in arbitration), more efficient transfer of I/O data, and multiple consumers.

The device with data *produces* the data on the network with the proper identifier. All devices that need data listen for messages. When devices recognize the appropriate identifier, they consume the data. With the producer-consumer model, the message is no longer specific to a particular source or destination. A single message from one controller can be used by multiple motor starters using less bandwidth.

DeviceNet defines two different types of messaging. They are called *I/O Messaging* and *Explicit Messaging.*

I/O messages are for time-critical, control-oriented data. They provide a dedicated, special-purpose communication path between a producing application and one or more consuming applications. They are exchanged across single or multi-cast connections and typically use high priority identifiers. I/O messages contain no protocol in the 8-byte data field. The only exception is for fragmented I/O messages where one byte is used for fragmentation protocol. The meaning of the message is implied by the connection ID (CAN identifier). Before messages are sent using these IDs, both the device sending and receiving them must be configured. The configuration contains the source and destination object attribute addresses for the producer and consumer of the data.

Explicit messages provide multi-purpose, point-to-point communication paths between two devices. They provide the typical request/ response-oriented network communications used to perform node configuration and problem diagnosis. Explicit messages typically use low priority identifiers and contain the specific meaning of the message right in the data field. This includes the service to be performed and the specific object attribute address.

Fragmentation services are provided for messages that are longer than 8 bytes. Each I/O Message *fragment* incurs only a single byte of protocol overhead. There is no limit on the number of fragments. Fragmentation is also defined for explicit messaging. This flexibility assures that as more sophisticated devices are introduced and more capabilities are designed into devices, they can be added to existing DeviceNet networks. With its object-oriented design and addressing scheme, DeviceNet is unlimited in its ability to expand without having to alter the basic protocol and connection model.

On the other end of the spectrum, a simple slave device application with two message connections (one I/O and one explicit) can be handled in less than 4K ROM and 175 bytes of RAM (Motorola 68HC05X4, a CPU with a built-in CAN interface).

The general format for I/O messages is shown in Figures 9-12.

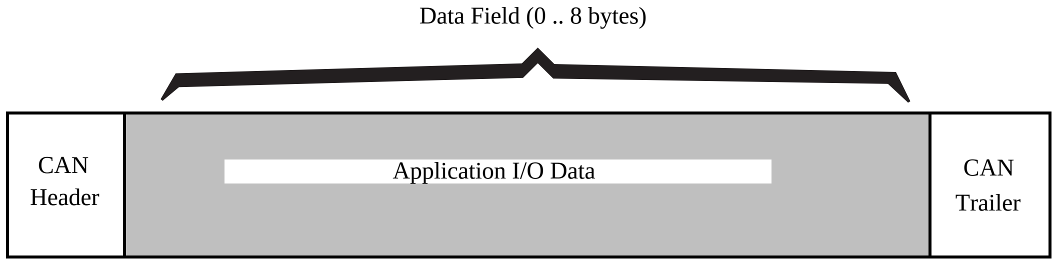

Figure 9. Format for I/O Message

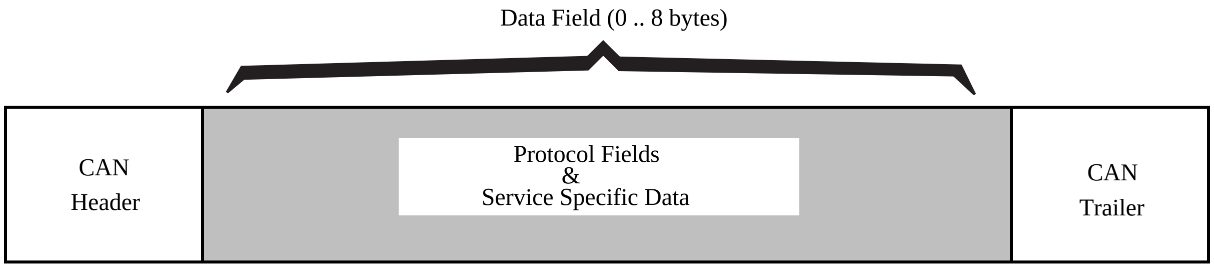
Data Field (0 .. 8 bytes)
CAN Header
Protocol Fields & Service Specific Data
CAN Trailer

Figure 10. Format for Explicit Message

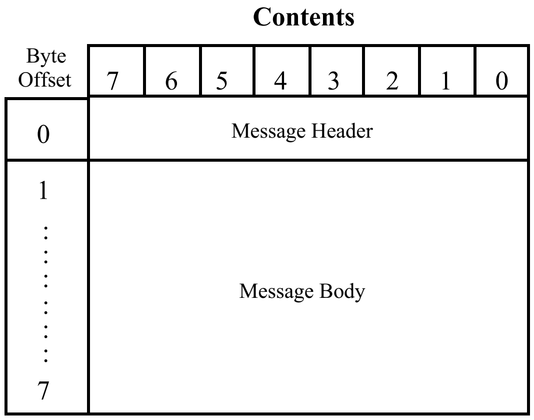

Figure 11. Non-Fragmented Explicit Message Data Field Format

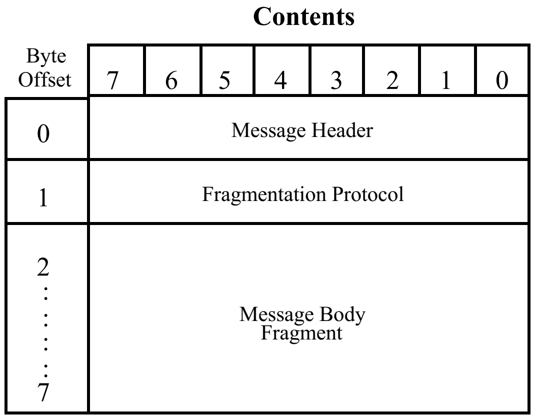

Figure 12. Fragmented Explicit Message Data Field Format

Predefined Master/Slave Connection Set

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

Many sensors and actuators are designed to perform some predetermined function (sense pressure, start motor, etc.) and 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 configuration 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 set within its slave(s).

Message Group 2 is used for the definition of these identifiers (refer back to Figure 7). One noticeable difference in Group 2 is that the MAC

ID is not specified as Source MAC ID. This allows the use of Destination ID. There are strict rules about the use of this kind of connection to prevent duplicate CAN identifiers on the bus. The use of Destination ID allows devices which are centralized and which must communicate with many nodes (a master) to borrow identifiers from those nodes. In addition, the MAC ID and Message ID fields are reversed. This allows the Group ID and MAC ID to fall within the most significant 8 bits of the CAN identifier. This is important because many low-cost, 8-bit CAN chips can hardware filter only the first 8 bits. The exclusive use of Destination MAC ID further allows devices to take advantage of hardware filtering. Another important benefit is that the establishment of connections from the Predefined Set is simplified considerably. Only a few messages are required to have I/O connections up and running. The Predefined Set contains one explicit messaging connection and allows several different I/O connections including *Bit Strobed Command/Response*, *Polled Command/Response*, *Change-of-State* and *Cyclic*. Chapter 7, Volume 1 contains detailed information about the Predefined Master/Slave Connection Set.

Change-of-State and Cyclic Transmission

With change-of-state, a device produces its data only when it changes. To be sure the consuming device knows that the producer is still alive and active, DeviceNet provides an adjustable, background *heartbeat rate*. Devices send data whenever it changes or the heartbeat timer expires. This serves to keep the connection alive and let the consumer know his data source has not faulted in some way. The minimum time on the heartbeat prevents inherently noisy nodes from dominating the network. By having the device generate the heartbeat, the controller is not burdened with having to send a nuisance request periodically just to make sure it is still there. This becomes even more efficient in the multicast case.

The cyclic option can reduce unnecessary traffic and packet processing. Instead of a temperature or analog input block being scanned dozens of times every second, it can be set up to report its data on a regular basis consistent with the rate of change it can detect. A temperature sensor on a slow PID loop with a 500 ms update time could have its cyclic rate set to 500 ms. Not only would this preserve bandwidth for more rapidly changing critical I/O data, it would also be more accurate as well. For example, it might be scanned once every 30 ms as part of a large scan list with many bytes of data per node on a heavily loaded master. This means that data used in a PID calculation might have been sampled anywhere from 470 to 530 ms. With cyclic production you know that the data samples will be at precisely 500 ms.

By default, both change of state and cyclic are *acknowledged exchanges* (ACKs) so that the producer knows its intended consumer(s) received the data. For applications where changes of state or cyclic rates are extremely fast, it makes no sense to clutter up the network with ACK packets. Unnecessary ACKs can be suppressed with the *Acknowledge Handler Object* (Chapter 6, Volume II, Rel 1.2, pages 6-252 to 6-272).

Now, even simple slave nodes can be set up to report at the most appropriate interval, whether that be cyclic or change-of-state. With the ACK Handler Object it is possible to have multiple consumers of the slaves' data, not just the master. This multicast is especially useful for operator interface (OI) devices which can just listen for the data they need, whether it is for display, alarm monitoring or data logging.

For alarm monitoring, it is important that a change-of-state not be missed, so the OI device would be included in the device's ACK list, assuring a retry (or several retries) if for some reason the OI missed the message. On the other hand, if it was primarily a data collection device logging values every 5 seconds (and the node is producing values every 300 ms for control purposes), you would not have the logger set to ACK. If the logger misses a value, it can grab the next one 300 ms later.

Cyclic and change-of-state from the master is defined and selectable on a per node basis.

Device Profiles

The DeviceNet specification goes beyond a physical connection protocol specification. It promotes interoperability by defining standard device models. All devices of the same model must support common identity and communication status data. Device-specific data is contained in device profiles that are defined for various device types. Simple devices (e.g., push buttons, motor starters, photocells, pneumatic valve actuators) from multiple vendors that comply with their device type profile will be logically interchangeable.

A device profile will contain the following sections:

- Definition of the device's object model - This often contains a drawing like the one shown in the object model (Figure 8, page 10). Usually there are tables which list all of the object classes present in the device, the number of instances in each class, how each object effects behavior, and the public interfaces to each object.

- Definition of the device's I/O data format - This usually includes definition of Assembly Object definition which contains the address (class, instance, and attribute) of the desired data components.

- Definition of the device's configurable parameters and public interfaces to those parameters. This information is contained in an *Electronic Data Sheet* (*EDS*) which is included with the device's user documentation.

As an example, a photoelectric sensor (Device Type 6) would contain the following objects:

• Identity Object	1
• Message Router	1
• DeviceNet	1
• Connection	2 (one explicit, one I/O)
• Assembly	1
• Parameter	1 (optional)
• Presence Sensing	1

The DeviceNet specification defines an *Electronic Data Sheet* (*EDS*) which is a simple file format that allows product-specific information to be made available by vendors for all other vendors. This makes possible user-friendly configuration tools that can be easily updated without having to constantly revise the configuration software tool.

In order not to restrict enhancements, a means is also provided for addition of vendor specific, value-add options. The DeviceNet specification has *public* classes, services and attributes which are defined in the DeviceNet specification and *vendor-specific* classes, services and attributes which can be added by individual vendors. This allows vendors to provide additional functions to their customers that are not covered in the specification. As vendor-specific items become more popular or commonplace, the mechanism to transition them to public items will be provided.

DeviceNet Specifications
Table of Contents