Identity Object – 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 – 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 – 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, baud rate, 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) – 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 – 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 – 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 starter 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 be expanded 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).

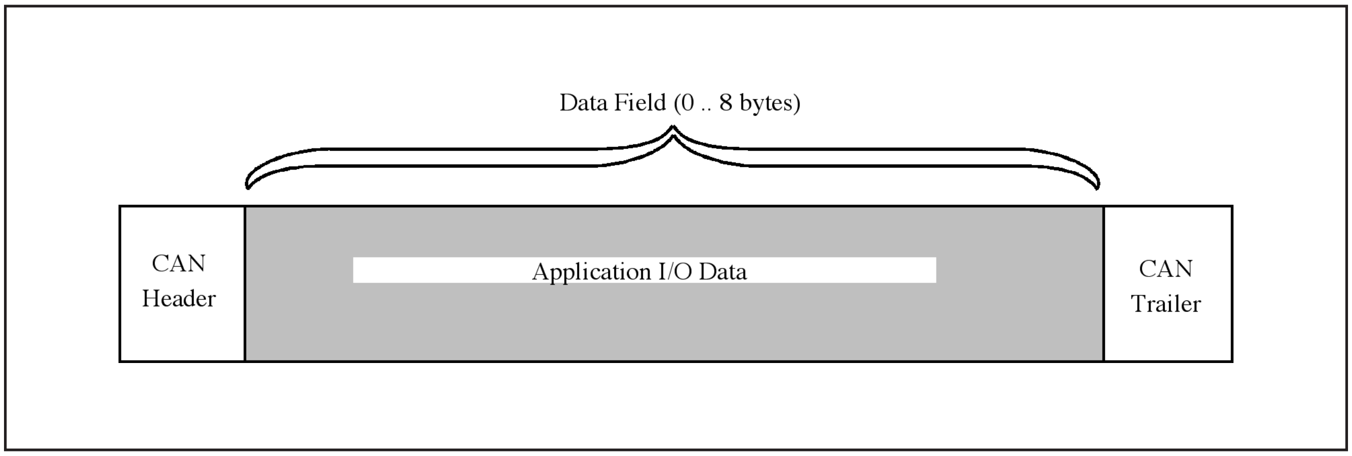

Figure 9. Format for I/O Message

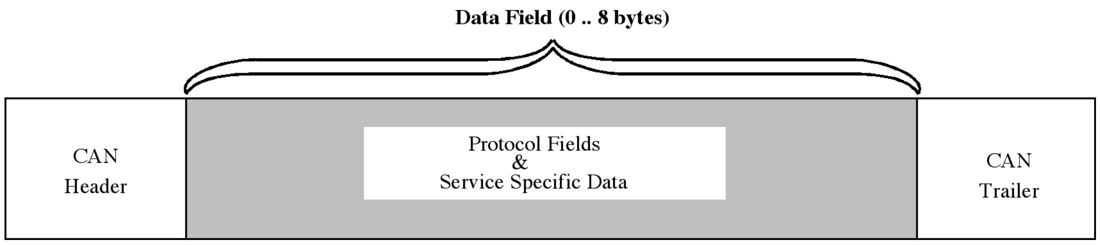
Data Field (0 .. 8 bytes)
CAN Header
Protocol Fields & Service Specific Data
CAN Trailer

Figure 10. Format for Explicit Message

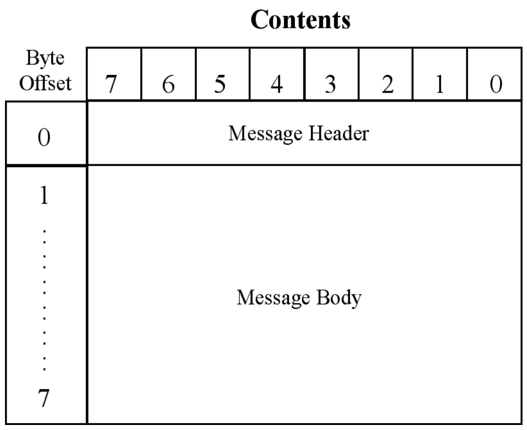

Figure 11. Non-Fragmented Explicit Message Data Field Format

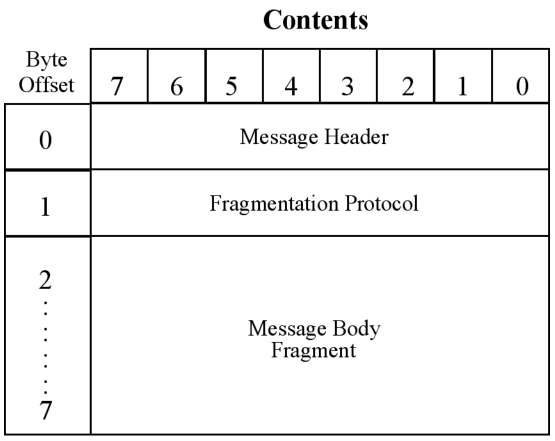

Figure 12. Fragmented Explicit Message Data Field Format

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 known 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 as 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 (Volume II,
Chapter 6-31).

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.