İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

INDUSTRIAL NETWORKS
(AN APPLICATION PROTOCOL ON CANBUS: CANUP)

M.S. Thesis by

Akın ÖZDEMİR, B.Sc.

Department :	**Control and Computer Engineering**
Programme:	**Control and Computer Engineering**

JANUARY 2004

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

INDUSTRIAL NETWORKS

(AN APPLICATION PROTOCOL ON CANBUS: CANUP)

M.S. Thesis by

Akın ÖZDEMİR, B.Sc.

(504981179)

Date of submission : 19 December 2003

Date of defence examination: 15 January 2004

JANUARY 2004

<u>**İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ**</u>

ENDÜSTRİYEL AĞLAR
(CANBUS ÜZERİNDE BİR UYGULAMA PROTOKOLÜ: CANUP)

YÜKSEK LİSANS TEZİ

Müh. Akın ÖZDEMİR

(504981179)

Tezin Enstitüye Verildiği Tarih : 19 Aralık 2003

Tezin Savunulduğu Tarih : 15 Ocak 2004

Tez Danışmanı :	**Prof.Dr. Eşref ADALI (İTÜ.)**
Diğer Jüri Üyeleri	**Prof.Dr. Tevfik AKGÜN (YÜ.)**
	Doç.Dr. Coşkun SÖNMEZ (İTÜ.)

OCAK 2004

FOREWORD

I am very grateful to my advisor Prof.Eşref ADALI for his guidance throughout this study.

Thanks to my family for their patience.

Special thanks to Melih Aybey for the opportunity he supplied me to put my theories into practise.

December 2003 Akın ÖZDEMİR

TABLE OF CONTENTS

ABBREVIATIONS

ASCII	:	American Standard Code for Information Interchange
ACK	:	Acknowledge
AUL	:	Active Units List
CAL	:	CAN Application Layer
CAN	:	Controller Area Network
CANUP	:	CAN Uygulama Protokolü (CAN Application Protocol)
CiA	:	CAN in Automation
CIP	:	Control and Information Protocol
CMS	:	CAN Message Specification
CP	:	Control Panel
CPU	:	Central Processing Unit
CRC	:	Cyclic Redundancy Check
CRL	:	Cabin Registration Lamps
DBT	:	Identifier Distributor
DCF	:	Device Configuration File
DLC	:	Data Length Code
DRL	:	Down Registration Lamps
EDS	:	Electronic Data Sheet
EMI	:	Electromagnetic Interference
EOF	:	End Of Frame
IDE	:	Identifier Extension
IPT	:	Information Processing Time
ISO	:	International Standardization Organization
LLC	:	Logical Link Control
LMT	:	Layer Management
LSB	:	Least Significant Bit
MAC	:	Medium Access Control
MSB	:	Most Significant Bit
NMT	:	Network Management
OD	:	Object Dictionary
ODVA	:	Open DeviceNet Vendors Association
PC	:	Personal Computer
PDO	:	Process Data Object
PM	:	Process Messages
RFI	:	Radio Frequency Interference
URL	:	Up Registration Lamps

LIST OF TABLES

Page Number

LIST OF FIGURES

Page Number

ENDÜSTRİYEL AĞLAR

(CANBUS ÜZERİNDE BİR UYGULAMA PROTOKOLÜ: CANUP)

ÖZET

Tezimiz, endüstriyel ağlara genel bir bakışla başlamaktadır. En yaygın olarak kullanılan 5 sistem Ethernet, Modbus, Profibus, Foundation Fieldbus ve CANbus kısaca anlatılarak avantaj ve dezavantajları incelenmiştir. Bu karşılaştırmada CANbus güçlü hata belirleme mekanizmaları, güvenilir yapısı, düşük maliyeti ve kolay uygulanabilirliği ile öne çıkmaktadır. Ayrıca tasarımcılar için hem yarı iletken hem de yazılım ve uygulama protokolü anlamında çok sayıda alternatif bulmak mümkündür.

Bundan sonra CANbus protokolü ayrıntılı olarak ele alınmıştır. Protokolün tarihçesi ve gelişim aşamaları kronolojik olarak anlatılmıştır. Protokol tanımlamalar ve işleyiş açısından ayrıntılarıyla incelenmiştir. Protokolün en önemli avantajlarından biriside birden fazla ünitenin aynı anda hatta erişmesine izin vermesidir. Herhangi bir çakışma durumunda ise yazılım ve donanım olarak bir problem oluşmadan yüksek öncelikli mesaja sahip olan ünite erişim hakkını kazanmakta diğer üniteler alıcı konumuna geçerek haberleşme devam etmektedir.

Sonraki bölümde CANbus üzerinde tanımlanmış en yaygın 2 uygulama protokolü CANopen ve DeviceNet kısaca anlatılmış daha sonra aşağıdaki 6 açıdan protokoller karşılaştırılarak benzerlik ve farklılıklar incelenmiştir.

- Mesaj Kimliği Atama Sistemi
- İşlem Verilerinin Aktarımı
- İki nokta arasında haberleşme kanalları
- İşlem veri mesaj bağlantılarının kurulması
- Ağ yönetimi
- Cihaz modelleme ve cihaz profilleri

Tezimizin en önemli bölümü ise CANbus üzerinde yeni bir uygulama protokolünün tanımlandığı bölümdür. CANUP (CAN Uygulama Protokolü) adını verdiğimiz bu protokol uygulamadan bağımsız olarak haberleşme hızı, mesaj kimliği atama, veri değişimi metotları, ağ yönetimi ve mesaj tetikleme mekanizmaları ile ilgili kuralları ve yapıları tanımlamaktadır.

Son bölümde ise tanımlanan CANUP protokolünün asansör kontrol sistemlerine uygulanması anlatılmaktadır. ASCAN olarak adlandırdığımız bu uygulamada asansör kontrol sistemlerinde kullanılan 3 ayrı seri haberleşme hattı tek bir CANbus hattında birleştirilmiştir. Ana kontrol paneli ile kat ve kabin üniteleri arasındaki haberleşme sistemi ayrıntılı olarak anlatılmıştır.

INDUSTRIAL NETWORKS

(AN APPLICATION PROTOCOL ON CANBUS: CANUP)

SUMMARY

Our thesis begins with an overview of industrial networks. Most commonly used 5 systems Ethernet, Modbus, Profibus, Foundation Fieldbus and CAN are briefly explained and advantages and disadvantages are analysed. In this comparison CANbus looks ahead due to its high reliability, robust error confinement mechanisms and easy and low cost implementation ability. Nevertheless there are lots of chips, software and application protocol alternatives for designers.

Then, CANbus protocol is explained in details. The history of the protocol and development stages are described in chronological order. The protocol is analyzed in detailed with definitions and functions. One of the most important advantages of the protocol is that it allows multiple units to access bus simultaneously. In case of collusion the unit with high priority message wins bus access right and other unit(s) becomes receiver. So communication is carried on without causing any hardware or software problem.

After that 2 well-known application protocols CANopen and DeviceNet defined on CANbus are explained briefly. Then the protocols are compared in following points and similarities and differences are discussed.

- Message Identifier Assignment Sytem
- Process Data Exchange
- Peer to peer communication channels
- Establisment of process data message connections
- Network Management
- Device modelling and device profiles

Definition of a new application protocol on CANbus is the most important part of the thesis. The protocol named as CANUP (CAN Uygulama Protokolü) defines general rules and structures for message identifier allocation, data exchange methods, network management, message triggering and baudrate.

At the last, an application of CANUP protocol for lift control systems is described. In this application named as ASCAN 3 different networks in lift control systems are gathered on single CAN network. Communication system between main control panel, cabin and floor units is described in details.

1. INDUSTRIAL NETWORKS

The Internet has become a ubiquitous force that is redefining how we live and work. Every imaginable kind of device will eventually be networked and this will transform all devices from information devices into communication devices.

But the device world is more diverse than the PC world. In most cases, it's more expensive to network a device with embedded intelligence than it is to connect a PC, and there are more ways of doing it. The plethora of network standards and the inherent difficulties in supporting more than one protocol make it difficult to decide while designing a new system.

The reason for adding network capability to a device is to save wiring when interconnecting multiple devices. Large factory automation and process control applications use industrial networks (fieldbuses) extensively. The elimination of large, unwieldy bundles of cables and the associated mess is an obvious advantage (Figure 1.1). Networking allows you to connect hundreds of devices to a single trunk line instead of using hundreds of individual wires. These benefits grow exponentially with the size of the system.

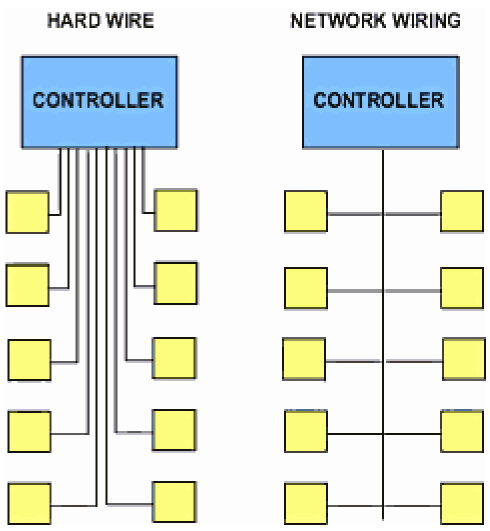


Figure 1.1 Classical Wiring versus Network Wiring

The modularity of a networked system is another major advantage. Connectivity is achieved through software, so taking a large system apart, putting it on a truck, and reassembling it somewhere else is much easier.

In addition to the wiring savings and modularity, there are three major reasons to add network capability to a device:

- **Diagnostics.** A networked device often can tell you if it is malfunctioning, or if something's about to go wrong. This information can be of great help. It's even more valuable if the information can be accessed remotely via the Internet.
- **Self-Configuration.** Machine controllers can automatically detect which modular components are connected to the network and determine what software configuration to load. This can save hours or even days of the delivery and setup time of a large system.
- **Enterprisewide Information Systems.** Initiatives to interconnect every system in a company may extend down to individual devices. Even the most mundane information can add money to the bottom line if used properly.

There are a lot of network standards out there. If you're going to add network capability to your device, which one should you support? It's best to view networks in terms of their scope. One of the reasons there are so many different network standards is that there are so many different requirements

1.1 Ethernet

One of the most widely accepted standards, Ethernet, is designed to transfer large amounts of data at high speed and to serve the needs of large installations.

The networking of millions of PCs in offices and the proliferation of the Internet around the world has made Ethernet a universal networking standard. Today, the standard is gradually working its way to the device level in data acquisition and control applications. The hardware and related software have evolved to the point where even inexperienced users can build simple networks and connect computers together.

Ethernet hardware can be purchased easily from office supply stores, computer stores, and e-commerce sites everywhere. The protocol appears to be a panacea for all those who are overwhelmed by the confusing array of standards vying for market dominance and who believe that the popular fieldbuses are expensive and difficult to use. Furthermore, a study by a big three automotive manufacturer showed that Ethernet could potentially serve up to 70% of plant floor networking applications.

On the other hand, Ethernet does have its drawbacks. For example, it has a high overhead-to-message ratio for small amounts of data. Also, Ethernet carries no power on the bus, and its RJ-45 connectors are physically vulnerable and more susceptible to EMI/RFI than most fieldbuses. And even now, its multiple open and proprietary standards are a source of confusion in the industry.

1.2 Modbus

Modbus Remote Terminal Unit (RTU) / ASCII is probably the most popular serial protocol in instrumentation, automation, and process control. Today, it provides everything from short serial linkage of smart devices to wide area networking of many devices. Modbus is commonly used with gateways and works well encapsulated in TCP/IP. Developed almost 25 years ago, it's a simple yet effective way of encapsulating analog and digital I/O and parameters in registers.

Modbus can link as many as 250 devices on an RS-485 cable. Furthermore, you can find many gateway devices that link Modbus and other networks, so if your product has the Modbus protocol on a serial port, you can get from there to almost any network using a black box converter.

The downside, though, is that transmission speed is slow on standard serial media. Also, the protocol lacks sophistication (i.e., it offers no peer-to-peer capabilities, and it is not object oriented).

1.3 Profibus

Profibus is commonly found in process control, large assembly, and material-handling machines—single-cable wiring of multi-input sensor blocks, pneumatic valves, complex intelligent devices, smaller subnetworks, and operator interfaces. Nearly universal in Europe and popular in North America, South America, and parts of Africa and Asia, Profibus is the most widely accepted international networking standard. It can handle large amounts of data at high speed and serve the needs of large installations. The DP, FMS, and PA versions collectively address the majority of automation applications.

Unfortunately, the network comes with a high overhead-to-message ratio for small amounts of data and carries no power on the bus. Profibus also costs slightly more than other buses. The standard’s European and Siemens centricity is occasionally an obstacle for the users at far locations.

1.4 Foundation Fieldbus

Foundation Fieldbus rapidly establishing itself as the future standard for process industry networking. Since its official introduction in 1997, many distributed control system vendors have embraced the protocol, developing and certifying devices that conformed to its specifications. This standard contends with Modbus, HART, and Profibus PA.

The fieldbus is a flexible, sophisticated protocol with many capabilities. It holds great appeal because it’s intrinsically safe and provides an integrated device-level/plant-level approach. Broader adoption of Foundation Fieldbus has been slowed by the protocol’s process-industry-centric nature, the limited availability of compatible devices, and the slow process of standardization. The fieldbus combines a device-level network and High-Speed Ethernet.

The standard is typically used in distributed control, continuous process control, batching, and oil and gas processing operations.

1.5 Controller Area Network

In the early 1980s, Bosch developed the Controller Area Network (CAN) so that the primary control components in an automobile (e.g., brake lights, airbags, sensors, lights, electric windows, and door locks) could be connected by a single cable instead of bundle of cables. Automotive manufacturers found that if a wiring harness was faulty, it was sometimes cheaper to scrap the entire car than to troubleshoot the wiring harness. With a network, you can wire a control panel virtually in software, rather than physically with a screwdriver and terminal blocks. The added hardware cost of the network is less than paid for by labor savings. The same applies to automated equipment in a factory.

Of course, in a vehicle, communication can mean the difference between life and death. You cannot tolerate network errors, regardless of origin. CAN lives up to these rigorous requirements, with a statistical probability of less than one faulty message per century.

The standard is minimally a three-wire bus, with ground and two opposing signal conductors. Signals consist of a pulse train centered at about 2.5 V, with the high signal raising to about 3.2 V and the low signal falling to about 1.8 V. This creates noise immunity, which is especially important in a vehicle.

CAN is a low-level message arbitration protocol implemented on inexpensive chips available from multiple vendors and manufactured by the millions. To have a functional network protocol, an additional software layer must be added.

Higher layer protocols (e.g., DeviceNet and CANopen) can be thought of as a sophisticated set of macros for CAN messages, specifically suited for automation.

2. CONTROLLER AREA NETWORK (CAN)

2.1 History of CAN

The number of electronic devices installed in modern motor vehicles is increasing at almost an exponential rate. Electronic controlled devices such as ABS, Electronic Transmission Control, Airbags, Keyless Entry, Active Suspension, Radios, Navigation, and many others too numerous to list have been added to the motor vehicle in just the last few years. These have been added because of customer demand, federal legalization, comfort, vehicle performance improvements and many other reasons.

The need for all of the electronic systems in the vehicle to exchange information is ever growing. Typically each sensor/device has a wire run from point A to point B, which results in many redundant sensors etc. In the vehicle, the need to exchange information has caused the number of cables and wire weight to grow substantially over the last ten years. The modern mid-line vehicle has many thousands of feet of wire. The number of wires (cut leads) has grown such that a driver passenger door can have more then fifty wires in the bundle going through the hinge to the driver's side door. These large bundles of wire are causing problems for manufacturing as well as system reliability. To reduce the number of wires and bundle size vehicle manufactures have gone to a method of serially transmitting data between modules.

Forced by the increasing number of distributed control systems in cars and the increasing wiring costs of car body electronics, the availability of a powerful and reliable serial data communication system for the exchange of messages between the different control units was becoming urgent. This was the starting point for BOSCH and Controller Area Network was officially introduced in February 1986. The first protocol controller chip (82526) was provided by INTEL in 1989. Originally developed for use in automotive applications, CAN has begun to be successful in many other fields of application. The first CAN applications were embedded machine control systems and lift controllers.

CAN first time applied in Mercedes S class cars, launched in 1992, providing a high speed network for communication between engine controller, gearbox controller and dashboard and a low speed network for distributed air conditioning control. BMW, Porsche, Jaguar, Volkswagen, Fiat and Renault applied CAN short time later.

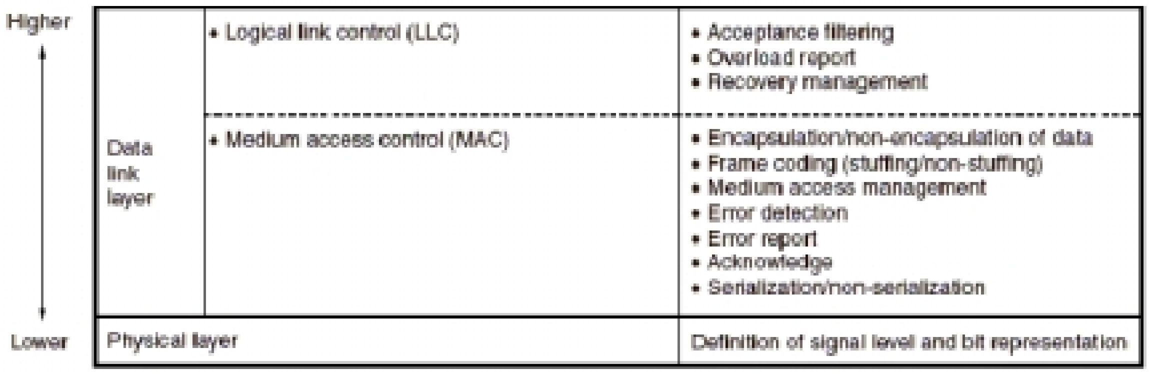
In 1993 CAN was adopted as worldwide standard ISO 11898 by the International Standardization Organization ISO, defining ISO-OSI Layers 1 and 2.

Today, more than 110 protocol controller implementations in form of stand-alone controllers or integrated into a microcontroller are available from 20 manufacturers including all of the main semiconductor manufacturers. According to a survey of the CAN-in-Automation organization, the already designed number of CAN chips were about 210 million in the year 2002 and estimated as 300 million at the end of 2003. Today, the CAN interface already may be regarded as the standard serial interface of microcontrollers, used in any type of distributed embedded applications.

Due to the outstanding features of the CAN protocol, the availability of low cost network controllers from many manufacturers as well as the ease of implementation, the CAN protocol today is in use not only in almost any type of mobile system (passenger cars, trucks and buses, agricultural equipment, ships, aircraft, elevators), but also in any type of machines, from textile, packaging, paper manufacturing machines up to any type of medical equipment or robot control systems. Controller Area Network is particularly well suited for networking of “intelligent” devices. Since the availability of higher layer communication standards and profiles, CAN-based networking is becoming one of the most promising solutions for open, distributed automation systems, competing very successfully with other bus standards in the field of industrial plant automation.

2.2 Protocol

The specifications of CAN are broadly classified into two layers: a physical layer and a data link layer. The data link layer consists of logical link control and medium access control. The configuration of each layer is shown in Figure 2.1.



Higher ↕ Lower			
	Data link layer	• Logical link control (LLC)	• Acceptance filtering • Overload report • Recovery management
		• Medium access control (MAC)	• Encapsulation/non-encapsulation of data • Frame coding (stuffing/non-stuffing) • Medium access management • Error detection • Error report • Acknowledge • Serialization/non-serialization
	Physical layer		Definition of signal level and bit representation

Figure 2.1 Layer Configuration

The scope of the LLC sublayer is

- to provide services for data transfer and for remote data request,
- to decide which messages received by the LLC sublayer are actually to be accepted,
- to provide means for recovery management and overload notifications.

There is much freedom in defining object handling. The scope of the MAC sublayer mainly is the transfer protocol, i.e. controlling the Framing, performing Arbitration, Error Checking, Error Signaling and Fault Confinement. Within the MAC sublayer it is decided whether the bus is free for starting a new transmission or whether a reception is just starting. Also some general features of the bit timing are regarded as part of the MAC sublayer. It is in the nature of the MAC sublayer that there is no freedom for modifications.

The scope of the physical layer is the actual transfer of the bits between the different nodes with respect to all electrical properties. Within one network the physical layer, of course, has to be the same for all nodes. There may be, however, much freedom in selecting a physical layer.

2.2.1 Frame Formats

2.2.1.1 Standard format frame

In this format, 2,048 types of identifiers can be set. Because the identifier of a standard format frame is 11 bits long, 2,048 types of messages can be handled.

2.2.1.2 Extended format frame

In this format, about 5.3 million types of identifiers can be set. The identifier of an extended format frame is extended to 29 bits (11 bits + 18 bits), the number of messages that can be handled increases to 2,048 x 2^{18}.

If the SRR and IDE bits in the arbitration field are “recessive: logical level 1”, the frame is sent in the extended format.

2.2.2 Frame types

The frames of the CAN protocol can be classified into the following four types.

Table 2.1 Frame Types and Their Usage

Frame Type	Description
Data frame	Frame to transmit data
Remote frame	Frame to request data frame
Error frame	Frame to detect and report an error
Overload frame	Frame to delay the next data frame or remote frame

2.2.2.1 Bus value

The bus has two values: “dominant” and “recessive”. The “dominant level” is expressed as logical 0, and the “recessive” level is expressed as logical 1. If both the dominant level and recessive level are simultaneously transmitted, the value of the bus is the dominant level.

2.2.2.2 Data frame

A data frame consists of seven fields, as illustrated below.

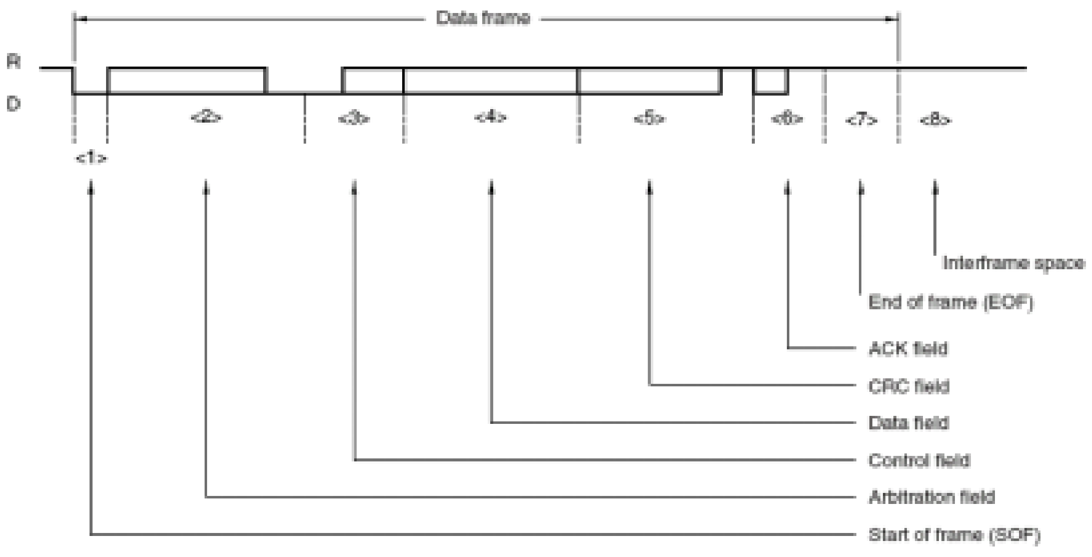


Figure 2.2 Data Frame

2.2.2.3 Remote frame

A remote frame consists of six fields, as illustrated below.

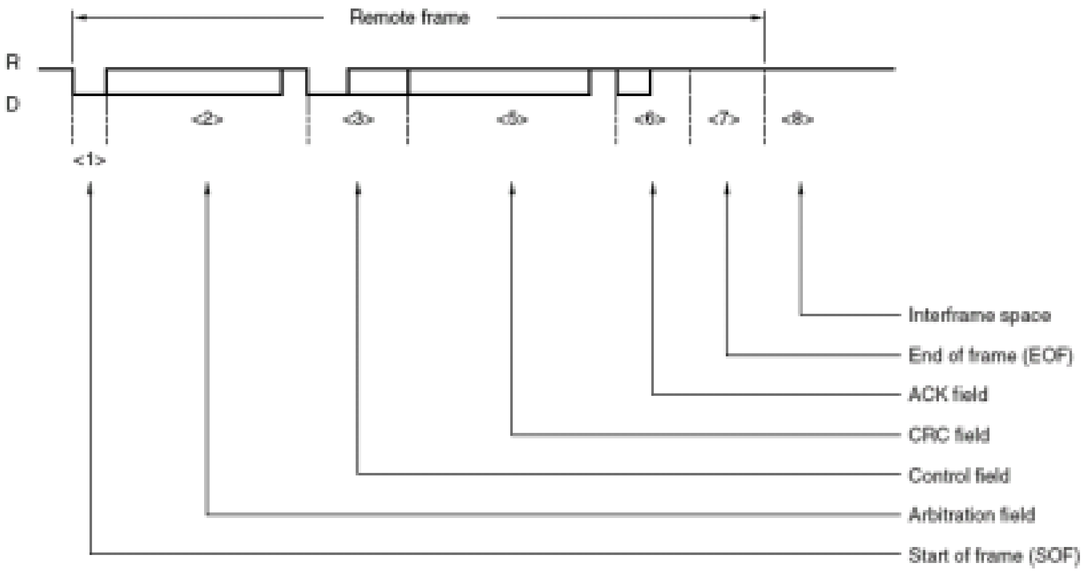


Figure 2.3 Remote Frame

<Description of each field>

<1> Start of frame: Indicates the start of a data frame or a remote frame.

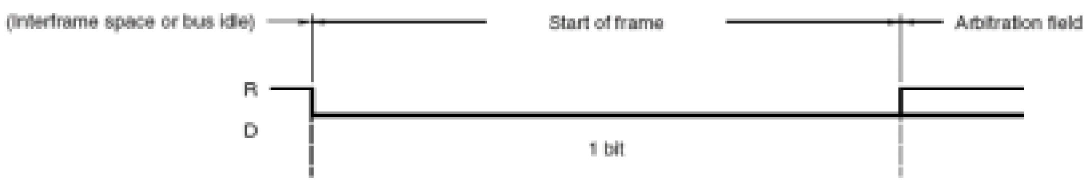


Figure 2.4 Start of Frame

- If a dominant level is detected while the bus is idle, it is recognized as the start of frame.
- If a recessive level is detected at the sample point of the start of frame, it is assumed to be noise and the bus state becomes idle again.

<2> Arbitration field: Sets priority, data frame/remote frame, and frame format.

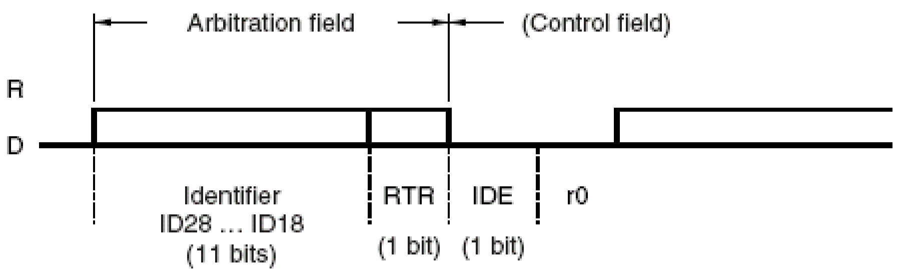


Figure 2.5 Arbitration Field (Standard Format Frame)

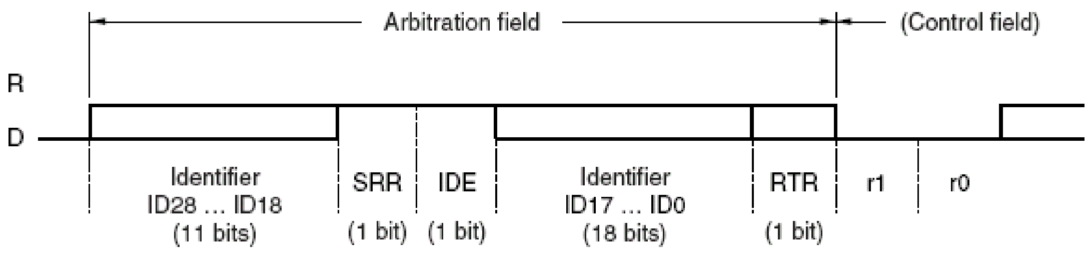


Figure 2.6 Arbitration Field (Extended Format Frame)

Table 2.2 Setting of RTR Bit

Type of Frame	Bit
Data frame	0 (D)
Remote frame	1 (R)

Table 2.3 Setting of Frame Format (IDE Bit) and Number of Bits of Identifier (ID)

Protocol Mode	SRR Bit	IDE Bit	Number of Bits
Standard format mode	None	0 (D)	11 bits
Extended format mode	1 (R)	1 (R)	29 bits

<3> Control field: Sets N, the number of data bytes of the data field.

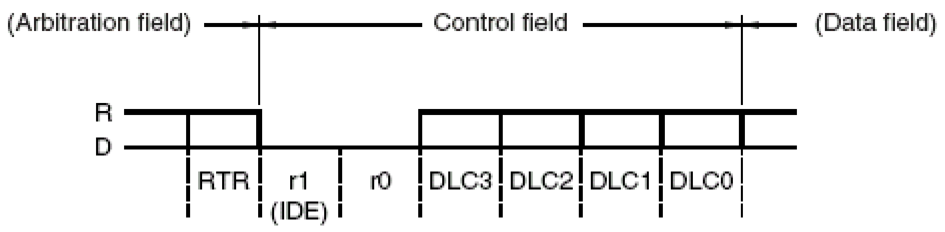


Figure 2.7 Control Field

The IDE bit of the control field and r1 are identical in the standard format frame.

Table 2.4 Data Length Setting

Data Length Code				Number of Data Bytes
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
⋮	⋮	⋮	⋮	
0	1	1	1	7
1	×	×	×	8

<4> Data field: A group of data (in bytes) set in the control field. Up to eight data can be set.

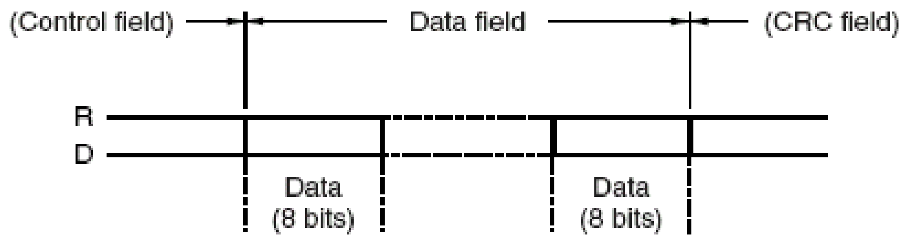


Figure 2.8 Data Field

<5> CRC field: 16-bit field that detects an error in transmit data

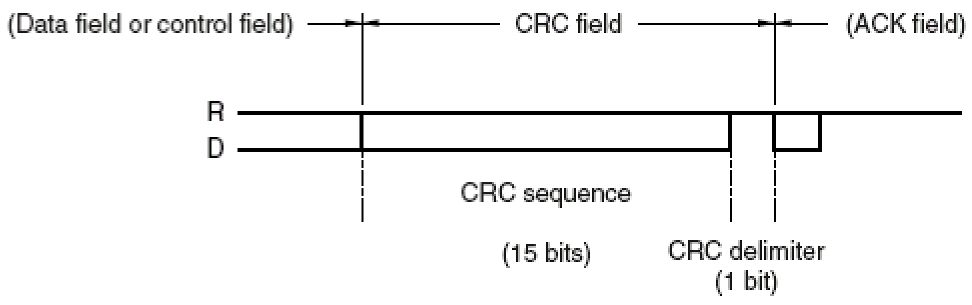


Figure 2.9 CRC Field

• The polynomial expression P(X) that generates a 15-bit CRC is

$$P(X) = X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1$$

• Transmission node: Transmits the CRC sequence calculated from the data of the start of frame, arbitration field, control field, and data field (data before bit stuffing).

• Reception node: Compares the CRC sequence calculated from the data bits of the receive data, excluding the stuff bits, with the CRC sequence in the CRC field. If the two sequences do not match, the node transmits an error frame.

<6> ACK field: Field for checking normal reception

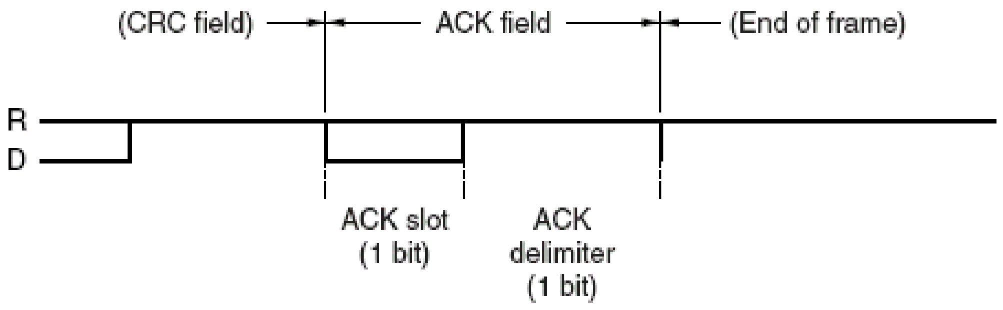


Figure 2.10 ACK Field

• If a CRC error is not detected, the reception node sets the ACK slot to the dominant level.

• The transmission node outputs 2 bits “recessive” level.

<7> End of frame: Frame that indicates the end of the data frame/remote frame.

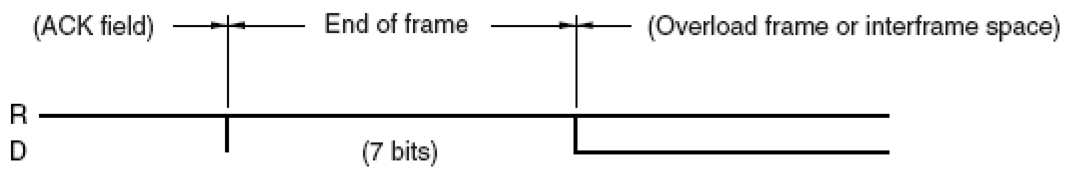


Figure 2.11 End of Frame

<8> Interframe space: Frame inserted between the data frame, remote frame, error frame, or overload frame and the next frame, as a delimiter between frames.

• The bus status differs depending on the error status.

(A) Node in error active state: Consists of 3 intermission bits and bus idle

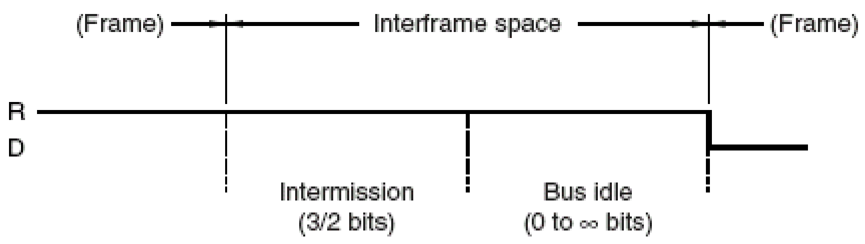


Figure 2.12 Interframe Space/Error Active

(B) Node in error passive state: Consists of intermission, suspend transmission, and bus idle.

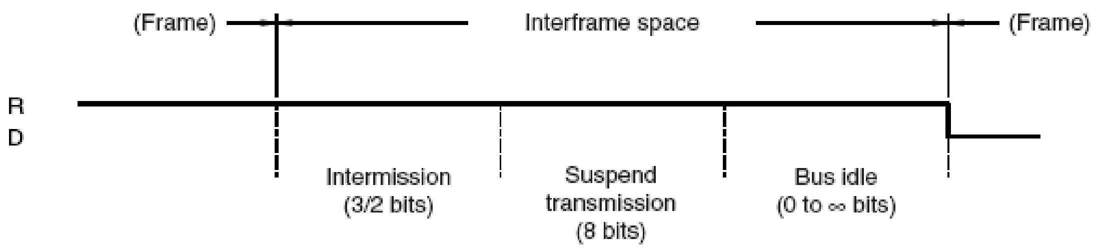


Figure 2.13 Interframe Space/Error Passive

• Operation in error state

Error State	Operation
Error active	A node in this state can start transmission in the bus idle state.
Error passive	A node in this state can start transmission after a bus idle (suspend transmission) state of 8 bits. If another node starts transmission, the node is in the reception state (the transmission priority of the source node drops).

• Operation if the third bit of intermission is "dominant"

Error State	Operation
Transmission not pending	The bit is judged as the start of frame output by another node, and reception is performed.
Transmission pending	The bit is judged as the start of frame output by the source node, and the node transmits the identifier.

2.2.2.4 Error frame

A node that has detected an error outputs this frame.

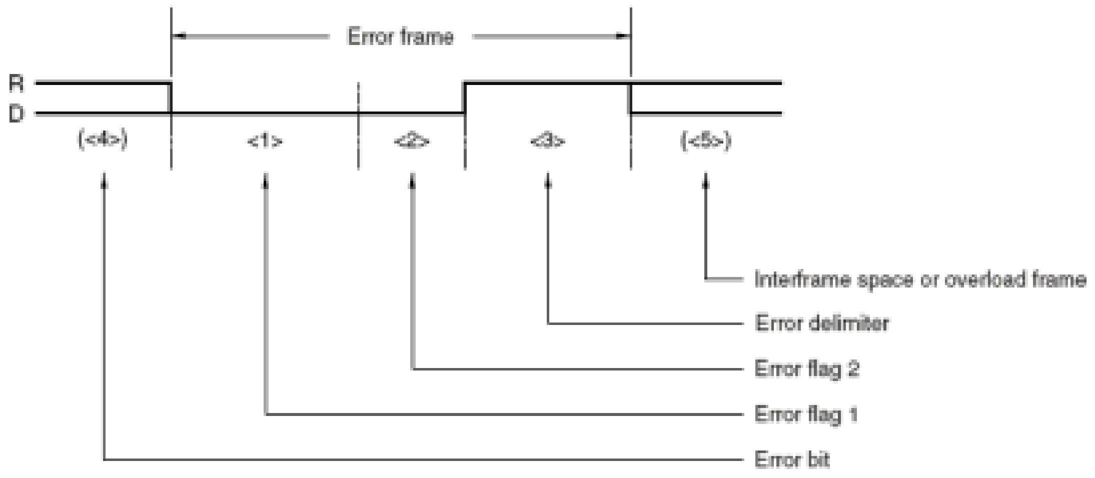


Figure 2.14 Error Frame

Table 2.5 Definition of Each Field for Error Frame

No.	Name	Number of Bits	Definition
<1>	Error flag 1	6	Error active node: Continuously outputs 6 bits of "dominant" level. Error passive node: Continuously outputs 6 bits of "recessive" level. If another node outputs the "dominant (D)" level while a passive error flag is being output, the passive error flag does not end until the same level is detected for 6 consecutive bits.
<2>	Error flag 2	0 to 6	This is the error flag output when the node that received error flag 1 re-outputs an error flag after detecting a bit stuffing error.
<3>	Error delimiter	8	Continuously outputs 8 bits of "recessive" level. If a "dominant" level is detected at the eighth bit, an overload frame is transmitted starting from the next bit.
<4>	Error bit	–	Bit at which an error has been detected. The error flag is output from the bit next to the error bit. In the case of a CRC error, the error flag is output following the ACK delimiter.
<5>	Interframe space/ overload frame	–	Interframe space or overload frame continues.

2.2.2.5 Overload frame

This frame is transmitted under the following conditions.

• If the reception node has not completed a receive operation.

• If the dominant level is detected in the first 2 bits during an intermission.

• If the dominant level is detected in the last bit (eighth bit) of the end of frame or the last bit (eighth bit) of the error delimiter/overload delimiter.

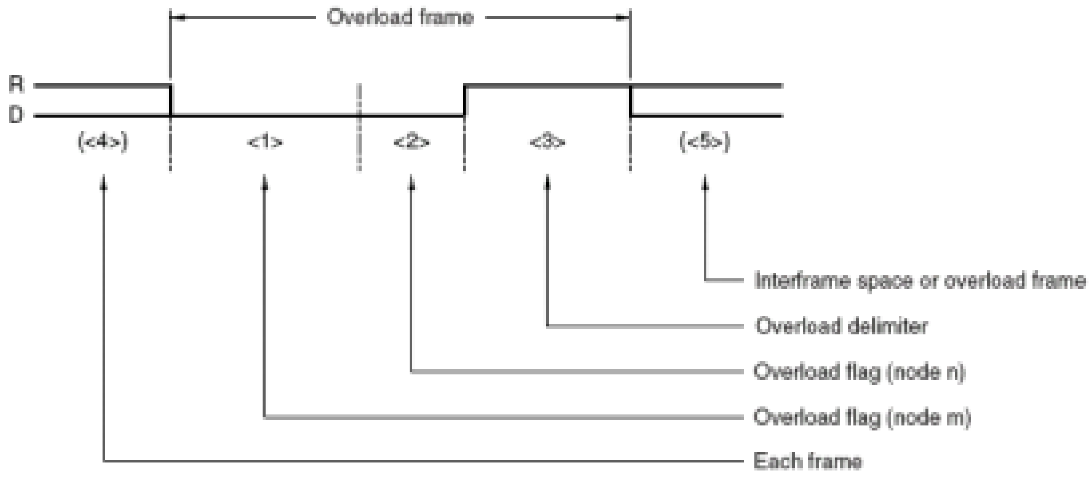


Figure 2.15 Overload Frame

Table 2.6 Definition of Each Field for Overload Frame

No.	Name	Number of Bits	Definition
<1>	Overload flag from node m	6	Continuously outputs 6 bits of "dominant" level. These bits are output because node m is not ready for reception.
<2>	Overload flag from node n	0 to 6	Node n that received an overload flag in the interframe space outputs an overload flag.
<3>	Overload delimiter	8	Continuously outputs 8 bits of "recessive" level. If "dominant" level is monitored at the eighth bit, an overload frame is transmitted starting from the next bit.
<4>	Each frame	–	Output following the end of frame, error delimiter, and overload delimiter.
<5>	Interframe space/overload frame	–	Interframe space or overload frame continues.

2.3 Function

2.3.1 Determining bus priority

(1) If one node starts transmission

• In the bus idle state, the node that outputs data first acquires the transfer rights, and outputs (transmits) data.

(2) If two or more nodes simultaneously start transmission

• The node that continuously outputs the "dominant (D)" level for the longest time, starting from the first bit of the arbitration field, acquires the bus priority (if the dominant level and recessive level are simultaneously transmitted, the value of the bus is the dominant level).

• The transmission node compares the arbitration field it has output with the data level on the bus.

Table 2.7 Determining Bus Priority

Bus Level in Arbitration Field	Operation of Transmission Node
If levels match	Continues transmission.
If levels do not match	Stops data output from the bit next to the one in which the discrepancy was found and a receive operation is performed.

(3) Priority of data frame and remote frame

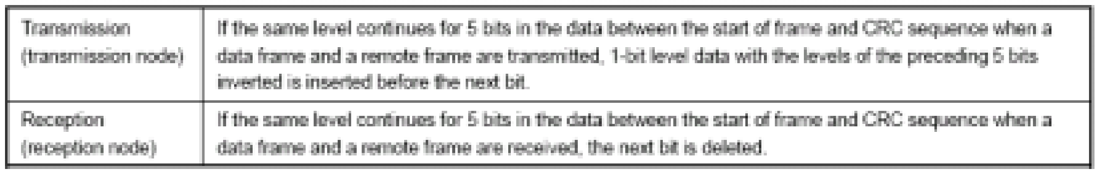
• If a data frame and remote frame conflict on the bus, the data frame, whose RTR bit is the “dominant level”, takes precedence.

If a data frame in the extended format conflicts with a remote frame in the standard format (if ID28 to ID18 of both are the same), the remote frame in the standard format takes precedence.

2.3.2 Bit stuffing

Bit stuffing is a mechanism that is used to append 1 bit of inverted data to establish synchronization if the same level continues for more than 5 bits, in order to prevent a burst error.

Table 2.8 Bit Stuffing



Transmission (transmission node)	If the same level continues for 5 bits in the data between the start of frame and CRC sequence when a data frame and a remote frame are transmitted, 1-bit level data with the levels of the preceding 5 bits inverted is inserted before the next bit.
Reception (reception node)	If the same level continues for 5 bits in the data between the start of frame and CRC sequence when a data frame and a remote frame are received, the next bit is deleted.

2.3.3 Multi masters

Because the bus priority (the node that acquires the transfer rights) is determined by the identifier, any node can be the bus master.

2.3.4 Multi cast

Although only one transmission node exists, the same data can be received by two or more nodes simultaneously because the same identifier can be set to these nodes.

2.3.5 Sleep mode/stop mode

The CAN sleep mode and CAN stop mode functions can be used to reduce the power consumption by placing the CAN controller in the standby state.

The CAN sleep mode is set using the procedure defined in the CAN Specifications. The CAN sleep mode is woken up by the CAN controller via bus operation, but the CAN stop mode is not woken up by bus operation (it is controlled by CPU access).

2.3.6 Error control

(1) Error types

Table 2.9 Error Types

Error Type	Description of Error		Node Detecting Error	Field/Frame
	Detection Method	Detection Condition		
Bit error	Comparison of output level with level on bus (except stuff bit)	The levels do not match	Transmission/reception node	Bit that outputs data on the bus at the start of frame to end of frame, error frame and overload frame
Stuff error	Check of the data received at the stuff bit	6 consecutive bits of the same level data	Transmission/reception node	Start of frame to CRC sequence
CRC error	Comparison of CRC generated from the receive data with the received CRC sequence	CRC does not match	Reception node	Start of frame to data field
Form error	Field/frame check of the fixed format	Detection of a fixed format error	Reception node	• CRC delimiter • ACK field • End of frame
ACK error	Check of the ACK slot by the transmission node	Detection of a recessive level in the ACK slot	Transmission node	ACK slot

(2) Error frame output timing

Table 2.10 Error Frame Output Timing

Error Type	Output Timing
Bit error, stuff error, form error, ACK error	Error frame is output at the next bit timing following the detected error.
CRC error	Error frame is output at the next bit timing following the ACK delimiter.

(3) Measures when error occurs

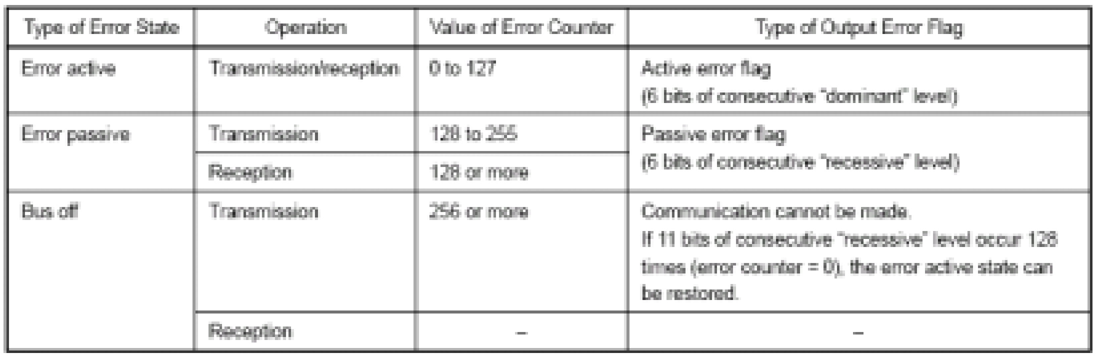
• The transmission node re-transmits the data frame or remote frame after the error frame.

(4) Types of error states

• Error states are classified into three types: “error active”, “error passive”, and “bus off”. The latter error states are more serious than the former (the error state is determined by the values of the transmission error counter and reception error counter).

• If the value of the transmission error counter or reception error counter reaches 96 or more while the node is in the error active state (while the value of the error counter is 127 or less), the chances are the bus has a severe fault and must be checked.

Table 2.11 Types of Error States

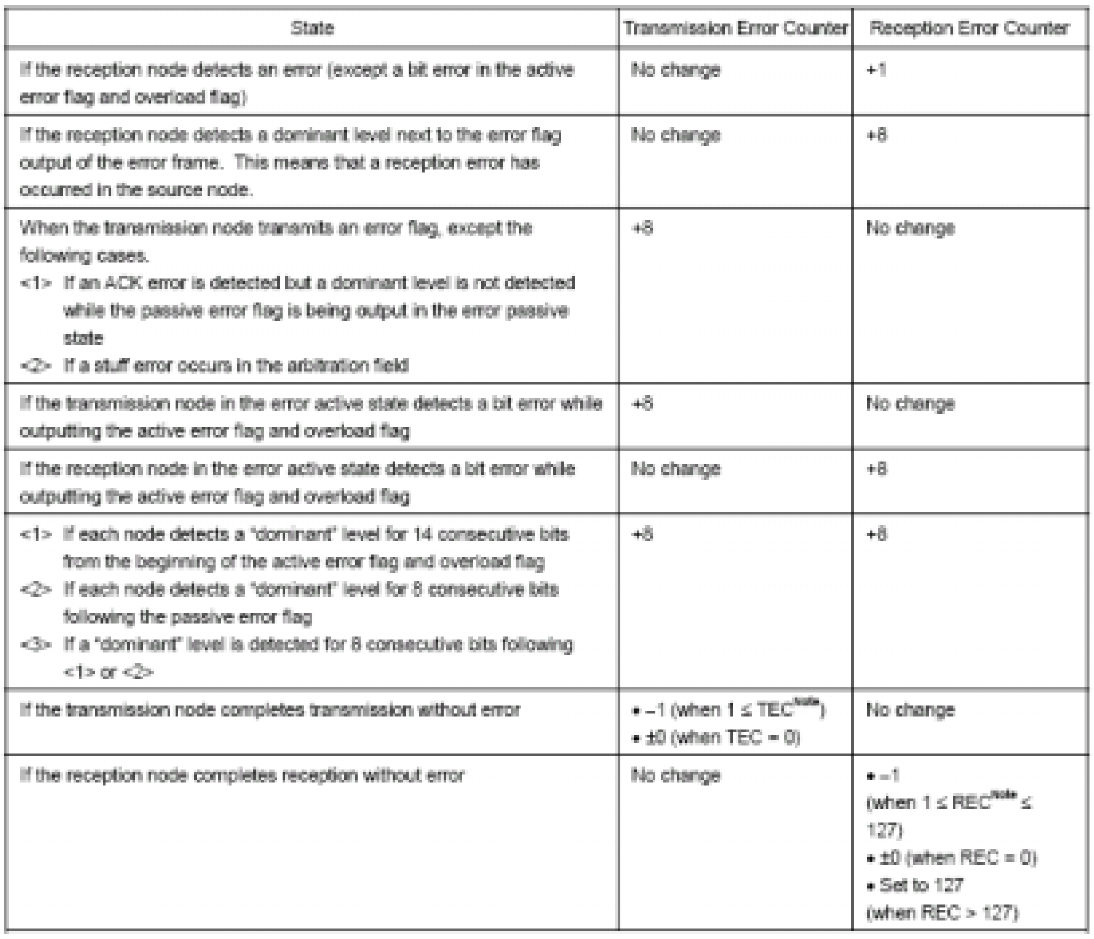


Type of Error State	Operation	Value of Error Counter	Type of Output Error Flag
Error active	Transmission/reception	0 to 127	Active error flag (6 bits of consecutive "dominant" level)
Error passive	Transmission	128 to 255	Passive error flag (6 bits of consecutive "recessive" level)
	Reception	128 or more	
Bus off	Transmission	256 or more	Communication cannot be made. If 11 bits of consecutive "recessive" level occur 128 times (error counter = 0), the error active state can be restored.
	Reception	–	–

(5) Error counter

The error counter counts up when an error occurs, and counts down when transmission or reception has been correctly performed.

Table 2.12 Error Counter



State	Transmission Error Counter	Reception Error Counter
If the reception node detects an error (except a bit error in the active error flag and overload flag)	No change	+1
If the reception node detects a dominant level next to the error flag output of the error frame. This means that a reception error has occurred in the source node.	No change	+8
When the transmission node transmits an error flag, except the following cases. <1> If an ACK error is detected but a dominant level is not detected while the passive error flag is being output in the error passive state <2> If a stuff error occurs in the arbitration field	+8	No change
If the transmission node in the error active state detects a bit error while outputting the active error flag and overload flag	+8	No change
If the reception node in the error active state detects a bit error while outputting the active error flag and overload flag	No change	+8
<1> If each node detects a "dominant" level for 14 consecutive bits from the beginning of the active error flag and overload flag <2> If each node detects a "dominant" level for 8 consecutive bits following the passive error flag <3> If a "dominant" level is detected for 8 consecutive bits following <1> or <2>	+8	+8
If the transmission node completes transmission without error	• –1 (when 1 ≤ TEC^{note}) • ±0 (when TEC = 0)	No change
If the reception node completes reception without error	No change	• –1 (when 1 ≤ REC^{note} ≤ 127) • ±0 (when REC = 0) • Set to 127 (when REC > 127)

Note: TEC: Transmit Error Counter, REC: Receive Error Counter

2.3.7 Baud rate control

2.3.7.1 Bit time

Bit time has a 1-bit length on the CAN protocol. Bit time consists of eight to 25 time quanta (time units), and is divided into four segment areas.

The structure is illustrated in Figure 2-16.

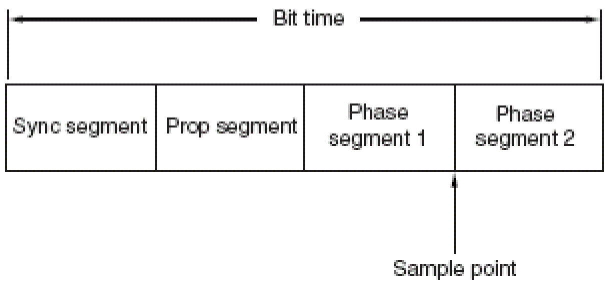


Figure 2.16 Bit Time

Table 2.13 Bit Time Segments

Sync segment	Segment that starts immediately after bit synchronization
Prop segment	Segment that corrects signal delay by the CAN transceiver and CAN bus (Concept) The ACK slot in the ACK field is returned from the reception node, until phase segment 1 of the transmission node, while the data and control frames are being transmitted. (Prop segment time ≥ CAN transceiver (transmission, reception delay) + CAN bus delay)
Phase segments 1, 2	Segments that correct error of bit time. (The larger these segments, the greater the permissible error can be. However, the communication speed is limited.)

Table 2.14 Segment Name and Segment Length

Segment Name	Number of Time Quanta That Can Be Set for Each Segment
Sync segment (Synchronization Segment)	1
Prop segment (Propagation Segment)	1 to 8
Phase segment 1 (Phase Buffer Segment 1)	1 to 8
Phase segment 2 (Phase Buffer Segment 2)	Phase segment 1 or IPT, whichever greater in value (IPT = 0 to 2)

IPT (Information Processing Time): Time necessary for determining the next bit level

2.3.7.2 Synchronization of data bit

Bit synchronization of the CAN protocol can be classified into two types: hard synchronization and bit resynchronization.

(a) Hard synchronization

Hard synchronization is performed if both the reception node and transmission node are in the bus idle state and detect the start of frame.

If the node detects a falling edge on the bus in the bus idle state, the sync segment of the start of frame starts.

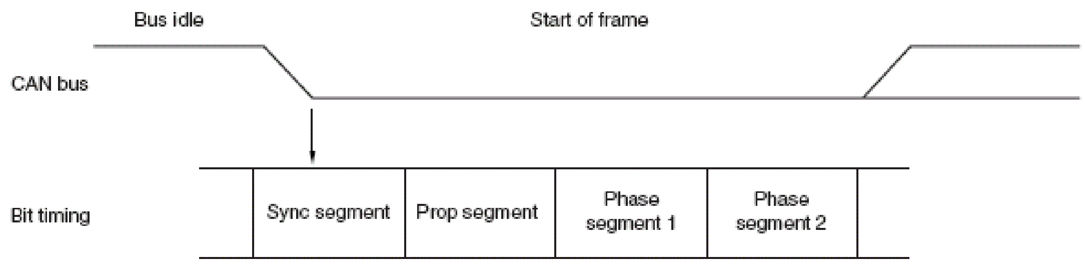


Figure 2.17 Synchronization of Data Bit

(b) Bit re-synchronization

If the node detects a falling edge on the bus in a segment other than the sync segment during reception or transmission, it performs bit re-synchronization.

Bit re-synchronization corrects the falling edge on the bus and the position of the sync segment.

• The phase error of the edge is given by the relative position of the detected edge and sync segment.

<Code of phase error>

0: The edge is in the sync segment.

Positive: The edge is before the sample point (positive phase error).

Negative: The edge is after the sample point (negative phase error).

• The width of correction for bit re-synchronization is specified by SJW. One to four time quanta are set for SJW.

Bit re-synchronization is performed according to the following criteria.

(i) Phase error ≤ SJW set value

• If phase error is positive: Phase error with SJW set value as the maximum

Phase segment 1 is extended by the time quanta.

• If phase error is negative: Phase error with SJW set value as the maximum

Phase segment 2 is shortened by the time quanta.

(ii) Phase error > SJW set value

• If phase error is positive: Phase segment 1 is extended by the SJW set value.

• If phase error is negative: Phase segment 2 is shortened by the SJW set value.

3. CAN-BASED APPLICATION LAYER PROTOCOLS

The CAN protocol has gained worldwide acceptance as a very versatile, efficient, reliable and economic platform for almost any type of data communication in mobile systems, machines, technical equipment and industrial automation. Based on sophisticated standardized higher layer protocols and profiles, CAN-based open automation technology successfully competes on the market of distributed automation systems. One of the main reasons for the enormous success of CAN-based systems obviously are the special features of the CAN-protocol, especially its producer-consumer-oriented principle of data transmission and its multimaster capability. With these properties, the CAN-protocol also from the technical point of view is very attractive for the usage in distributed systems applications.

When referring to the "CAN standard" or "CAN protocol" we understand the functionality, which is standardized in ISO 11898. This standard comprises the Physical (Layer 1) and Data Link Layer (Layer 2) according to the OSI-reference model. Whereas Layer 1 is responsible for functions like physical signaling, encoding, bit timing and bit synchronization, Layer 2 performs functions like bus arbitration, message framing and data security, message validation, error detection and signaling and fault confinement. The CAN standard does not specify the medium attachment unit and the medium upon which it resides, nor an Application Layer.

The Layer 2 of the CAN protocol offers two types of connectionless transmission services to the user:

- Unacknowledged transfer of a CAN-message and
- Unacknowledged remote request of a CAN-message

Connectionless transmission means that no data link connection has to be established before performing a message transfer or request. Reception of messages is supported by the CAN chips in form of different type object filtering and object buffering methods. A Layer 2 CAN data message according to the CAN

Specification V 2.0 is determined by the message identifier, standard/extended format indication, data length and the data to be transmitted.

Since the CAN-Protocol specifies no rules for the assignment of message-identifiers, a variety of different, application-specific usages are possible. Assignment of the CAN message identifiers therefore is one of the most important decisions when designing a CAN-based communication system. Assignment and allocation of message identifiers also is one of the main items of a higher Layer approach.

3.1 The Requirement of Higher Layers

In practice the implementation even of very simple distributed CAN-based systems shows that beside of the basic Layer 2 services further functionality is required or desirable e.g. for the transmission of data blocks longer than 8 bytes, acknowledged or confirmed data transfer, identifier distribution, network startup or the supervision of nodes. Since this additional functionality directly supports the application process, it is understood as "Application Layer". If implemented properly, the introduction of an Application Layer in addition with an appropriate Application Layer Interface provides a clearly defined separation of communication and application processes.

Since the CAN protocol provides very unique features, most of the known higher layer protocols conserve this features for the user of the Application Layer by providing direct access to the services of the Data Link Layer (no additional protocol overhead for basic functions).

Especially for industrial automation applications, the need for open, standardized higher layer protocols was raised which support interoperability and exchangeability of devices of different manufacturers. Supplementary to a standardized Application Layer therefore the specification of standard device models, "standard devices" and "standard applications" of basic functionality is required.

3.2 Survey of CAN-based Higher Layer Protocols

According to the widespread usage of CAN networks with different objectives and requirements beside of many special solutions several main standards of CAN-based Higher Layer protocols are available today. According to the different requirements these solutions differ significantly with respect to their scope and performance.

Main representatives of open distributed system standards for industrial applications are CANopen and DeviceNet.

The industrial application of open distributed systems standards comprises low-level networking of industrial devices (sensors, actors, controllers, men-machine interfaces) in industrial automation. The main requirements of this type of application are configurability, flexibility and extendibility. To provide manufacturer independence the definition of device functionality has to be specified in form of "Devices Profiles". Accordingly, communication systems solution of that type provides a complete framework of communication and systems services, device modeling, and facilities for system configuration and device parametrising.

3.3 CAN Application Layer

CAL (CAN Application Layer) was specified as one of the first work items of CAN-in-Automation (CiA) and was published in 1993 as layer 7 standard CiA DS 201-207.

The protocol offers an application-independent, object-oriented environment for the implementation of CAN-based distributed systems. It provides objects and services for communication, identifier distribution, network and layer management. Main application areas of CAL are CAN-based distributed systems, which do not require configurability and standardized device modeling. Therefore in the CAL specification only general communication procedures are defined as they are required in distributed systems.

CAL provides objects, protocols and services for the event driven or requested transmission of CAN messages and for the transmission of larger data blocks between CAN devices. Furthermore CAL offers mechanisms for the automatic distribution of CAN identifiers and for the initialization and monitoring of nodes.

CANopen and DeviceNet specify also the structure and parts of the application itself mainly by fixed access methods for data exchange and data representation. In contrast to these protocols CAL doesn't define data contents or specific communication objects which a certain device has to provide or which are expected by the system. So the user has the possibility to adapt the communication system exactly to the requirements of his application or system and not the other way round.

Using a communication and device profile like CANopen or DeviceNet could mean higher resource overhead for the realized application. Therefore CAL is suitable for the realization of specific system solutions like medical systems or measuring systems as well as for the realization of closed control systems with decentral intelligent units like machine control systems (machine bus). CAL allows the realization of systems with complex communication relations between devices in a system. These systems can be installed without any configuration effort.

First versions of the CAL software have been available already in 1993. Until today, the software was successfully in use in numerous applications. Hence the software achieved a high and stable development state.

3.4 CANopen

The profile family CANopen defines a standardized application for distributed industrial automation systems based on CAN as well as the communication standard CAL. CANopen is a standard of CAN-in-Automation (CiA) and has already, soon after its release, found a broad acceptance. Especially in Europe CANopen can be considered the leading standard for CAN based industrial system solutions.

The CANopen profile family is based on a so-called "Communication Profile", which specifies the basic communication mechanisms and their description.

The most important device types such as digital and analog I/O modules, drives, operating devices, controllers, programmable controls or encoders, are described by so called "Device Profiles". The device profiles define the functionality of standard devices of the corresponding types. The configuration of devices through the bus is the foundation of the preferred manufacturer-independent configuration by means of the profile family.

The central element of the CANopen standard is the description of the device functionality through an object dictionary (OD). The object dictionary is divided in two sections. The first section contains general device information like device identification, manufacturer name, etc., as well as communication parameters. The second section describes the specific device functionality.

A 16-Bit index and an 8-Bit sub-index identify the entry ("object") in the object dictionary. The entries in the object dictionary provide the standardized network access to the "Application Objects" of a device, such as input and output signals, device parameters, device functions or network variables.

The functionality and characteristics of a CANopen device can be described by means of an "Electronic Data Sheet" (EDS) using the ASCII-format. In this case the EDS must be understood as a kind of template. The actual device settings are described by the so-called "Device Configuration File (DCF)". EDS and DCF can be provided in form of a data carrier, which can be downloaded from the Internet or stored inside the device. Similar to other well-known field bus systems CANopen also distinguishes two basic data transfer mechanisms: The high-speed exchange of small process data portions through so called "Process Data Objects (PDO)" as well as the access to entries in the object dictionary through so called "Service Data Objects (SDO)". The ladders ones are primarily used for the transmission of parameters during the device configuration as well as in general for the transmission of larger data portions. Process data object transmissions are generally event triggered, cyclic or requested as broadcast objects without the additional protocol overhead. A PDO can be used for the transmission of a maximum of 8 data bytes. In connection with a synchronization message, the transmission as well as the acceptance of PDOs can be synchronized through the entire network ("Synchronous PDOs"). The assignment of application objects to a PDO (Transmission Object) is adjustable through a structure description ("PDO Mapping"), which is stored in the object dictionary, thus allowing the adjustment of a device to the corresponding application requirements.

The transmission of SDOs is performed as a confirmed data transfer with two CAN objects in form of a peer-to-peer connection between two network nodes. The addressing of the corresponding object dictionary entries is accomplished by providing the index and the sub-index of the object dictionary entry. Transmitted messages can be of very large length. The transmission of SDO messages involves an additional protocol overhead.

Standardized event-triggered "Emergency Messages" of high priority are reserved to report device malfunctions. A common system time can be provided through a central timing message. The required functionality for the preparation and coordinated start of a distributed automation system is compliant to the under CAL network management (NMT) defined mechanisms. The same applies to the cyclic "Node Guarding".

Alternatively, it is possible to display the communication capability of a CANopen device through a so-called "Heartbeat Message".

The assignment of CAN message identifiers to PDOs and SDOs is possible by direct modifications of identifiers inside the data structure of the object dictionary or, for simple system structures, through the use of pre-defined identifiers.

3.5 DeviceNet

DeviceNet was developed by Rockwell Automation as an open fieldbus standard based on the CAN-protocol. Designed as a powerful protocol for automation technology, it plays a leading role today in the USA and in Asia. More and more systems solutions are also being implemented with DeviceNet in Europe.

The ODVA, being the organization of DeviceNet users, is responsible for the specification and maintenance of the DeviceNet standard. In addition, the ODVA promotes the worldwide distribution of DeviceNet. The currently available version 2.0 of the standard includes more functions and some corrections.

DeviceNet is an open protocol and every ODVA member can participate in the further development of this standard in the various Special Interest Groups (SIGs). Use of DeviceNet is free of charge. It is only necessary to sign the "Terms of Agreement" in order to be able to use the DeviceNet technology and to receive the vendor ID, a manufacturer's number. The only costs normally incurred are the purchase of the specification. Membership of the ODVA is not necessary.

At the moment over 300 companies are registered members of the ODVA. A total of approx. 700 companies offer DeviceNet products.

DeviceNet is one of three open network standards (DeviceNet, ControlNet and EtherNet/IP), which all use a common application layer (ISO Layer 7), the so-called "Control and Information Protocol" (CIP). This common application layer and open software and hardware interfaces will in future enable a universal connection of automation components from the field level with the Internet. The "Control" part of the CIP defines the exchange I/O data in real time via I/O messages (I/O Messaging or Implicit Messaging). The "Information" part of the CIP defines the exchange of general data for configuration, diagnosis and management via explicit messages (Explicit Messaging). These two message types provide optimum communication for industrial controls. CIP thus provides the user with 4 basic functions:

- Uniform control services
- Uniform communication services
- Uniform distribution of messages
- Common knowledge base

The DeviceNet protocol is designed as a simple, inexpensive yet powerful protocol at the lowest fieldbus level, i.e. for the networking of sensors, actuators and corresponding controls. The devices that can be connected to DeviceNet range from a simple light barrier to a complex vacuum pump for semiconductor manufacture.

The core function of the DeviceNet protocol is, as with other protocols, data exchange between devices and their corresponding controls. Communication between two devices is based on a connection-based communication model, either via a point-to-point or a Multicast connection. This allows the development of Master/Slave systems as well as Multi-Master systems.

The so-called "Predefined Master/Slave Connection Set" was specified for simple DeviceNet slave devices. This subset of the DeviceNet protocol supports Explicit Messages, Polled-I/O, Multicast-Polled-I/O and Bit-Strobed I/O messages from the master to the slave as well as Change-of-State/Cyclic I/O messages from the slave to the master. The "Unconnected Message Manager Port" (UCMM) and the dynamic creation of explicit and I/O-connections were specified for more complex slave devices that are Multi-Master-capable and can maintain point-to-point connections with other devices. The Device Heartbeat Message and Device Shutdown Message functions were specified particularly for safety-critical systems. The Offline-Connection-Set simplifies the configuration of off-the-shelf components.

DeviceNet presents communication and application in the object model. Predefined objects facilitate the data exchange of different devices and manufacturers.

Further standardization of benefit to the user was accomplished by creating various device profiles.

Besides Layer 7 (Application Layer), the DeviceNet specification also defines parts of Layer 1 (Transceiver) and Layer 0 (Transmission Media), thus standardizing the physical connection of DeviceNet nodes. Connectors, cable types and cable lengths are specified as well as communication-based displays, operating elements and the corresponding labeling of the housing.

Table 3.1 DeviceNet Layers

ISO-Layer 7	**Application Layer**	**DeviceNet Specification Volume II**
ISO-Layer 2	**Data Link Layer**	**CAN Specification 2.0**
ISO-Layer 1	**Physical Signaling**	
ISO-Layer 1	**Transceiver**	**DeviceNet Specification Volume I**
ISO-Layer 0	**Transmission Media**	

A DeviceNet network can run up to 64 nodes using baud rates of 125, 250 or 500 kBaud. The devices either have their own power supplies or are being supplied through the DeviceNet bus.

Compared to CANopen, DeviceNet provides approximately the same functions, however, with an emphasis on different priorities. The network management, for instance, is stored in each individual node. As a result each node monitors the other. CANopen, however, uses a central authority, the NMT-Master. The communication mechanisms under CANopen are simpler, thus allowing the use of less complex devices. In turn DeviceNet provides higher safety in the protocol use, but does also require more resources.

3.6 Main Items of CAN-based Higher Layer Protocols

In the following the main solutions for industrial automation CAL/CANopen, and DeviceNet will be evaluated closer. This will be done by considering the main items of CAN-based higher layer protocols. These are the

- message identifier assignment system,
- method of exchanging Process Data,
- peer-to-peer communication,
- method of establishing Process Data connections,
- network management,
- principle of device modeling and device profiles

3.6.1 Message Identifier Assignment System

The method of message identifier assignment may be regarded as the major architectural element of CAN-based systems, since the identifier of a CAN-message determines the relative priority of the message and such the message latency time. It also has influence on the applicability of message filtering, possible communication structures and the efficiency of identifier usage.

Concerning identifier assignment quite different philosophies have been chosen in the considered system solutions. Whereas CAL and CANopen, apart from reserving some identifiers for management purposes do not apply a predefinition of identifiers for general system structures, DeviceNet does.

CAL/CANopen provide a common pool of identifiers, available to all devices and a central instance, which automatically or manually allocates identifiers according to the requirements of the devices. With this approach, identifier usage and such the real-time-behaviour of the data communication system may be completely determined through the system designer or integrator. Also maximum usage of the available identifiers is possible since almost the complete set of message identifiers is available for distribution.

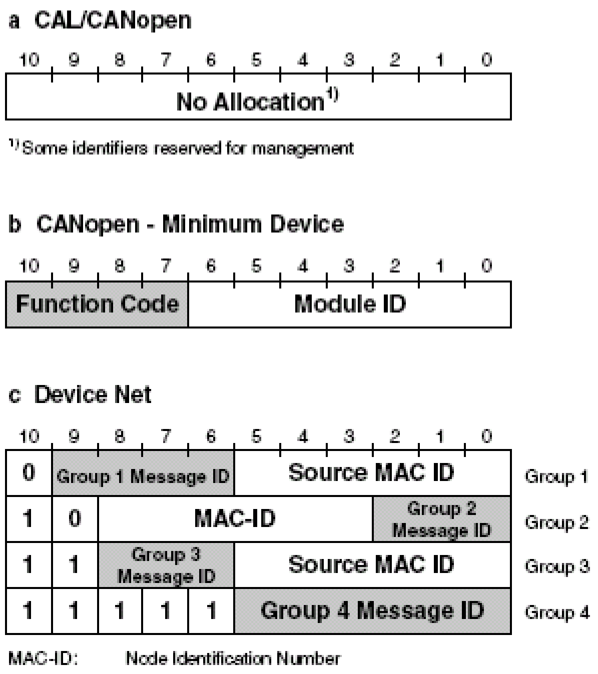


Figure 3.1 Identifier Usage in CAL/CANopen and DeviceNet

In Fig. 3.1.a and Fig. 3.2.a the identifier assignment scheme of general CAL/CANopen systems is shown, in Fig. 3.1.b and Fig. 3.2.b the identifier assignment of a minimum configuration CANopen system is shown with the predefined set of messages. 1760 message identifiers are available for general usage. Since in a CAL-based system up to 256 nodes may be addressed, 256 messages are reserved for node guarding, in CANopen 128 nodes may be addressed, only 128 messages are reserved for node guarding.

In the minimum system configuration, CANopen specifies a device-oriented identifier allocation scheme by which default connections between up to 127 devices to a master device are provided. By means of the 4-bit function code 16 basic functions are distinguished for the reception and transmission of two process data channels, one peer-to-peer channel, node state control, node guarding, emergency notification and the reception of the synchronization and time stamp message. Since the priority of a message should be determined by its function, the function code is located in the most significant bits of the message identifier.

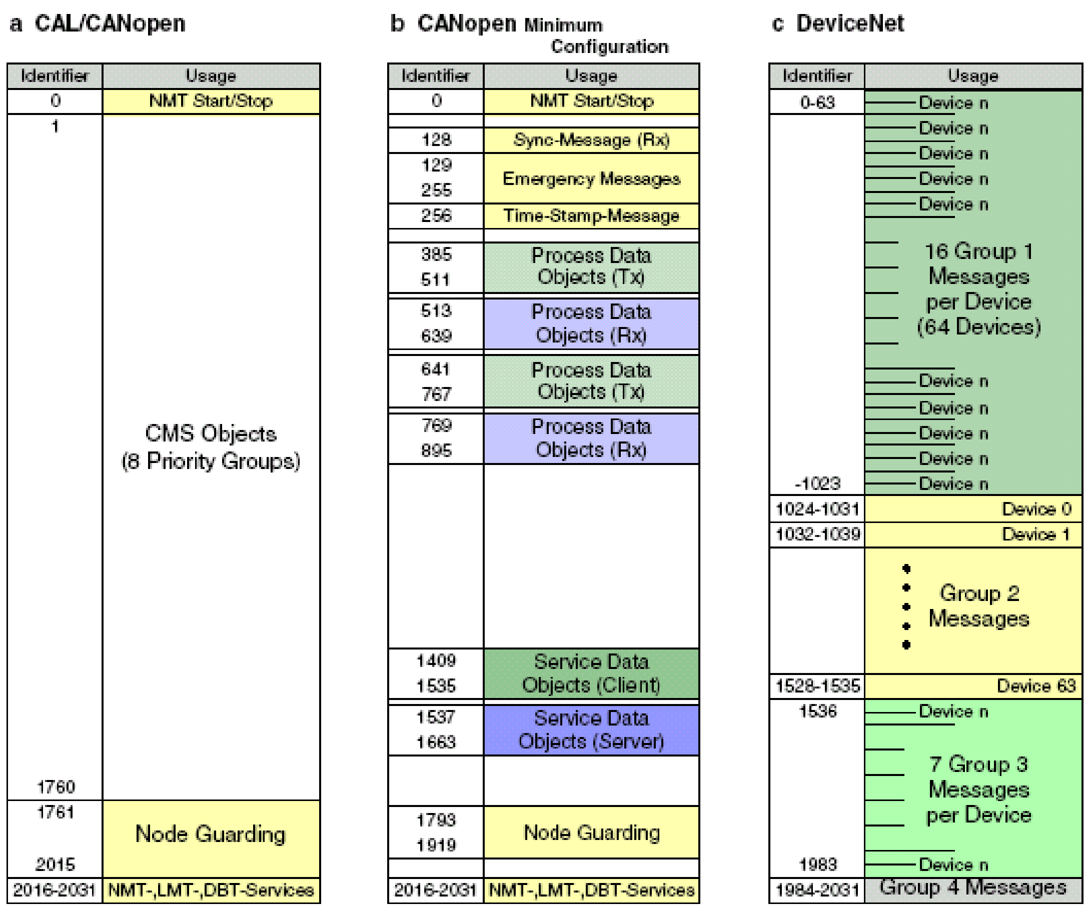
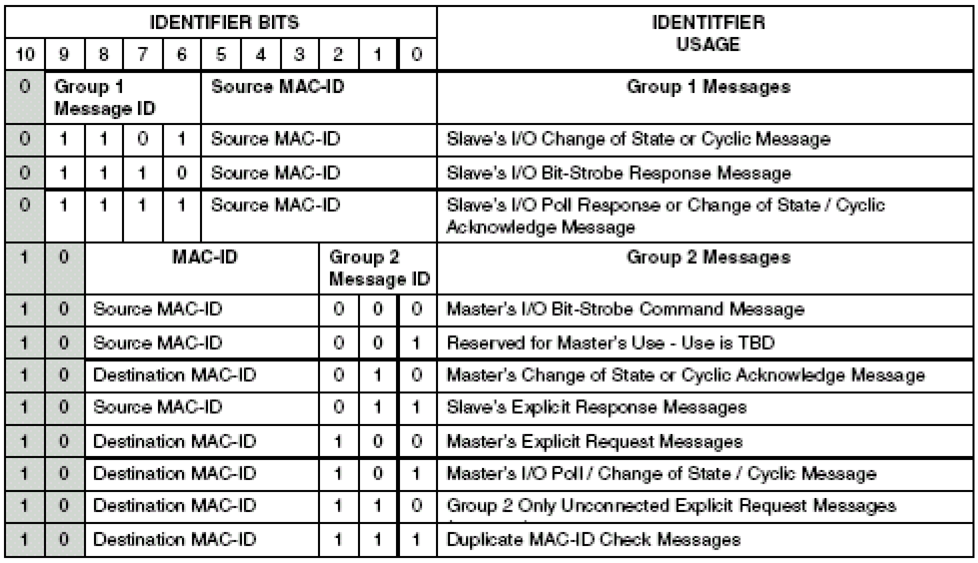


Figure 3.2 Usage of Identifiers in CAL/CANopen and DeviceNet

With the CANopen minimum system configuration a 1:n communication structure is supported per default. By means of identifiers not used by the predefined connection set also direct connection between devices may be established.

One of the basics of the DeviceNet identifier assignment scheme is the node-oriented ownership of message identifiers. Each of the maximum 64 nodes of a DeviceNet system owns a set of identifiers out of 3 message groups (Fig. 3.1.c). Message group 1 provides a high priority message pool of 16 messages per device, message group 3 five (5) low priority identifiers per device. The identifiers/priorities of those groups are distributed evenly among all the devices on the network. The reservation of message identifiers for the maximum number of devices implies, that for networks of less than 64 nodes, the identifiers of the unused nodes are not available for the system.

With Message Group 2 it was intended to support devices with limited message-filtering capabilities due to Basic-CAN type controllers. Therefore, a filtering according to the node number (MAC-ID) was chosen. This means that the priority of messages of that group is primarily determined by the node number. Two messages of that group are reserved for management tasks (allocation of predefined connection set, Duplicate MAC-ID check). The MAC-ID of Group 2 messages may be destination or source address.



IDENTIFIER BITS											IDENTITFIER USAGE
10	9	8	7	6	5	4	3	2	1	0	
0	Group 1 Message ID				Source MAC-ID						Group 1 Messages
0	1	1	0	1	Source MAC-ID						Slave's I/O Change of State or Cyclic Message
0	1	1	1	0	Source MAC-ID						Slave's I/O Bit-Strobe Response Message
0	1	1	1	1	Source MAC-ID						Slave's I/O Poll Response or Change of State / Cyclic Acknowledge 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	Destination MAC-ID						0	1	0	Master's Change of State or Cyclic Acknowledge Message
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 / Change of State / Cyclic 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

Figure 3.3 DeviceNet Predefined Master/Slave Connection Set Identifier Assignment

DeviceNet specifies a so-called "Predefined Master/Slave Connection Set" to facilitate the communication observed in a Master-Slave system configuration.

Fig. 3.3 shows the identifier assignment of this set. The following channel functions are supported for exchanging of I/O and Explicit messages between a Slave and a Master device based on the predefined connection set:

- Explicit Message channel
- Master Poll/Change of State/Cyclic channel
- Slave I/O Change of State/Cyclic channel
- Bit Strobe channel

The Explicit Message Channel mainly serves for configuration of a device. With the Master Poll/Change of State channel the master can request I/O data from the device and send Output data to the Slave device. With the Slave I/O Change of State/Cyclic a Slave device can transmit Input data to the Master, triggered by change of state, cyclically or by the Slaves application. By means of a Bit Strobe command the Master can request input data from any of up to 64 Slave devices with only one message. Since all of these messages are acknowledged 8 message identifiers are allocated for these functionalities. If Bit Strobe requested data acquisition is not used a very effective identifier filtering on the Slave devices is possible by means of the destination address field.

3.6.2 Exchange of Process Data

The transmission of process data between the devices of a distributed automation system is the purpose of a CAN-based communication system. This should be accomplished in the most efficient way. Therefore transmission of application specific data (process data, I/O data) should be performed according to the producer-consumer model, with the meaning of the transferred data implied by the associated message ID. Producer and consumer of a message in that case are assumed to have knowledge of the intended use or meaning of the transmitted data.

In the following, the main characteristics of the different solutions for exchanging of process data will be outlined for CAL, CANopen and DeviceNet.

CAL is intended to provide standard, application-independent communication facilities for the implementation of distributed systems. It provides communication objects (CMS objects) in terms of "Variables", "Events" and "Domains". CMS objects are specified by a set of attributes and are identified by symbolic names. Objects and services of CAL are directly accessible by the user application. Variables may be of type "Basic" or "Multiplexed". With Basic Variables and Events, the multicast transmission of up to 8 bytes of data is provided, without any protocol overhead. Multiplexed Variables contain a multiplexer within the first data byte to distinguish between 128 "Multiplex-Variables" per message identifier and allow the transmission of 7 bytes of data. Basic-Variables also may have different access type (read-only, write-only, read-write). With a read-write Variable, an acknowledged transfer of data between two devices is supported. The transmission of a variable is initiated by a client, the transmission of an Event is initiated by a server of the corresponding object. Fig. 3.4.a shows the "Store-And-Immediately-Notify-Event"-protocol, Fig. 3.4.b the "Read-Event"-protocol, by which a client also can read previously stored data.

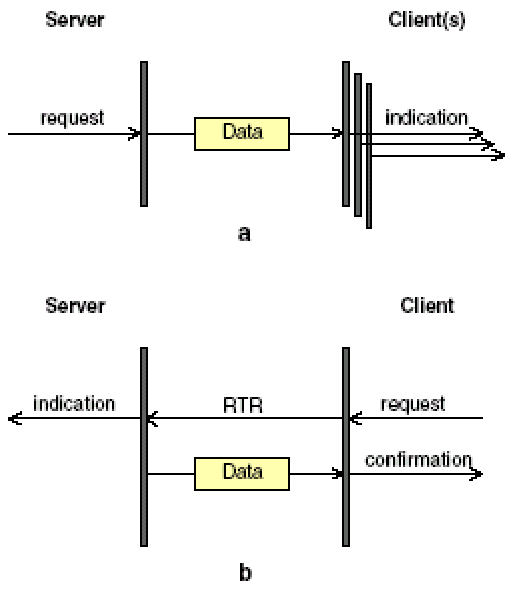


Figure 3.4 Store-and-Immediately-Notify-Event-Protocol (**a**), Read-Event-Protocol (**b**) (CAL/CANopen)

Basic and Multiplexed Domains support the acknowledged transmission of data of more than 8 bytes by means of a flow-controlled fragmented protocol. Since each data segment is acknowledged, a receiver-controlled flow control is implicitly provided. With the 3 byte Domain-Multiplexer a variety of different domains may be identified per message identifier. The transfer of a data block is initiated by an "Initiate"-request/response sequence, following data segments are transmitted by means of a Data Segment request/response sequence. Fig. 3.5 shows the principal structure of the data field of a "Initiate-Multiplexed-Domain" request and a "Download-Segment" request. In the Control byte the message type (Initiate/Download Segment/Upload Segment), the transfer type (expedited / nonexpedited), toggle-bit and number of bytes with no data is indicated. With the expedited (non-fragmented) protocol the transmission of up to 4 bytes of data is possible.

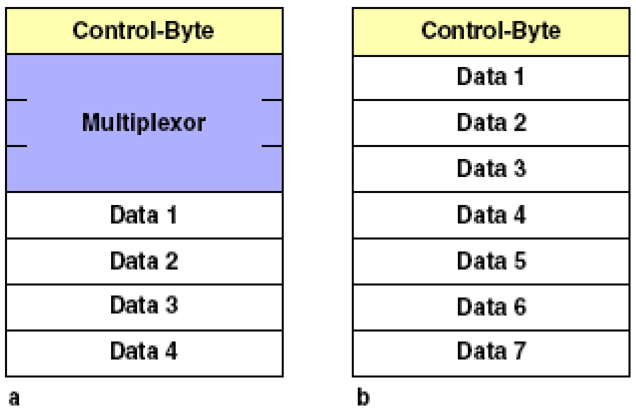


Figure 3.5 Structure of the Data Field of an Initiate_Multiplexed_Domain.request (**a**) and a Download_Segment.request (**b**)

CANopen and DeviceNet at a first glance provide quite similar communication mechanisms for transferring process and service/configuration data. With CANopen, the transmission of process data occurs by means of so-called "Process Data Objects (PDOs)", with DeviceNet by means of "I/O-messages".

Table 3.2 Exchange of Process Data in CANopen and DeviceNet

	CANopen	**DeviceNet**
Name of Communication Object	Process Data Object	I/O-Message
Maximal Number of Communication Objects per Device	512 Transmit PDOs 512 Receive PDOs	27 I/O- Transmit Messages 1701 I/O Receive Messages per device
Maximal length of Data Field	8 bytes	8 bytes fragmented:Arbitrary length

Protocol	Unfragmented: No overhead, Notify/Read "Stored-Event"-protocol (CAL/CMS) Unacknowledged	Unfragmented: No overhead, three "Transport Classes" supported: • Unacknowledged, • Acknowledged by Server Connection Object, • Acknowledged by Application
		Fragmented: Unacknowledged fragmented protocol 1 byte protocol overhead per frame
Message Production Triggering Modes	• On Request of local or remote application • Cyclic/acyclic synchron	• Cyclic • Change-of-State • Application specific
Mapping of Application Objects	Maximum number of mappable application objects/PDO dependent on data size of objects (1-bit objects: 64 application objects mappable) Definition of Application objects by means of "Mapping Parameter Record" (configurable) Dynamic mapping supported	Arbitrary number of Application objects mappable with fragmented protocol Definition of Application objects by means of Assembly Object (several Assembly Objects possible) Dynamic mapping supported

In Table 3.2 the main characteristics of process data exchange are summarized for CANopen and DeviceNet. One of the main differences is the provision of an unacknowledged fragmentation protocol in DeviceNet, which makes it possible to transmit also process data with length more than 8 bytes. Also three different protocols with respect to acknowledgement ("Transport Classes") are supported (Fig. 3.6) and determined of the "Transport Class" of a Connection end point.

Transport class 2 or 3 may be used for example for efficient "polling" of devices. For that purpose a master device implements the communication resources (connection objects) associated with each Poll Command as a Client Transport Class 2 or 3. Each slave implements a Server Transport Class 2 or 3 Connection Object to receive the Poll Command and to transmit the associated response data.

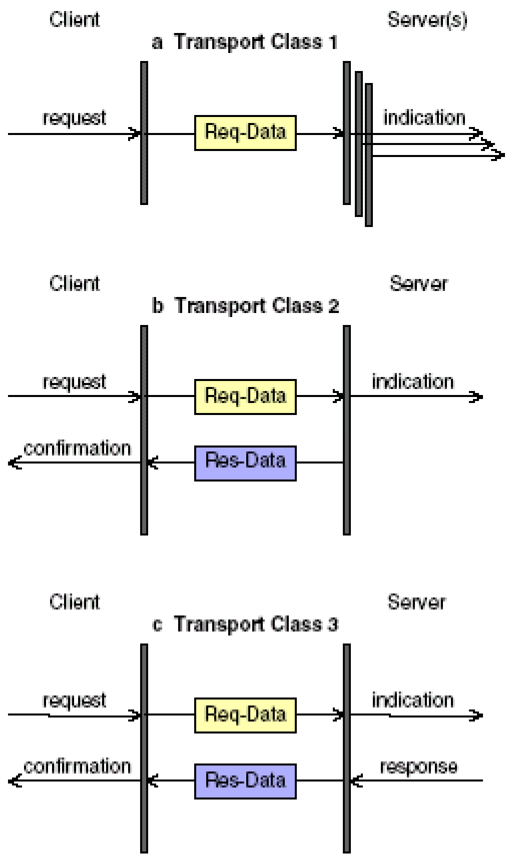


Figure 3.6 DeviceNet Transport Classes

3.6.2.1 Message Triggering

All of the regarded protocols provide alternative modes of message triggering supported by the Application Layers.

DeviceNet supports the triggering modes Cyclic, Change-of-State and Application Object Triggered. With Cyclic Triggering the expiration of a message-specific Transmission Trigger Timer starts the transmission of a message. With Change-of-State the transmission occurs when a change-of-state of an Application Object is detected. A message is also transmitted when a specified time interval has elapsed without a transmission. With Application Object Triggering the Application Object decides when to trigger the transmission.

CANopen distinguishes between triggering On Event, Application Request or after Reception of a predefined Synchronization Message.

Triggering On Event may occur on a profile or application specific event ("Asynchronous PDO"). The transmission of a PDO may also be triggered by the reception of a remote request message (Remotely Requested). "Synchronous PDOs" are cyclically triggered by the reception of a specified number of Synchronization Messages.

The Synchronization Message also may be used for network wide synchronization of data acquisition and the strobing of output data.

3.6.2.2 Mapping of Application Objects

Network devices normally will produce and/or consume more than one Application Object and assembling of more than only one Application Object within one PDO respectively I/O-Message will be appropriate.

In CAL-based applications, the mapping of application data is done by the programmer when defining communication objects (e.g. CMS Variables or Events).

CANopen and DeviceNet provide very sophisticated means for a flexible mapping of application data into communication objects.

CANopen specifies the mapping of Application Objects into a PDO by means of a data structure called "PDO-Mapping Record". This structure specifies the mapped application object data in form of a list of object identifications (Object Directory index/subindex) and data length. Since the PDO Mapping is accessible by means of SDOs, PDO mappings are configurable by means of a configuration tool.

In DeviceNet the grouping of Application data is specified by means of instances of the "Assembly" object, which defines the format of the transmitted application object data. A device may contain more than one I/O assembly and the selection of the appropriate assembly (consumed/produced_connection_path) may be a configurable device option.

3.6.3 Peer-to-Peer Communication Channels

For configuration of devices by means of a configuration tool, specific device functions or program loading multi-purpose communication channels are required. These non-time-critical communication channels always exist between two devices, e.g. between a configuration tool and the device to be configured. The transfer of data has to be performed by means of an acknowledged fragmentation protocol. Any of the different higher layer protocols, which support some kind of device configuration, provide this kind of peer-to-peer communication facility.

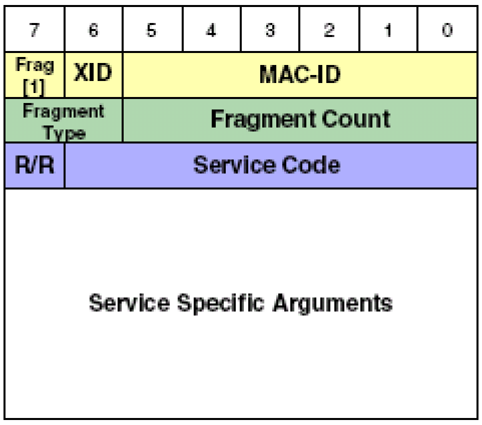
CAL for that purpose provides "configuration services" across predefined management channels to each device as part of the CAL Network Management service element. For that purpose two identifiers are reserved, the addressed device is specified in the data field of the first fragment of a message by its node ID.

CANopen provides so-called "Service Channels" across which "Service Data Objects" (SDOs) may be exchanged between any two devices according to the CAL Multiplexed Domain protocol. This protocol provides the acknowledgement of any frame transmitted. Within the first three bytes of the data field of the Initiate-Domain-Request the address of the Object Directory entry is specified by means of a 16-bit index and 8-bit subindex. With the index/subindex of an Object Directory entry the function to be performed is specified implicitly.

Data of less than 5 bytes may be transferred with only the Initiate-Domain Request Frame ("expedited protocol"), if more than 4 bytes of data have to be transmitted, the acknowledged fragmented protocol has to be applied, with 7 bytes of data per fragment. Each CANopen device has to provide a default server SDO-connection with two predefined message identifiers according to the predefined connection set. Across this default server SDO connection a device may be accessed by a configuration tool.

For applications which require a dynamic establishment of SDO connections (e.g. between test tools and devices) the "SDO-Manager" instance is introduced. The SDO Manager is the owner of the predefined set of SDO connections and therefore has access to any device on the network. A SDO-connection requesting device first has to address the SDO Manager and to ask for establishing the requested connection.

DeviceNet provides multi-purpose device-oriented channels and services. Writing and Reading of object attributes, control of objects (reset, start, stop etc.), storing/restoring of classes/objects attributes etc. is performed by means of "Explicit Messages". These are exchanged across "Explicit Messaging Connections". The meaning/intended use of an Explicit Message is stated in the CAN data field. In Fig. 3.7 the data field format of a fragmented Explicit Message is shown. For unfragmented transfer the "Fragment Byte" is not transmitted. For a service request normally the access path (class number, instance number, attribute number) of the addressed object attribute is specified (Service specific arguments).



Frag	Fragment Bit
XID	Transaction ID
MAC-ID	Source or Destination MAC-ID
Fragment Type	Indicates first, middle or last fragment
Fragment Count	
R/R	Request /Response
Service Code	Type of Service

Figure 3.7 DeviceNet Fragmented Explicit Message
Data Field Format (Request/Response)

An Explicit Message Connection has to be established by means of the "Unconnected Message Manager (UCMM)". The UCCM provides two services for opening and closing of an Explicit Message Connection. Each device supporting an UCMM reserves message identifiers for transmitting UCMM request and response message. For "Group 2 Only" devices (devices not supporting an UCMM port) a master device first has to allocate the Explicit Messaging Connection of the devices' Predefined Connection Set. The request to allocate a Group 2 Only device is transmitted as a Group 2 Only Unconnected Explicit Request with a reserved message identifier.

In Table 3.3 the main characteristics of peer-to-peer communication channels of CANopen and DeviceNet are summarized.

Table 3.3 Main Characteristics of Peer-to-Peer Communication Channels

	CANopen	**DeviceNet**
Name	Service Data Channel	Explicit Message
Maximum number of channels	128 Client SDOs, 128 Server SDOs per device	27 Explicit Transmit Messages 1701 Explicit Receive messages per device
Protocol	< 5 byte: Acknowledged unfragmented > 4 byte: Fragmented transmission (7 bytes per fragment) Each frame acknowledged Any length (CAL Multiplexed Domain protocol)	< 7 byte: Acknowledged unfragmented > 6 byte: Fragmented transmission. (6 bytes per fragment) Each frame acknowledged Any length
Establishing of Connections	Dynamic establishment by means of SDO Manager Default predefined connections	Dynamic establishment by means of Unconnected Message Manager Group 2 Only devices: Allocation of Explicit Message from Predefined Connection Set
Connection Services and Arguments	Initiate, Abort Upload/Download Segment/Domain	Open/Close Creation, Configuration, Start, Stop, Reset etc. of Objects
	Index and Subindex of addressed Object Directory Entry	Object attribute access path, Service arguments

3.6.4 Establishment of Process Data Message Connections

Allocation of identifiers to the transmit messages of the message producers respectively receive messages of the message consumers establishes the communication paths in a CAN network. Establishing message connection is possible through usage of predefined messages with already allocated message identifiers or through a variable allocation of identifiers to messages.

DeviceNet and CANopen also make use of a predefined connection set approach for 1:n system structures. A DeviceNet Master for example which has allocated a Slave device’s predefined poll-connection already "knows" the message IDs for transmitting the poll request and expecting the poll response message since they are derived from the Slaves MAC-ID, according to the predefined set. Similarly in CANopen the default predefined connection set, besides of other predefined messages, provides two predefined Receive and Transmit PDOs. The usage/meaning of the Default-PDOs is determined by the device type.

The main advantages of a non-predefined identifier allocation is the possibility of establishing any type of communication structure, the availability of the maximum number of message identifiers and the design-controlled allocation of message identifiers according to the requirements of the application.

Whereas CAL/CANopen are based on a variable identifier allocation scheme based on a central message identifier pool, DeviceNet distributes the available identifiers across the maximum 64 devices of a DeviceNet system.

The allocation of identifiers with a common identifier pool is controlled by a specific network instance (the Distributor in CAL) or by means of a configuration tool (CANopen), which supports the building of message connections, by the system administrator.

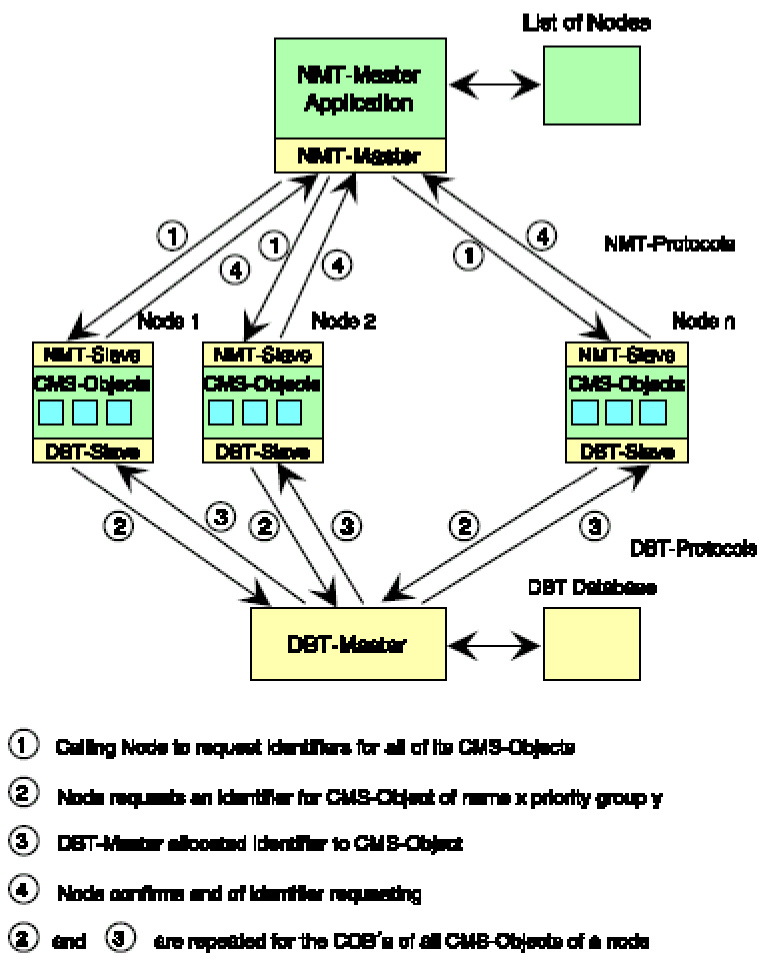

List of Nodes
NMT-Master Application
NMT-Master
NMT-Protocols
Node 1
Node 2
Node n
NMT-Slave
CMS-Objects
DBT-Slave
DBT-Protocols
DBT Database
DBT-Master
1 Calling Node to request identifiers for all of its CMS-Objects
2 Node requests an identifier for CMS-Object of name x priority group y
3 DBT-Master allocated identifier to CMS-Object
4 Node confirms end of identifier requesting
2 and 3 are repeated for the COB's of all CMS-Objects of a node

Figure 3.8 Identifier Allocation Process with CAL Distributor

CAL and CANopen are based on a variable identifier allocation with a common identifier pool. Identifier allocation in CAL-based systems may be performed by the "Distributor" service element (Figure 3.8). The Distributor (DBT) master instance allocates message-IDs from a central pool of message-identifiers according to the requests (priority group, name and type (transmitter/receiver)) of all of the communication objects of the devices. By linking of requests according to object name and type client(s) and server(s) of a message are connected. The distribution process is controlled by a Network-Master application across network management connections to the devices.

If no CAL-Distributor is used, configuration of message identifiers in CANopen based systems may be performed by setting the corresponding PDO identifiers in the Object Directory of the devices via a SDO channel.

The generic identifier allocation method of DeviceNet is determined by the fact that here the devices are owner of message identifier pools. Therefore the connection of

I/O messages first requires the allocation of an identifier out of the identifier pool of the message-transmitting device. This identifier then has to be assigned to the consuming device(s).

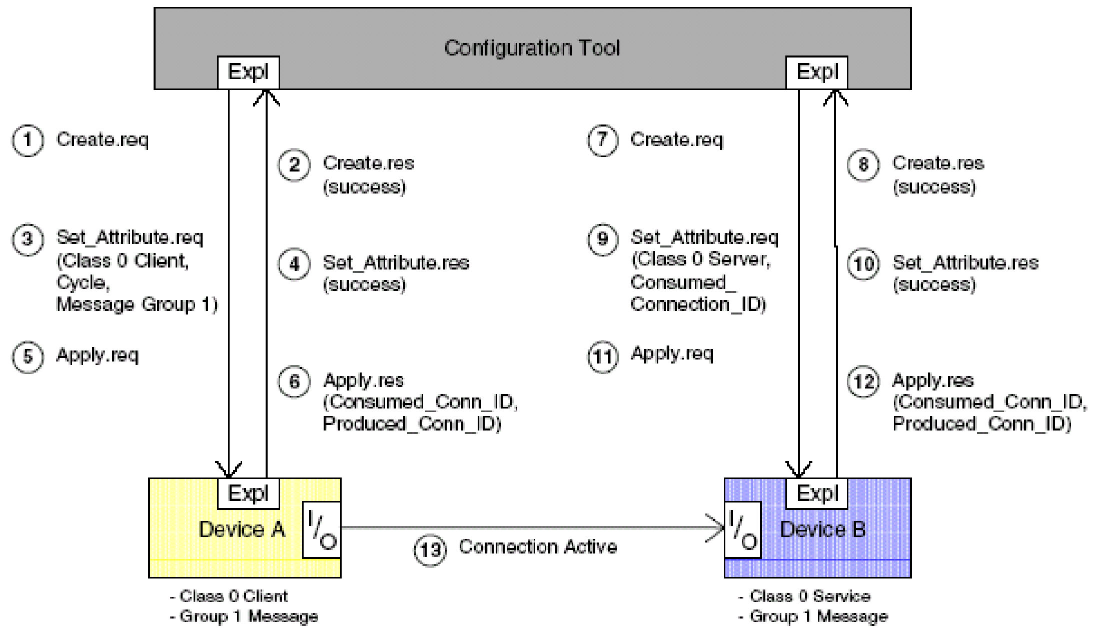

Configuration Tool
Expl
Expl
1 Create.req
2 Create.res (success)
3 Set_Attribute.req (Class 0 Client, Cycle, Message Group 1)
4 Set_Attribute.res (success)
5 Apply.req
6 Apply.res (Consumed_Conn_ID, Produced_Conn_ID)
7 Create.req
8 Create.res (success)
9 Set_Attribute.req (Class 0 Server, Consumed_ Connection_ID)
10 Set_Attribute.res (success)
11 Apply.req
12 Apply.res (Consumed_Conn_ID, Produced_Conn_ID)
Expl
Device A
I/O
13 Connection Active
I/O
Expl
Device B
- Class 0 Client
- Group 1 Message
- Class 0 Service
- Group 1 Message

Figure 3.9 Creation, Configuration of I/O Connection Instances

In Figure 3.9 the process of establishing an I/O Message Connection between two devices by means of a configuration tool is illustrated. I/O connections are established by addressing the Connection Class across an already established Explicit Messaging Connection. This involves creation of an I/O Connection object and configuring the Connection instance at the end point of the connection. During that process a message-producing module allocates a free message-ID from the pool of its message-IDs and combines this with its Source MAC ID to generate a so-called "Connection ID". The "automatic" allocation of an identifier out of the message group may be overridden by a direct modification of the Connection ID attributes.

3.6.5 Network Management

Due to the fact that an application is distributed, certain events have to be handled (e.g. failure of parts of an application or failure of a node) which would not occur if the same application had not been distributed. Main tasks of an appropriate network management therefore are the detection and indication of failures in the network and services, which allow controlling the communication status of the distributed nodes in a coordinated manner. Depending on the system solution, network management functionality is provided by means of an explicit Network Management facility or implicitly by means of other measures.

CANopen network management is based on the CAL NMT service element, which applies the principle of "Node Guarding" for the detection of node failures. For this purpose, a NMT master application cyclically transmits a guard request to each node (NMT slave) of the network by means of a Remote-Request Frame. The addressed slave responds to each request with its actual communication state. If the NMT master detects a change in the node state or no response from the addressed node, a guard error is indicated to the NMT master application. Node guarding starts, when a node is connected to the network. Each node also supervises the arrival of its guard request message. If there is no further guard request after expiration of the nodes "life time" a network error is signaled to the nodes application.

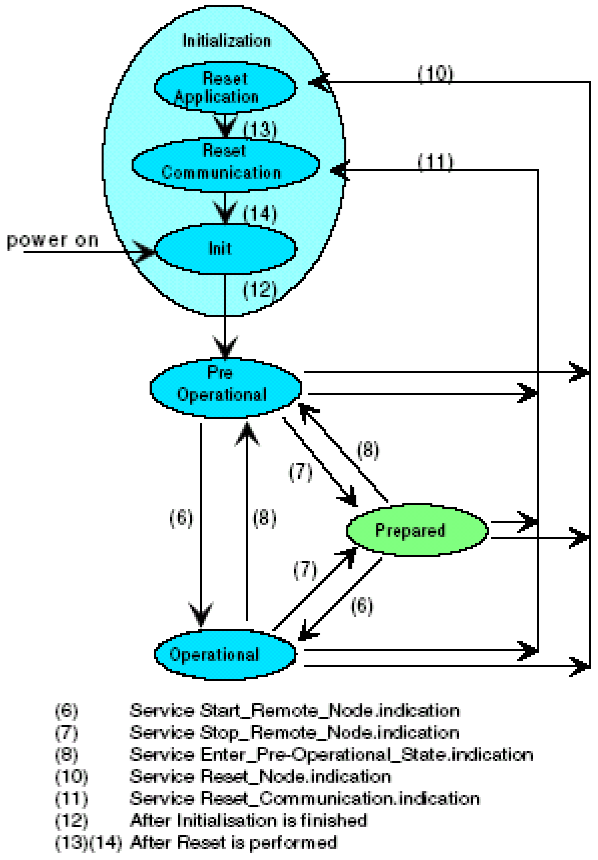

Initialization
Reset Application
(13)
Reset Communication
(14)
power on
Init
(12)
(10)
(11)
Pre Operational
(8)
(7)
(6)
(8)
Prepared
(7)
(6)
Operational
(6) Service Start_Remote_Node.indication
(7) Service Stop_Remote_Node.indication
(8) Service Enter_Pre-Operational_State.indication
(10) Service Reset_Node.indication
(11) Service Reset_Communication.indication
(12) After Initialisation is finished
(13)(14) After Reset is performed

Figure 3.10 CANopen Node State Diagram

Co-ordination of the communication status of the nodes is also supported by the NMT master instance. Fig. 3.10 shows the node state transitions diagram of a

CANopen node. After power on, a node initializes and transits to the "Preoperational State". In this state communication across SDO channels is possible for node configuration, but not yet across PDOs. With the NMT message "Start Remote Node" a selected or any nodes on the network can be set into the "Operational State". In this state also the exchange of data by means of PDOs is possible. With enabling the operation of all nodes of a network at the same time a co-coordinated operation of the communicating system is secured.

According to its connection-oriented design, in DeviceNet each connection is supervised. Therefore each receiving connection end point owns an "Inactivity/Watchdog-Timer" to supervise the arrival of a message according to the configured "expected packet rate". If the timer expires the connection performs the specified "Timeout Action". Fig. 3.11 shows the state transitions diagram for an I/O connection object. After reception of a Create Service (Explicit Message) the connection is configured by applying the appropriate sequence of Explicit Message services and enabled after the complete connection has been configured.

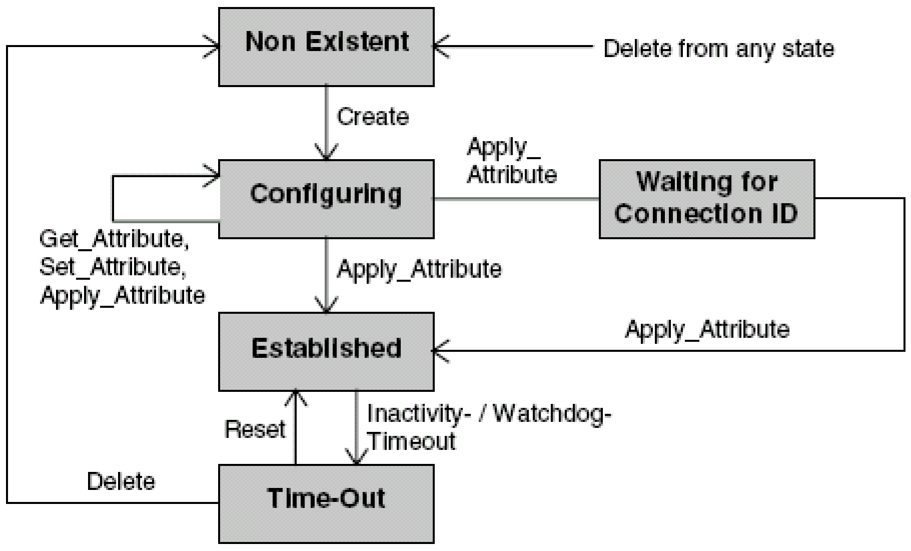


Figure 3.11 Device Net I/O Connection Object State Transition Diagram

Prior to getting access to the network every DeviceNet node has to perform the so-called "Duplicate MAC ID Check". With this specific protocol sequence the uniqueness of the MAC ID of a device is secured. All DeviceNet modules are required to participate in this MAC ID detection algorithm.

An optional means for the supervision of devices is provided by means of a "Heartbeat-Message" which may be broadcasted by the devices by means of the

UCMM in form of an Unconnected Response Message or by Group 2 only devices by means of an Unconnected Response Message. In the data field of this message the device state is transmitted. The Heartbeat Message is triggered by the Identity Object. A node may optionally broadcast fault information before going offline.

3.6.6 Device Modeling and Device Profiles

For open automation systems, besides of standard communication, in addition interoperability and interchangeability of alike device is demanded. Therefore open systems higher layer protocols like DeviceNet and CANopen describe the functionality of devices as seen from the network in form of a "Device Model". To promote the interchangeability of alike devices "Device Profile" of main device classes of industrial automation have to be specified which secures the same basic ("standard") behavior of devices of different manufacturers.

Beside of a description of the functionality of the device the device model must also provide a description of the device's identity, version number, status, diagnostic information, communication facilities and configuration parameters.

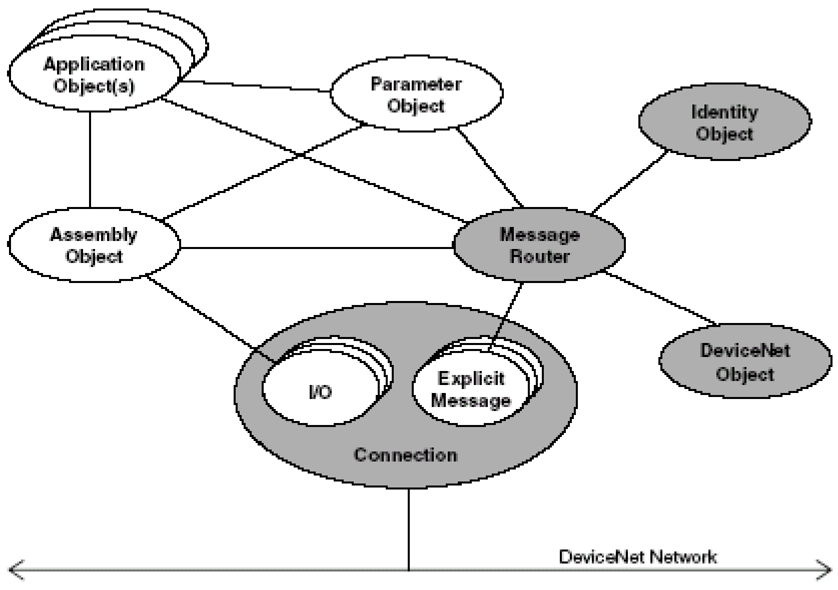


Figure 3.12 DeviceNet Object Model

In Fig. 3.12 the model of a DeviceNet node is shown. This includes several objects, some required by DeviceNet, and others required by the product's application function. An object provides an abstract representation of a particular component within a device and represents the related data (attributes) and procedures (services) on that data. In Table 3.4 the main function of the objects of a DeviceNet example node are summarized.

Table 3.4 Objects of a DeviceNet node

Object	Function
Connection	Instantiates connections (I/O or Explicit Messaging)
DeviceNet	Maintains configuration and status of physical attachments to DeviceNet.
Message Router	Routes received Explicit Messages to appropriate target objects
Assembly	Groups attributes of multiple objects into a single block of data, which can be sent and received over a single connection
Parameter	Provides a standard means for device configuration and attribute access
Identity	Provides general information about the identity of a device
Application	Supplies application-specific behaviour and data

Object addressing in DeviceNet is based on a hierarchical addressing scheme and consists of the MAC-ID (Medium Access Control Identifier), which distinguishes a node among all other nodes on the same link, the class identifier (Class ID), which identifies the object class, the instance identifier (Instance ID), which identifies an instance among all instances of the same class and the attribute identifier which identifies an attribute within a class or instance.

A DeviceNet device profile must contain the following information:

- an object model for the device type
- the I/O data format for the device type
- configuration data and the public interfaces to that data

The CANopen approach is based on the description of a device's functionality by means of an "Object Directory". Entries of the Object Directory are identified by a 16-bit index and an 8-bit subindex number with the function of an entry (data, parameter or function) implicitly specified. Beside of a section used for the definition of data types, three main sections are distinguished (Fig. 3.13): The Communication Profile Section, Standardized Device Profile Section and Manufacturer Specific Section.

Index (hex)	Object
0000	Not used
0001-009F	Definition of static, complex, manufacturer-specific and device profile-specific data types
00A0-0FFF	Reserved
1000-1FFF	Communication Profile Area
2000-5FFF	Manufacturer Specific Profile Area
6000-9FFF	Standardised Device Profile Area
A000-FFFF	Reserved

Figure 3.13 CANopen Object Directory Structure

Index	Object Class	Object
1000h 1001h	VAR VAR	Device Type Error Register Device Specification Data Device Global Communication Parameters Number of supported PDOs and SDOs
1200h	Record	1st Server SDO Parameter ↓ 128th Server SDO Parameter
1280h	Record	1st Client SDO Parameter ↓ 128th Client SDO Parameter
1400h	Record	1st Receive PDO Parameter ↓ 512th Receive PDO Parameter
1600h	Record	1st Receive PDO Mapping ↓ 512th Receive PDO Mapping
1800h	Record	1st Transmit PDO Parameter ↓ 512th Transmit PDO Parameter
1A00h	Record	1st Transmit PDO Mapping ↓ 512th Transmit PDO Mapping

Figure 3.14 CANopen Object Directory
Communication Profile Section

The Communication Profile Section Information is identical for any CANopen device type and contains device related information, parameters and functions that are related for device identification, error management and the definition of the device's communication channels including the mapping of application objects into Process Data Objects (Fig. 3.14). Related to DeviceNet the CANopen Communication Profile Section may be compared with the functionality of the DeviceNet, Identity, Connection and Assembly Objects.

The CANopen Device Profile Section provides the interface to the functionality of a basic ("standard") device of a specific class. Some of these entries are mandatory and some are optional. The mandatory, common entries shall ensure, that a device behaves in a defined basic manner. Different Device Profiles for main industrial devices like I/O modules or drives are specified to promote interchangeability of devices.

Manufacturer specific or non-standardized device functionality may be provided by means of the Manufacturer Specific Profile Section.

4. NEW PROTOCOL DEVELOPMENT: CANUP

CANUP (CAN Uygulama Protokolü) is application-independent CAN-based application layer protocol which defines general rules and structures for message identifier allocation, data exchange methods, network management, message triggering and baudrate.

Protocol defines only one master and maximum 127 slaves total 128 units for a network. Master is responsible and only authority for network management, baudrate settings, ID and Unit Number assignment.

In a CANup based system every unit has a Unit Number that may be the same but theoricaly is independent from message ID.

4.1 Baudrate

All units on network must support the baudrate 25 kbit/sec. This baudrate is used at startup and during initialization process. Beside, any unit must also support at list one of the defined baudrates at Table 4.1.

Table 4.1 Defined baudrates and codes

BAUDRATE	CODE
10 kbit/s	10
25 kbit/s	25
50 kbit/s	50
100 kbit/s	100
125 kbit/s	125
250 kbit/s	250
500 kbit/s	251
800 kbit/s	253
1 Mbit/s	255

4.2 ID Assignment

CANUP supports both static and dynamic message identifier distribution. Basically 11-bit standard IDs are used. But protocol also supports 29-bit extended ID usage.

11-bit standard ID field is divided into 2-main sections.

10	9	8	7	6	5	4	3	2	1	0
TYPE			MESSAGE ID							

Figure 4.1 General ID Fields

Although function of these two fields depends on application, they have some general meanings. Upper 3-bit field named as TYPE determines message type or message group. Lower 8-bit field has different functions such as sender no, target no, and message content.

If constant ID is used for a unit, ID must be assigned by taking consideration about priorities for both fields.

For nonconstant ID units each unit has a startup ID. Type field is set according to device priority group and message ID has a random value.

4.2.1 System Messages

255 messages having the highest priority with TYPE field all zero are called as System Messages. These messages are application-independent and can be used for only defined purposes. System messages are classified into 4 groups.

The messages with ID between 0 and 9 are Emergency Messages for master and slave units.

The messages with ID between 10 and 59 are System Messages that can be used only by master.

The messages with ID between 59 and 99 are System Messages that can be used only by slave units.

The messages with ID as 100+Unit Number (100-227) are Special System Messages that can be sent by specific Unit.

4.2.1.1 Emergency Messages

Message 0

SENDER	ID	DLC	DATA0	Meaning
Slave	0	1	Unit No	Leaving from network

Message 1

SENDER	ID	DLC	DATA0	Meaning
Slave	1	1	Unit No	TEC exceeded 245

Message 2

SENDER	ID	DLC	DATA0	Meaning
Slave	2	1	Unit No	REC exceed 245

Message 3

SENDER	ID	DLC	DATA0	Meaning
Slave	3	1	Unit No	TEC exceeded 127

Message 4

SENDER	ID	DLC	DATA0	Meaning
Slave	4	1	Unit No	REC exceed 127

Message 5

SENDER	ID	DLC	DATA0	Meaning
Slave	5	1	Unit No	TEC exceeded 96

Message 6

SENDER	ID	DLC	DATA0	Meaning
Slave	6	1	Unit No	REC exceed 96

Message 7

SENDER	ID	DLC	DATA0	Meaning
Slave	7	1	Unit No	Unable to support new baudrate

4.2.1.2 General System Messages

Message 10

SENDER	ID	DLC	Meaning
Master	10	0	All Units start communication

Message 11

SENDER	ID	DLC	Meaning
Master	11	0	All Units Stop Communication

Message 12

SENDER	ID	DLC	DATA0	Meaning
Master	12	1	Baudrate Code	New Baudrate

Message 13

SENDER	ID	DLC	DATA0	Meaning
Master	13	1	Unit Number	The unit joined network

Message 14

SENDER	ID	DLC	DATA0	Meaning
Master	14	1	Unit Number	The unit start communication

Message 15

SENDER	ID	DLC	DATA0	Meaning
Master	15	1	Unit Number	The unit left network

Message 16

SENDER	ID	DLC	DATA0	Meaning
Master	16	1	Unit Number	The unit stop communication

Message 17

SENDER	ID	DLC	DATA0	Meaning
Master	17	1	Unit Number	LongData transmission allowed.

Message 18

SENDER	ID	DLC	DATA0	Meaning
Master	18	1	Unit Number	LongData transmission not allowed.

Message 19

SENDER	ID	DLC	Meaning
Master	19	0	The units with constant ID, Introduce yourselves

Message 20

SENDER	ID	DLC	DATA0	Meaning
Master	20	1	Unit Number	Unit number assignment

Message 21

SENDER	ID	DLC	Meaning
Master	21	0	The units with dynamic ID, Introduce yourselves

Message 22

SENDER	ID	DLC	DATA0	Meaning
Master	22	1	Unit Number	Unit number assignment

Message 23

SENDER	ID	DLC	Meaning
Master	23	0	Baudrate determination process started.

Message 24

SENDER	ID	DLC	DATA0	Meaning
Master	24	1	Baudrate Code	Baudrate offer

Message 25

SENDER	ID	DLC	DATA0	Meaning
Master	25	1	Unit Number	Send your state information.

Message 59

SENDER	ID	DLC	DATA0	Meaning
Slave	59	1	Unit Number	Unit number confirmation

Message 60

SENDER	ID	DLC	DATA0	Meaning
Slave	60	1	Priority	A unit wants to join network

Message 61

SENDER	ID	DLC	Meaning
Master	61	0	Faulty node recovering

4.2.1.3 Unit System Messages

DLC is always greater than zero in Unit System Messages. The first byte of data (DATA0) always defines type or meaning of message. So 255 different messages can be defined for each unit. General format of the messages are below:

Message 100+x

SENDER	ID	DLC	DATA0	DATA1	DATA2	....DATAn
Unit x	100+X	>1	Message Type			

x=Unit Number (0-127)

Figure 4.2 Unit System Message Format

Message Type 0

DLC	DATA0	DATA1	DATA2	Meaning
3	0	Length (MSB)	Length (LSB)	LongData Transmission Request

Message Type 1

DLC	DATA0	DATA1	DATA2	Meaning
3	1	Unit Number	Baudrate Code	I only support this baudrate

Message Type 2

DLC	DATA0	DATA1	DATA2	Meaning
3	2	REC	TEC	State Information

4.3 Data Exchange

Data transmission is divided into 4 different methods according to length:

- Instant (Logical) Data (0 byte)
- Short Data (1-8 bytes)
- Mid-Data (9-48 bytes)
- Long Data (49-1785 bytes)

4.3.1 Instant Data

In this type of transmission DLC is 0(zero). Data is carried by some bits of ID field.

4.3.2 Short Data

DLC shows amount of total data bytes. All data is transmitted in one package. DLC can be maximum 8.

4.3.3 Mid-Data

DLC takes the values between 9 and 14. Total data is transmitted by 2 to 6 packages. Package segment number is calculated by following formula:

Package Order = DLC – 8

DLC=9 >>>> Package Order = 9 - 8 = 1

DLC=14 >>>> Package Order = 14 - 8 = 6

4.3.4 Long Data

DLC is always 15. All packages contain 8 bytes of data. Data0 content shows package order.

Startup (first) package has different format. The content of Data0 is 0(zero). Data1 and Data2 content show total data number of bytes that will be transmitted. Data3...Data7 content has no meaning.

Data 0	0
Data 1	Length LSB
Data 2	Length MSB
Data3-7	---

Figure 4.3 Long Data Startup Package Format

Because producer and consumer of long data know about total length, unused data bytes of last package are ignored.

4.4 Network Management

4.4.1 Startup

Initialization is divided into two steps: Registration and baudrate determination process.

First of all, master broadcast message 12 with the baudrate code of 25.

Then master generates message 11 twice and starts registration process. Master broadcasts message 19 to know if there is any unit with constant ID. If there is no message within 20-bit time, master assumes there is no unit with constant ID.

The units with constant ID answer to master using data frames with their ID and without data field. Here there cannot be two units with same constant ID and this is responsibility of system designer.

A slave unit, which lost arbitration during this process aborts transmission and waits until completion of registration process for arbitration winner unit.

Master answers each message with message 20. By this way every units gets a Unit Number. Master is free about assigning unit number. For constant ID units master assigns a unit number related to constant ID. The same unit produces message 59 by using assigned Unit Number for confirmation. So registration process for a slave unit with constant ID has been completed.

Master holds a table called Registered Units List (RUL). Master adds every unit to RUL after registration process.

After broadcasting of message 59, master unit waits 20-bit time and if there is no more messages it assumes that registration process for constant ID units completed.

Master unit broadcasts message 21 for units with non-constant ID to introduce themselves. The units get the message and wait for the random bit time that is defined in the lower 8-bit of their startup ID and broadcast data frames with their ID and without data field.

A slave unit, which lost arbitration during this process aborts transmission and restarts random wait time.

Master answers each message with message 20. Master assigns unit numbers from low to high by order. Then slave unit answers with message 59 and registration completed. All units also are added to RUL.

After broadcasting of message 59, master waits 255-bit time and finishes whole registration process.

Second step at initialization is baudrate determination. Broadcasting of message 23 by master starts this process. Master waits for 10-bit time for the units, which supports only one baudrate. If there is any unit that supports only one more extra baudrate, then it suddenly broadcasts unit system message 1 with supported baudrate code. Then master broadcasts message12 using the same baudrate code and baudrate determination process finishes.

If there is no unit with single baudrate, master firstly offers highest baudrate for network by using message 24 and starts to wait for 15-bit times. Any unit that cannot support offered baudrate, broadcasts emergency message 7. Then master offers one lower baudrate. By this way, the baudrate that is not rejected by any unit is determined as new baudrate and broadcasted by message 12. And baudrate determination process finishes.

4.4.2 Node Guarding

For node guarding purposes master units hold two tables: Active Units List (AUL) and Passive Units List (PUL). A unit can be member of AUL or PUL not both at any time. After initialization all units in RUL are also added to AUL.

After completing initialization, master starts a periodic timer, which counts the time called as Cycle Time. Cycle time is calculated by following formula:

Cycle Time = Registered Unit number * 5

During Cycle time, the units, which broadcasted any message is marked as active unit. When cycle time finishes, master starts to send node-checking message (message 25) to the units that are in AUL but not marked as active unit by unit number order. A unit that takes this message, answers it by sending unit message type 2. This message includes current state of unit. If there is no answer in 15-bit time then the unit is transferred from AUL to PUL.

After every 10-cycle time, master sends node-checking message to all units in PUL. After every message, master waits 15-bit time for answer. Any units that answered this message by unit message type 2 are again transferred to AUL.

4.4.3 Detecting and Recovering Faulty Nodes

A unit broadcasts emergency message 5 as soon as Transmit Error Counter (TEC) exceeds 96 and message 6 as soon as Receive Error Counter (REC) exceeds 96. In this case master broadcasts message11 10 times then faulty unit broadcasts message61 10 times consecutively. Then master broadcasts message10 and recovering process ended.

A unit broadcasts emergency message 3 as soon as Transmit Error Counter (TEC) exceeds 127 and message 3 as soon as Receive Error Counter (REC) exceeds 127. In this case master broadcasts message11 20 times then fault unit broadcasts message61 20 times consecutively. Then master broadcasts message10 and recovering process ended.

A unit broadcasts emergency message 1 as soon as Transmit Error Counter (TEC) exceeds 245 and message 2 as soon as Receive Error Counter (REC) exceeds 245. In this case master broadcasts message11 30 times then fault unit broadcasts message61 30 times consecutively. Then master broadcasts message10 and recovering process ended.

For a faulty unit maximum 5 consecutive recovering processes can be applied. If error state of the faulty node continues no more recovering process is applied and master reports this state to user. In such cases measures are application-dependent. But decreasing baudrate or inhibiting faulty node from communicating by sending message16 may be recommended.

4.4.4 Node Adding-Removing

A unit that will join to a running network listens bus at all baudrates from high to low until detecting current baudrate. Then unit sends message 60 to join network. Master replies this message by sending message 13 and this message includes unit number for new unit. Master sends message 14 to allow new unit to start communication. The unit is added to both RUL and AUL.

A unit that will shut down or leave from network sends emergency message 0. Master also sends message 15 to inform other units. The unit is removed from RUL and AUL or PUL.

4.5 Message Triggering

The messages on a CANUP network are classified into two types: System Messages (SM), Process Messages (PM). SM and PM transmission of a unit is triggered in 3 modes: Event-Driven, Periodically and Remotely Requested.

4.5.1 Event-Driven

Message transmission is triggered by the occurrence of an application specific event. This event may be change of state, expiration of a specified period etc.

4.5.2 Periodically

Message transmission is triggered by expiration of the application specified transmission period.

4.5.3 Remotely Requested

Message transmission may be initiated on receipt of a remote request made by another unit.

5. A CANUP APPLICATION: LIFT COMMUNICATION PROTOCOL ASCAN

There are several units, which communicate each other in different methods in lift control systems. Classical systems use point-to-point wiring and all units are connected to main control panel directly. For example for a 10-stop lift system there are approximately 30 cables from control panel to cabin unit and 20 cables to each floor push button unit. This costs much wiring and workmanship and causes difficulties while expanding system and increases wrong connection rate.

Serial communication looks like the most effective solution in all aspects. CAN is chosen as serial communication bus because of its high reliability and cost efficiency.

There are 3 different serial lines in lift control systems. One line for communication between control panel, cabin and floor push buttons, one line for communication between control panel and sensors, speed control units, one line for between control panels in group operations. In this CANUP application called as ASCAN, all these lines are gathered on a single CAN network.

5.1 Baudrate

ASCAN supports the defined CANUP baudrates lower than 250 kbit/s. Because urgent (time-critical) data in lift control systems such as security circuit signals are connected to main control panel directly, so these low baudrates will be enough. As the number of units and consequently network length increase, baudrate also decreases.

For communication of control panel and floor push button units recommended baudrate is 50 kbit/s.

5.2 ID Distribution

The units in lift control system use defined type of IDs as in Table 5.1.

Table 5.1 ID Distribution in ASCAN

TYPE FIELD		MESSAGE
000	0	System Messages
001	1	Control Panels
010	2	Sensors
011	3	Smart Unites
100	4	Cabins
101	5	Floor Units 1
110	6	Floor Units 2
111	7	Common

5.3 Messages On ASCAN

5.3.1 Control Panel Messages

Control panel (CP) is master of all system. It uses Type 0 IDs for network management messages and uses Type 1 IDs for lift control messages. In group operation there can be up to 16 control panels. In such cases, CP0 is master of all system.

CP message ID format is described in Figure 5.1

Type Field			Control Panel Number				Message Number			
ID10	**ID9**	**ID8**	**ID7**	**ID6**	**ID5**	**ID4**	**ID3**	**ID2**	**ID1**	**ID0**
0	0	1	x	x	x	x	x	x	x	x

Figure 5.1 Control Panel Message ID Format

Message 0 is broadcasted to inform all units about system parameters. There are 2 bytes of data in the message. Data0 carries number of stops in system and Data1 carries traffic system code as explained in Table 5.2

Table 5.2 Traffic System Codes

CODE	Traffic System
0	Simple Push Button
1	Simple Collective
2	Down Collective
3	Up Collective
4	Full Collective

For floor push button and cabin units CP broadcasts 3 similar messages: Message number 1,2 and 3. In these messages there are 5-8 bytes of data and Data0..4 are in the same format as in Figure 5.2.

DATA0	DATA1	DATA2	DATA3
Left Display Code	Right Display Code	Signal Lamps State	Cabin State

Figure 5.2 Data 0..4 of CP message

Data0 and Data1 bytes carry display codes that are in ASCII format for left and right displays at cabin and floor display units.

Data2 carries Signal Lamps State information for signals lamps at cabin and floor units, which has a format as in Figure 5.3.

B7	B6	B5	B4	B3	B2	B1	B0
X	Minimum Load	Full Load	Overload	Up Arrow	Down Arrow	Busy	Out Of Service

Figure 5.3 Signal Lamps State Byte Format

For each bit logic 1 means signal is active and logic 0 means signal is off.

Data3 carries information about cabin's motion state code. Here two types of codes are used. Decimal values up to 127 gives 10 times cabin speed in m/s. To calculate actual speed the value is divided by 10. For example decimal 25 means that speed is 2.5 m/s.

When msb bit of the byte is 1 (Data3 > 127) then lsb 4 bits are read as in Table 5.3

Table 5.3 Cabin State Byte

CODE					EXPLANATION
b3	b2	b1	b0	Dec	
0	0	0	0	0	Cabin stops.
0	0	0	1	1	Cabin is about to move.
0	0	1	0	2	Cabin is moving slowly.
0	1	0	0	4	Cabin is moving fast.
1	0	0	0	8	Cabin is moving very fast.

Number of data bytes in the messages depends on the number of stops in lift system. For every 8 stops one more byte is added to data field. For an 8-stop system there are 5 data bytes and for 32 floors there are 8 data bytes.

Format of Data5..7 differs at every messages as in Figure 5.4. Message 1 carries CRL information, message 2 carries URL information and message 3 carries DRL information.

DATA4	DATA5	DATA6	DATA7
CRL 0-7	CRL 8-15	CRL 16-23	CRL 24-31

CRL: Cabin Registration Lamps

DATA4	DATA5	DATA6	DATA7
URL 0-7	URL 8-15	URL 16-23	URL 24-31

URL: Up Registration Lamps

DATA4	DATA5	DATA6	DATA7
DRL 0-7	DRL 8-15	DRL 16-23	DRL 24-31

DRL: Down Registration Lamps

Figure 5.4 Format of Data4..7 of CP messages

Master also broadcasts second types of messages, message 4 and 5 for answering password-checking messages of floor units and cabin. These messages contain 1 byte of data, which carries the unit number of answered unit. Message 4 means password is correct and call request has been accepted. Message 5 means password is invalid and call request has been rejected.

5.3.2 Floor Unit Messages

Floor Units broadcast 2 types of messages: One type of message for reporting change of states of buttons and one for password checking.

Floor Units message ID format is described in Figure 5.5.

Type Field			**Floor Unit Number**					**Message Type**		
ID10	**ID9**	**ID8**	**ID7**	**ID6**	**ID5**	**ID4**	**ID3**	**ID2**	**ID1**	**ID0**
1	0	1	x	x	x	x	x	x	x	x

Figure 5.5 Floor Unit Message Format

Floor Unit broadcasts the message as in Figure 5.6 to inform master about state change of buttons.

Type Field			**Floor Unit Number**					**Message Type**		
ID10	**ID9**	**ID8**	**ID7**	**ID6**	**ID5**	**ID4**	**ID3**	**ID2**	**ID1**	**ID0**
1	0	1	x	x	x	x	x	0	Up	Down

Figure 5.6 State-Change Message of Floor Units

This message is in Instant Data form and has no data field. ID2 bit of message ID is always 0. ID1 shows the last state of Up-button and ID0 shows the last state of Down-button. For these bits logic 1 means that button is pressed and logic 0 means button is released.

For password checking messages, message ID format is described in Figure 5.7.

Type Field			**Floor Unit Number**					**Message Type**		
ID10	**ID9**	**ID8**	**ID7**	**ID6**	**ID5**	**ID4**	**ID3**	**ID2**	**ID1**	**ID0**
1	0	1	x	x	x	x	x	1	Up	Down

Figure 5.7 Password-checking Message ID format

ID2 bit is always logic1. Logic 1 in ID1 bit shows up call request and in ID0 bit down call request for that floor. The message contains two bytes of data. Data0 carries upper byte of password and Data1 carries the lower byte.

5.3.3 Cabin Messages

Similar to floor units, cabin also broadcasts 2 types of messages: One type of message for reporting changes of state of buttons and one for password checking.

Cabin message ID format is described in Figure 5.8.

Type Field			**Cabin Number**				**Message Type**			
ID10	**ID9**	**ID8**	**ID7**	**ID6**	**ID5**	**ID4**	**ID3**	**ID2**	**ID1**	**ID0**
1	0	0	x	x	x	x	x	x	x	x

Figure 5.8 Cabin Message ID Format

Message 0,1,2 and 3 are used to report state-changes at cabin buttons. All these messages carry 1 byte of data. Message0 carries information about buttons 0-7, message1 buttons 8-15, message2 buttons16-23 and message3 buttons 24-31. For these bits logic 1 means that button is pressed and logic 0 means button is released. Cabin broadcasts these messages according number of stops in lift system.

5.4 Message Triggering

CP broadcasts message0 just after initialization. Message0 is also broadcasted if there is a change in related lift parameters.

CP messages of 1,2 and 3 are normally broadcasted in order 10-bit time later after bus is idle. If bus doesn’t become idle after 5 messages then master again broadcasts the messages. But if a cabin or floor unit message is received or there is a change in display or signal lamps information then related messages are produced immediately.

CP broadcasts message 4 and 5 as soon as receiving password-checking messages.

Floor Units and cabin broadcast their messages as soon as there is a change in state of buttons or there is a call request with password.

5.5 Network Management

If there is still a faulty node even if 5 consecutive recovering processes were applied then master decreases baudrate and starts new recovering process. Master applies this procedure 3 times. If the problem continues then no more recovering process is applied.

6. CONCLUSION AND FUTURE WORK

Due to the widespread use of the CAN protocol from automotive to any kind of industrial applications, several classes of higher layer protocols have been developed. In this study some open system solutions for industrial automation have been mentioned. It was not intended to provide a guideline for choosing the best-suited protocol, but primarily to provide a better functional understanding of higher layer protocols.

Caused by the immense distribution of CAN in the non-automotive automation several higher layer protocols and open systems approaches have been originated almost simultaneously.

With CAL, a widely accepted and approved application layer standard is available for use in any application, which has to fulfill specific requirements in a fixed configuration environment. Due to the compatibility with CANopen, CANopen modules may be used within CAL systems.

With DeviceNet and CANopen two CAN-based sophisticated open systems standards are available today. On a first glance both solutions provide about the same functionality, although the approaches, which are taken, are quite different. Most significantly this concerns the usage of message identifiers, which completely left open to the system designer or integrator in CANopen. Although DeviceNet provides a free usage of message identifiers, there are some limitations. DeviceNet is based on a connection-oriented view but CANopen is based on a message-oriented view. Each of the system uses different data transport protocols with DeviceNet providing the most variety.

We also defined a new application protocol CANUP. We think CANUP protocol is easier to understand and to implement for small applications rather than well-known protocols. And we described a lift communication application on CANUP.

However, it is also acceptable that CANUP protocol defined in thesis is just a preliminary study and there may be some imperfect parts and structures. Anyway, it is certain that CANUP must be developed according to the needs of different applications and device profiles. There are some undefined message IDs in system messages and these can be configured according to requirements. Protocol structure also supports extended ID (29-bit) and lots of new messages and structures can be added to the protocol.

There are also lots of future works in lift communication application ASCAN. In this study, we made general ID assignment for whole lift network but only the communication between control panel and cabin and floor units is described in details. The communications between different control panels, sensors and other smart units including definitions and structures left open for future work.

REFERENCES

[1] Perry Sink, 2003, A Comprehensive Guide to Industrial Networks, http://bgfx.com/comprehensive_guide_to_industria.htm

[2] Synergetic Micro Systems, 2000, Eight Popular Open Architecture Fieldbuses, Embedded Solutions Magazine.

[3] Etschberger, K., 2001,Controller Area Network Basics, Protocols, Chips and Applications, IXXAT Press, Weingarten.

[4] ISO-IS 11898, 1993, Road vehicles – Interchange of digital information – Controller Area Network (CAN) for high-speed communication

[5] Bosch, 1991, CAN specification, Version 2.0, Robert Bosch GmbH.

[6] NEC, 2003, UPD78F9852 8-Bit Single-Chip Microcontroller with CAN Interface Preliminary User's Manual

[7] CAN-in-Automation, 1996**,** CAN Application Layer for Industrial Applications CiA DS 201-207 Version 1.1.

[8] CAN-in-Automation, 2000**,** CANopen Communication Profile for Industrial Systems CiA-301 Version 4.01.

[9] CAN-in-Automation, 2000**,** CANopen Framework for Programmed CANopen Devices CiA-302 Version 3.0.

[10] ODVA, 1997**,** DeviceNet Specifications Release 2.0, Vol. I: Communication Model and Protocol, Vol. II: Device Profiles and Object Library

[11] Etschberger, K., 1997, CAN-based Higher Layer Protocols and Profiles, Proceedings of the 4th International CAN Conference, Berlin, Germany, October 1997.

[12] Etschberger, K., 1994, Modelling Distributed Application Processes with CAL, Proceedings of the 1st International CAN Conference

[13] Lennartsson, K., 1995, Fundemental Parts in SDS, DeviceNet and CAN-Kingdom A Comparison, Proceedings of the 2nd International CAN Conference,

CURRICULUM VITAE

Akın Özdemir was born in 1976 in İstanbul. He completed primary and secondary schools in Istanbul. He graduated from Kadir Has high school in 1993. He received his Bsc degree from İstanbul University Electronics Engineering Department in 1998. He started M.S. study in İstanbul Technical University in 1998.

He has been a research & development engineer in Aybey Elektronik Company since 1999.