CiA 201 to 207 Version 1.1

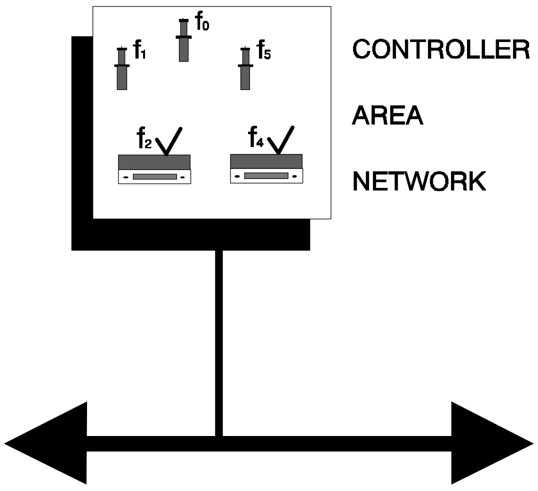
CAN Application Layer for industrial applications

Contents

History

Date	Document	Changes
Feb 96	**DS202-1**	- chapter 4 'Data Types' deleted; numbering of chapter 5 to 7 changed to 4 to 6;
		- definition of segmented remote services of domains changed; in remote result parameter 'success' becomes mandatory; parameter 'failure' deleted; service description for case of a failure;
		- paragraph 3.4 'CMS Data Types' changed
Feb 96	**DS202-3**	- document completely revised;
		- simplyfication of encoding rules definitions;
		new extended data types added
Feb 96	**DS203-2**	- node connect protocol: definition of parameter
		'Node-ID' changed; parameter 'a' (abortion flag) added; definition of error codes changed; description of cs 1 and 4 changed;
		- added paragraph 3.4 'Usage of Command Specifiers'
Feb 96	**DS204-1**	- service ´create user definition´: changed the definition of a ´free COB definition´
Feb 96	**DS205-1**	- service 'activate bit timing parameters':
	DS205-2	description of parameter 'switch-delay' changed;
		- service ´configure bit timing parameters´: value ´0´ for parameter table_selector now references the CiA standard bit timing parameters as defined in DS102;
		- added services and protocols to identifiy nodes with their LMT Address
Feb 96	**DS205-2**	- paragraph 5.6 renumbered to 5.5 (5.5 didn´t exist)
Feb 96	**DS206**	- new document
Feb 96	**DS207**	- NMT object name syntax: definition of 'NMT-Address' changed

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS201
February 1996

CAN in the OSI Reference Model

1. SCOPE

This document contains a description of the CAN Reference Model. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: ISO 7498: 1984, Information Processing Systems - Open Systems Interconnection - Basic Reference Model

/2/: ISO 11898: Road Vehicles, Interchange of digital information - Controller Area Network (CAN) for high-speed communication, November 1993

/3/: CiA/DS 102-1, CAN Physical Layer for Industrial Applications - Part 1: Two - Wire Differential Transmission

/4/: Robert Bosch GmbH, CAN Specification 2.0 Part B, September 1991

3. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

CAL:
CAN Application Layer. The application layer for CAN as specified by CiA.

CAN:
Controller Area Network. A network originally defined for use as a communication network for control applications in automobiles.

CMS:
CAN based Message Specification. One of the service elements of the application layer in the CAN Reference Model. CMS is a language that can describe how the functionality of a module can be accessed at its CAN interface.

COB:
Communication Object. A unit of transportation in a CAN Network. Data must be sent across a CAN Network inside a COB. There are 2032 different COB's in a CAN Network. A COB can contain at most 8 bytes of data. In /4/, the possibility of having more than 2032 COB's is described. The CAN Application Layer as specified by CiA can be extended in the future in a compatible way to include this possibility.

COB-ID:
Each COB is uniquely identified in a CAN Network by a number called the COB Identifier (COB-ID). The COB-ID determines the priority of that COB for the MAC sub-layer.

Remote COB:
A COB whose transmission can be requested by another module.

DBT:
COB-ID Distributor. One of the service elements of the application layer in the CAN Reference Model. It´s the responsability of the DBT to distribute COB-ID's to COB's that are used by the CMS service element.

LME:
Layer Management Entity. This entity serves to configure parameters for each of the layers of the CAN Reference Model.

LMT:
Layer Management. One of the service elements of the application layer in the CAN Reference Model. It serves to configure parameters of each of the layers in the CAN Reference Model via the CAN network.

LLC:
Logical Link Control. One of the sub-layers of the Datalink Layer in the CAN Reference Model that gives the user an interface that is independent from the underlying MAC layer.

MAC:
Medium Acces Control. One of the sub-layers of the Datalink Layer in the CAN Reference Model that controls who gets access to the medium to send a message.

MDI:
Medium Dependent Interface. One of the sub-layers of the Physical Layer in the CAN Reference Model that specifies the mechanical and electrical interface between the medium and a module.

NMT:
Network Management. One of the service elements of the application layer in the CAN Reference Model. The NMT serves to configure, initialize, and handle errors in a CAN network.

PLS:
Physical Layer Signalling. One of the sub-layers of the Physical Layer in the CAN Reference Model that specifies the bit representation, timing and synchronization.

PMA:
Physical Medium Attachment. One of the sub-layers of the Physical Layer in the CAN Reference Model that specifies the functional circuitry for bus line transmission/reception and may provide means for failure detection.

4. THE CAN REFERENCE MODEL

The *Controller Area Network (CAN)* is a data communication network designed to fit *distributed real-time control applications*. It was originally developed and applied by the automotive industry to solve the cabling problem inside vehicles. However CAN also provides good properties as a control network for industrial applications.

The purpose of the CAN Reference Model and its related service- and protocol specifications is to make CAN an *open network* where modules from different suppliers can cooperate in distributed applications.

4.1 Layered Architecture of CAN

The CAN Reference Model is a layered architecture to describe the functionality that CAN offers to an application and is based on the OSI Reference Model. A basic knowledge of the OSI Reference Model and its terminology is required to understand the CAN Reference Model (see /1/).

There exists an ISO Standard /2/ for CAN. This draft specifies the Physical and Data Link layer. The CAN Reference Model extends the MDI sublayer of the Physical Layer of /2/ to guarantuee interoperability on the medium. In addition to /2/, the CAN Reference Model contains an Application Layer and a Layer Management Entity (LME) to guarantuee interoperability between applications. The CAN Reference Model and its relation to the OSI Reference Model are shown in Fig. 1.

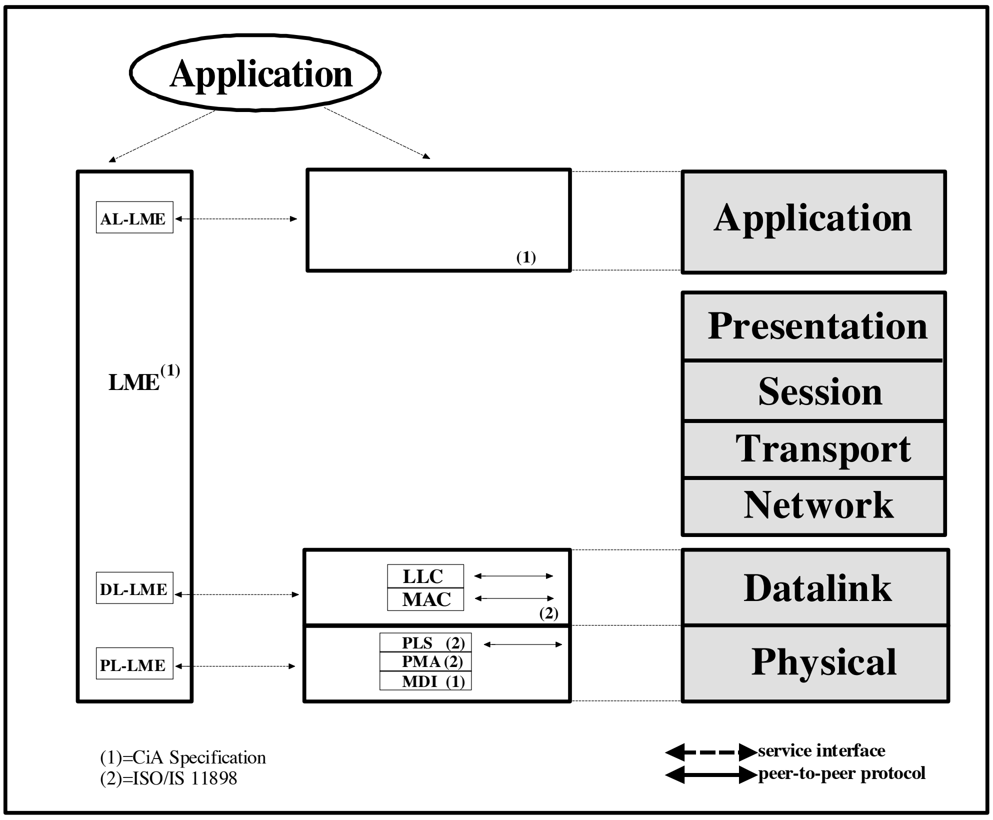


Fig. 1: The CAN Reference Model

The absence of the OSI layers 3-5 has the following reasons:

- NETWORK LAYER. There is no inter-networking or routing function in CAN: every COB reaches all modules on the bus.

- TRANSPORT LAYER. The purpose of the Transport Layer in the OSI Reference Model is to enable the upper layers to **reliably transfer messages of arbitrary length over unreliable networks** by offering functions as segmentation, sequencing, automatic retries and duplicate frame detection. For distributed real-time control applications however, each message transfer attempt stands on its own. This type of applications require **high speed transfer of short messages** and need to know immediately whether a message transfer attempt succeeded or failed to be able to respond in time. Since there is no Network Layer and the Datalink Layer of CAN is considered to be reliable enough, CAN applications do not need a Transport Layer to guarantuee a reliable message transfer service. The Application Layer provides services to enable those applications that need this, to send arbitrary length messages. This leaves no functionality for a Transport Layer.

- SESSION LAYER. In distributed real-time control applications the concept of sessions, synchronization points and roll-back mechanisms are usually not supported. However, CIA may specify an optional CAN session layer in future to support power reduction by a Standby Capability

- PRESENTATION LAYER. The presentation layer deals with the transfer of application data and its meaning via the network. In the CAN Reference Model all applications must use a *structure* consisting of *basic data types* to describe their data. This data is encoded to a *transfer syntax* and it is assumed that all applications know the meaning of the data a priori. This leaves no functionality for the Presentation Layer.

4.2 The Physical Layer

The Physical Layer and its sub-layers are defined in /2/ and /3/.

4.3 The Datalink Layer

The Datalink Layer and its sub-layers are defined in /2/.

4.4 The Application Layer

The application layer is the interface between the data communication environment and the application that uses that environment to cooperate with other applications. Together the cooperating applications form a *distributed application*.

Message Oriented vs. Node Oriented

The Datalink Layer of CAN only offers a broadcast service of identified messages (COB's). COB's are identified through a COB Identifier (COB-ID). Data is not sent to applications on specific nodes in the network (node-oriented network). Each application itself decides whether or not it will receive the data contained in a COB (message oriented).

In a CAN Network therefore, by definition the receiving applications are not known by the transmitter. This message oriented nature of CAN is preserved in the services of the Application Layer.

Application Layer Structure

The functionality that the application layer offers to an application is logically divided over different *service elements* in the application layer. A service element offers a specific functionality and all the related services (e.g File Transfer). These services are described in the *Service Specification* of that service element.

Distributed applications interact by invoking services of a service element in the application layer. To realize these services, this service element exchanges data via the CAN

Network with (a) peer service element(s) via a protocol. This protocol is described in the *Protocol Specification* of that service element.

Application Layer Service Primitives

Service primitives are the means by which the application and the application layer interact. There are four different primitives:

- a *request* is issued by the application to the application layer to request a service
- an *indication* is issued by the application layer to the application to report an internal event detected by the application layer or indicate that a service is requested
- a *response* is issued by the application to the application layer to respond to a previous received indication
- a *confirm* is issued by the application layer to the application to report on the result of a previously issued request.

Application Layer Service Types

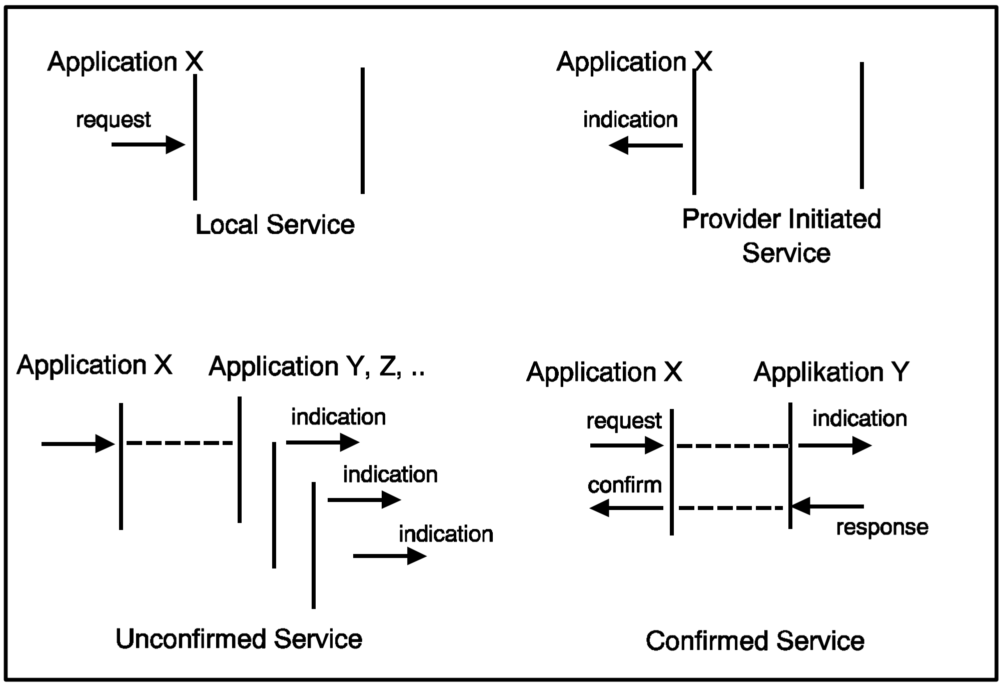


Fig. 2: Application Layer Service Types

A *service type* defines the primitives that are exchanged between the application layer and the cooperating applications for a particular service of an application layer service element. The service elements of the application layer of the CAN Reference Model offer the following service types (see Fig. 2):

- A *local service* involves only the local service element. The application issues a request to its local service element that executes the requested service without communicating with peer service elements.

- An *unconfirmed service* involves one or more peer service elements. The application issues a request to its local service element. This request is transferred to the peer service element(s) that each pass it to their application as an indication. The result is not confirmed back. Note that in CAN it is unknown by the transmitter which service elements will receive the request!

- A *confirmed service* can involve only one peer service element. The application issues a request to its local service element. This request is transferred to the peer service element that passes it to the other application as an indication. The other application issues a response that is transferred to the originating service element that passes it as a confirmation to the requesting application. Note that in CAN it is unknown by the transmitter which service element will receive the request!

- A *provider initiated service* involves only the local service element. The service element (being the service provider) detects an event not solicited by a requested service. This event is then indicated to the application.

Unconfirmed and confirmed services are collectively called*remote services.*

Application Layer Service Elements

The most important function of the application layer is to determine*what* an application can do with the communication environment. The CAN application layer provides four application layer service elements (see Fig. 3):

- CAN based Message Specification (CMS)
- Network Management (NMT)
- Distributor (DBT)
- Layer Management (LMT)

CMS offers an open, object oriented environment to design user applications. CMS offers Variable-, Event-, and Domain objects to design and specify how the functionality of a module can be accessed at its CAN interface. The Encoding Rules define how to encode and decode application data into the transfer syntax and vv.

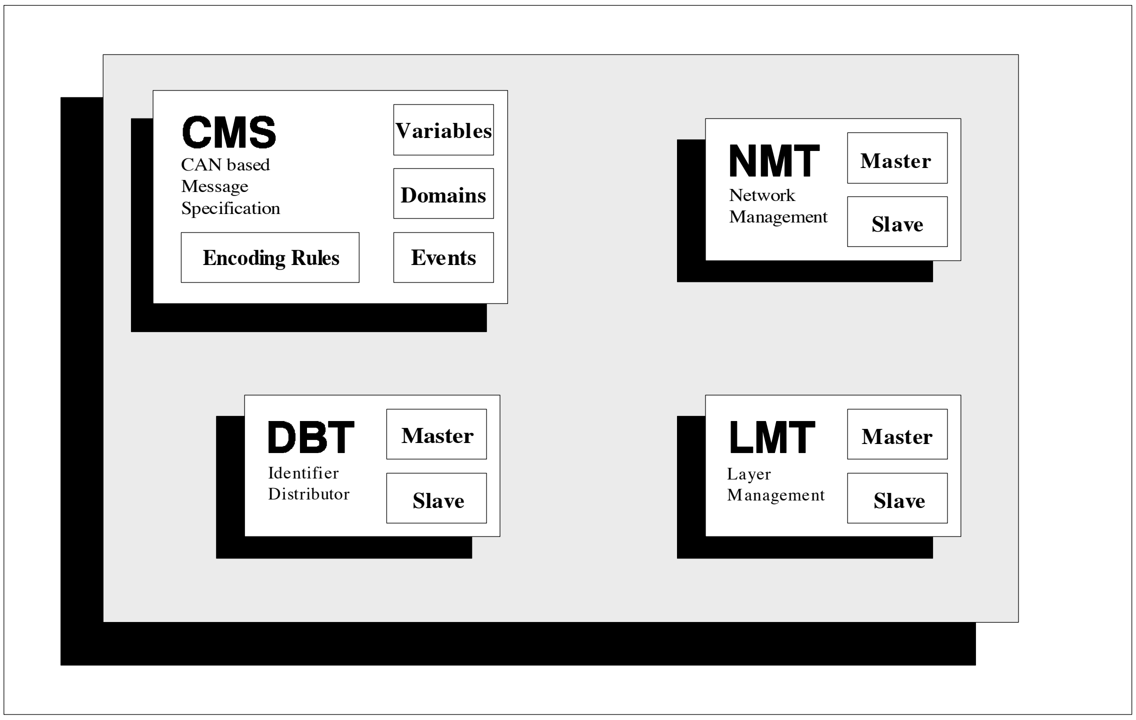


Fig. 3: The CAN Application Layer

NMT offers an open object oriented environment to let one module (the NMT Master) deal with the initialization and possible failures of the other modules (NMT Slaves).

The essential problem in defining an open CAN environment, is the *distribution of the COB Identifiers*. A COB Identifier determines the priority for the MAC protocol of that COB. Therefore, the value of the identifiers may not be fixed by the suppliers of the different CAN modules since the systems integrator wants to have system-wide control over the priorities of the COB's. The DBT offers a *dynamic distribution of the identifiers* by one module (the DBT Master) to the other modules (DBT Slaves).

LMT offers the possibility to let one module (the LMT Master) control the settings of certain layer parameters at another module (LMT Slave) via the CAN Network.

Application Layer Service Notation

This section defines the notation that is used in the service specifications of each service element of the application layer.

- **NOTE:** These notations do not suggest or restrict possible implementations of interfaces in either hardware or software. Hardware and software interface descriptions are product specific and do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

In the description of an application layer service all parameters and their attributes of the involved service primitives are specified. A parameter has the following attributes:

- **name**. This attribute symbolically indicates the purpose of the parameter.
- **usage**. This attribute specifies whether the parameter is *mandatory* (i.e the parameter must be present), *optional* (i.e the parameter may or may not be present), or *selection* (i.e one parameter must be selected from a list of parameters).

Indentation is used to indicate that a parameter is a sub-parameter of another parameter. The usage of a sub-parameter is always dependent on the usage of the parameter it appears under.

All this information is given in a tabular form. In the vertical direction the names of the parameters are listed. In the horizontal direction, the service primitives are listed. If a parameter is used in a certain service primitive, the value of the usage attribute is specified at the cross point in the table. The indentation of the parameters is preserved in the location of these values. All parameters that have the same level of indentation in the same column and whose usage is 'selection' form a list from which one must be selected.

Parameter	*Request/Indication*	*Response/Confirm*
Par 1	**Optional**	
subpar 1	selection	
subpar 2	selection	
Par 2		**Mandatory**
subpar 3		mandatory
subpar 4		optional

Table 1: Application Layer Service Notation

In the example of Table 1, Par1 is optional for the request and indication primitives. It has a list of sub-parameters: either subpar1 or subpar2 must be present if and only if Par1 is present. The parameters Par2 and subpar3 are mandatory for the response and confirm primitives. Subpar4 may be left out.

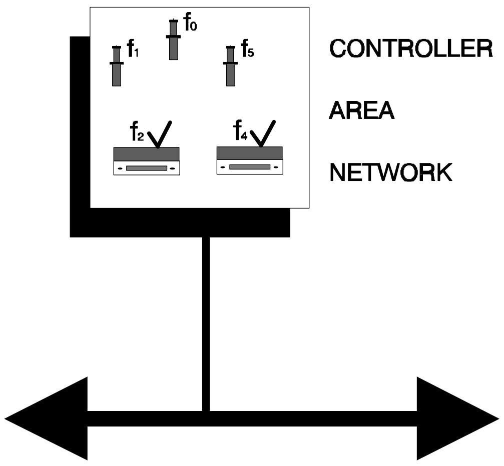
Naming Conventions

The service elements of the Application Layer use objects to model their functionality. Through the application layer services, an application can create instances of these objects. Each instance has a *name*. To syntax of these names must be according to the Application Layer Naming Conventions.

4.5 The Layer Management Entity

The Layer Management Entity (LME) is an entity that allows an application to control the settings of layer parameters. The values can originate from the application itself or from the application on another module through the LMT service element. Within the LME, each layer has its own specific layer management entity, see Fig. 1. Note that there is no LME protocol since all LME services are local.

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS202-1
February 1996

CMS Service Specification

1. SCOPE

This document contains the service specification of the CAN based Message Specification (CMS). CMS is part of the CAN Application Layer. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS202-2, CMS Protocol Specification

/3/: CiA/DS202-3, CMS Data Types and Encoding Rules

/4/: CiA/DS207, Application Layer Naming Conventions

3. GENERAL DESCRIPTION

3.1 CMS Perspective

CMS is one of the application layer service entities of the CAN Reference Model, see /1/.

3.2 CMS Objects and Services

CMS is a language to specify what COB's a module uses and how they are formatted. CMS can describe all CAN layer 2 features. This means also that existing applications can be described in CMS. Furthermore CMS offers the application a possibility to model its behaviour in the form of objects and remote services on these objects. This allows other applications to cooperate with it by executing these services that CMS supports on these objects. The different service types and primitives are defined in /1/. The notation that is used to describe the CMS Services is also explained in /1/.

3.3 CMS Clients and Servers

An example is given in fig. 1 where CMS is used to model the control of a light switch. The *server* of the switch "physically" interacts with the switch to put on the light. The server "translates" this behaviour into the CMS language e.g a CMS variable with access type 'Write_Only' and data type BOOLEAN. The *client* "logically" interacts with the switch by using the remote CMS 'Write Variable' service.

CMS allows for an object to have one or more servers and zero or more clients, depending on what the object models.

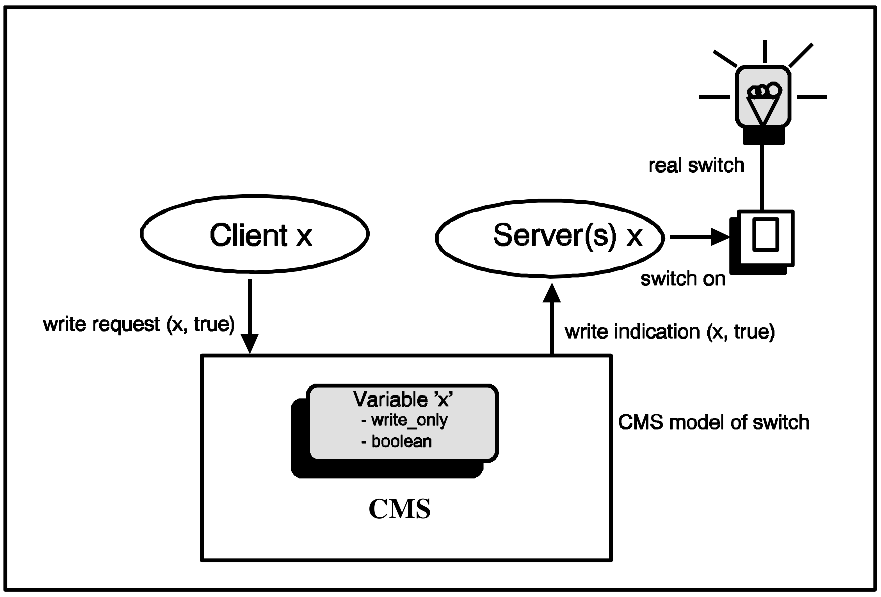


Fig. 1: A CMS Model for a light switch

3.4 CMS Data Types

In order to give a server sufficient information on what he has to do, the client may have to exchange data with the server, e.g the required and current position of the valve. CMS models this by the concept of data types. CMS defines a number of basic types such as INTEGER(5). It is also possible to construct a compound type by collecting basic types in an ARRAY or a STRUCTURE. CMS defines also a number of extended data types.

CMS defines a transfer syntax that describes how values of a particular data type have to be transmitted over the CAN network. The data types and transfer syntax is described in the CMS Data Types and Encoding Rules, see /3/.

3.5 CMS Object Priorities

The priority of a CMS object indicates its importance relative to other CMS objects and is used as an arbitration value by the Medium Access Control of CAN. CMS defines eight priorities in the range [0, 7]. 0 is the highest, 7 the lowest priority.

Priorities can be assigned by the application or the Distributor Service Element (see /1/). In case the Distributor Service Element assigns a priority, the Network Management Service Element (see /1/) controls when the assignment takes places.

3.6 CMS Object Inhibit Times

To guarantee that no starvation on the network occurs for CMS objects with low priorities, CMS objects can be assigned an inhibit time. The inhibit-time of a CMS object defines the minimum time that has to elapse between two consecutive invocations of a CMS remote service for that CMS object.

Inhibit-times can be assigned by the application or the Distributor Service Element (see /1/). In case the Distributor Service Element assigns an inhibit-time, the Network Management Service Element (see /1/) controls when the assignment takes places.

3.7 CMS Service Descriptions

The CMS services are described in a tabular form that contains the parameters of each service primitive that is defined for that service. The primitives that are defined for a particular service determine the service type (e.g unconfirmed, confirmed, etc.). How to interprete the tabular form and what service types exist is defined in /1/. In the service descriptions, [a, b] denotes the range of integers from a to b with a and b included. If a > b, the range is empty.

All services assume that no failures occur in the Data Link and Physical Layer of the CAN network. These failures are resolved by the Network Management Service Element, see /1/.

CMS executes a protocol to implement the services on a CMS object. All protocols are defined in /2/. Each protocol needs one or two COB's to be able to transmit and receive data over the CAN network. This document specifies for each CMS service the used COB's and their attributes:

- the COB-Class. There are 8 COB-Classes. A COB-Class relates the number of (remote) receivers and (remote) transmitters for that COB. The Distributor

COB Class	*#receivers*	*#transmitters*
0	0 .. 1	0 .. 1
1	1	0 .. 1
2	≥ 1	0 .. 1
3	≥ 0	0 .. 1
4	0 .. 1	1
5	1	1
6	≥ 1	1
7	≥ 0	1

Service Element checks for each COB whether the total number of (remote) transmitters and (remote) receivers matches the COB-Class.

- the COB-type for both the server- and client:

rx	= receives a COB
tx	= transmits a COB
RTR-rx	= receives the data of a remote COB
RTR-tx	= transmits the data of a remote COB

- the COB-length. If '*' is specified it means that the data length is determined by the CMS Encoding Rules (see /3/), the data- and error type attribute of the CMS object, and the CMS protocol that this COB is used for (see /2/). If a number is specified it means that the COB has a fixed length.

4. VARIABLES

4.1 Attributes

The following attributes are defined for variables:

- name:	see /4/
- user_type:	one of the values {Client, Server}
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n $\underline{>>}$ 0
- data_type:	see /3/
- error_type:	see /3/
- class:	one of the values {Basic, Multiplexed}
- access_type:	one of the values {Read_Only, Write_Only, Read_Write}

A variable whose class is 'Basic' is called a basic variable. A variable whose class is 'Multiplexed' is called a multiplexed variable. A multiplexed variable is multiplexed with other multiplexed variables into a variable set.

The following attributes are only defined for multiplexed variables:

- var_set_name:	see /4/
- mutiplexor:	a value in the range [0, 127]

The multiplexor is a natural number that uniquely identifies the variable within the variable set. Multiplexed variables inherit the values of the user_type, access_type, priority, and inhibit-time attributes from the corresponding attributes of the variable set. Hence, all multiplexed variables within one variable set have the same value for these attributes. The following attributes are defined for variable sets:

- name:	see /4/
- user_type:	one of the values {Client, Server}
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n $\underline{>>}$ 0
- access_type:	one of the values {Read_Only, Write_Only, Read_Write}

4.2 Usage

The access type of a variable is seen from the point of view of the client. Variables with access_type 'Read_Only' can be used by a client only to collect data. For basic variables the collected data will be the data that was set by the server in the last 'update variable' service it executed. Data from previous updates is lost. For multiplexed variables the server has to supply the data when requested by the client.

Variables with access_type 'Write_Only' can be used by a client to request one or more servers to execute a command. The client will not know the result of the command execution.

Variables with access_type 'Read_Write' can be used by a client to collect the 'current data' from the server (read service) or to request a server to execute a command and be informed about the result of the command execution (write service).

Variable sets can be used to "multiplex" several variables. All these multiplexed variables will then be mapped onto the COB's that are used by that variable set. This reduces the number of COB's. Within a variable set the variables are identified by a unique "multiplexor".

4.3 Local Services

Define Variable

This service creates a variable with the requested attributes. Var_set_name must have been defined as a variable set. The attributes must not cause the data length of the used COB's to overflow the maximum.

Parameter	*Request*
Argument	**Mandatory**
var_name	mandatory
data_type	mandatory
error_type	optional
class	mandatory
basic_var	selection
user_type	mandatory
acess_type	mandatory
priority	optional
inhibit-time	optional
mux_var	selection
var_set_name	mandatory
multiplexor	mandatory

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and the server of the variable.

Define Variable Set

This service creates a variable set with the requested attributes.

Parameter	*Request*
Argument	**Mandatory**
var_set_name	mandatory
user_type	mandatory
access_type	mandatory
priority	optional
inhibit-time	optional

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and the server of the variable set.

Update Variable

Through this service the server of var_name updates the value of var_name. Previously updated values for var_name are lost. Var_name must be a basic variable with access_type 'Read_Only' and user_type 'Server' and value must match the data_type attribute of var_name.

Parameter	*Request*
Argument	**Mandatory**
var_name	mandatory
value	mandatory

4.4 Remote Services

Write Variable

Through this service the client of var_name supplies a value to the server(s) of var_name. Var_name must be a variable with access_type 'Write_Only' or 'Read_Write'. The supplied value must match the data_type attribute of var_name.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
var_name	mandatory	
value	mandatory	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

- **Write_Only variables:** The service is unconfirmed. The supplied value is indicated to the server. There are no Response/Confirm primitives. There can be at most one client. There must be at least one server.

- **Read_Write variables:** The service is confirmed. The supplied value is indicated to the server. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally a value of the error_type attribute of var_name confirms the reason. There can be at most one client. There must be exactly one server.

Read Variable

Through this service the client of var_name requests the server of var_name to supply its value. Var_name must be a variable with access_type 'Read_Only' or 'Read_Write'. The supplied value must match the data_type attribute of var_name.

- **Read_Only basic variables:** The service is confirmed. The Remote Result parameter will confirm the requested value as set by the last Update Variable service. There can be zero or more clients. There must be exactly one server.

Parameter	*Request / Indication*	*Response / Confirm*
Argument var_name	**Mandatory** mandatory	
Remote Result value		**Mandatory** mandatory

- **Read_Write variables, Read_Only multiplexed variables:** The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of success, the requested value is confirmed. In case of a failure, optionally a value of the error_type attribute of var_name confirms the reason. There can be at most one client. There must be exactly one server.

Parameter	*Request / Indication*	*Response / Confirm*
Argument var_name	**Mandatory** mandatory	
Remote Result success value failure reason		**Mandatory** selection mandatory selection optional

4.5 Used COB's

- Read_Only Basic Variable

Client COB-Type	Server COB-Type	COB-Class	COB-Length
RTR-rx	RTR-tx	7	*

- Read_Only Multiplexed Variable

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx rx	rx tx	1 4	1 *

- Write_Only Variables

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	2	*

- Access_Type = Read_Write:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx rx	rx tx	1 4	* *

5. DOMAINS

5.1 Attributes

- name:	see /4/
- user_type:	one of the values {Client, Server}
- class:	one of the values {Basic, Multiplexed}
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n $\underline{>>}$ 0

For a domain there can be at most one client and there must be exactly one server. A domain whose class is 'Basic' is called a basic domain. A domain whose class is 'Multiplexed' is called a multiplexed domain. The following attribute is only defined for multiplexed domains:

- mux_data_type:	see section 4.2

5.2 Usage

Basic domains can be used to transfer an arbitrary large block of data from a client to a server and vv. The contents of a data block is application specific and does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

Multiplexed domains can be used to transfer multiple *data sets* (each containing an arbitrary large block of data) from a client to a server and vv. The client can control via a *multiplexor* which data set is to be transferred. This multiplexor is a value of a CMS Data Type. The CMS Data Type of the multiplexor and the contents of the data sets are application specific and do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

A domain is transferred as a sequence of *segments*. Prior to transferring the segments there is an initialization phase where client and server can prepare themselves for transferring the segments. For multiplexed domains, it is also possible to transfer a data set during the initialization phase. This mechanism is called an *expedited* transfer.

It is always the client that takes the initiative for a transfer. Both the client and the server can take the initiative to abort the transfer of a domain. By using the *segmented* services, the application will be responsible for the segmentation of the domain. By using the *non-segmented* services, CMS will be responsible for the segmentation.

5.3 Local Services

Define Domain

Parameter	*Request*
Argument	**Mandatory**
dom_name	mandatory
user_name	mandatory
priority	optional
inhibit-time	optional
class	mandatory
basic_dom	selection
mux_dom	selection
mux_data_type	mandatory

This service creates a domain with the requested attributes.

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and the server of the domain.

5.4 Remote Services (non-segmented)

When using these services, CMS will be responsible for transferring the domain as a sequence of segments.

Domain Download

Through this service the client of dom_name downloads data to the server of dom_name. The data and optionally its size are indicated to the server. For multiplexed domains the multiplexor of the data set that has been downloaded is indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name.

The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason is confirmed.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
data	mandatory	
size	optional	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

Domain Upload

Through this service the client of dom_name uploads data from the server of dom_name. For multiplexed domains the multiplexor of the data set that has to be uploaded is indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
Remote Result		**Mandatory**
success		selection
data		mandatory
size		optional
failure		selection
reason		optional

The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason is confirmed. In case of success, the data and optionally its size are confirmed.

5.5 Remote Services (segmented)

When using these services, the application will be responsible for transferring the domain as a sequence of segments.

Initiate Domain Download

Through this service the client of dom_name requests the server of dom_name to prepare for downloading data to the server. Optionally the size of the data to be downloaded is indicated to the server.

For multiplexed domains the multiplexor of the data set whose download is initiated and the transfer_type are indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name. In case of an expedited download, the data of the data set identified by multiplexor is indicated to the server.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
size	optional	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
transfer_type	mandatory	
normal	selection	
expedited	selection	
data	mandatory	
Remote Result		**Mandatory**
success		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of a successful expedited download of a multiplexed domain, this service concludes the download of the data set identified by multiplexor.

Download Domain Segment

Through this service the client of dom_name supplies the data of the next segment to the server of dom_name. The segment data and optionally its size are indicated to the server. The continue parameter indicates the server whether there are still more segements to be downloaded or that this was the last segment to be downloaded.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
data	mandatory	
size	optional	
continue	mandatory	
more	selection	
last	selection	
Remote Result		**Mandatory**
success		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of success, the server has accepted the segment data and is ready to accept the next segment. There can be atmost one Download Domain Segment service outstanding for a Domain.

For basic domains a successful 'Initiate Domain Download' service must have been executed prior to this service. For multiplexed domains a successful 'Initiate Domain Download' service with transfer_type 'normal' must have been executed prior to this service.

Initiate Domain Upload

Through this service the client of dom_name requests the server of dom_name to prepare for uploading data to the client. For multiplexed domains the multiplexor of the data set whose upload is initiated is indicated to the server. The value of multiplexor must match the mux_data_type attribute of dom_name.

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of success, optionally the size of the data to be uploaded is confirmed. In case of successful expedited upload of a multiplexed domain, this service concludes the upload of the data set identified by multiplexor and the corresponding data is confirmed.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
basic_dom	selection	
mux_dom	selection	
multiplexor	mandatory	
Remote Result		**Mandatory**
success		mandatory
size		optional
basic_dom		selection
mux_dom		selection
multiplexor		mandatory
transfer_type		mandatory
normal		selection
expedited		selection
data		mandatory

Upload Domain Segment

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
dom_name	mandatory	
Remote Result		**Mandatory**
success		mandatory
data		mandatory
size		optional
continue		mandatory
more		selection
last		selection

Through this service the client of dom_name requests the server of dom_name to supply the data of the next segment. The continue parameter indicates the client whether there are still more segements to be uploaded or that this was the last segment to be uploaded. There can be atmost one Upload Domain Segment service outstanding for a Domain.

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a failure, an abort domain transfer request must be executed. In case of success, the segment data and optionally its size are confirmed.

For basic domains a successful 'Initiate Domain Upload' service must have been executed prior to this service. For multiplexed domains a successful 'Initiate Domain Upload' service with transfer_type 'normal' must have been executed prior to this service.

Abort Domain Transfer

Parameter	*Request / Indication*
Argument	**Mandatory**
dom_name	mandatory
reason	optional

This service aborts the up- or download of dom_name. Optionally the reason is indicated. The service is unconfirmed. The service may be executed at any time by both the client and the server of dom_name. If the client of dom_name has a confirmed service outstanding, the Abort Indication is taken to be the Confirm of that service.

5.6 Used COB's

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	1	8
rx	tx	4	8

6. EVENTS

6.1 Attributes

- name:	see /4/
- user_type:	one of the values {Client, Server}
- class:	one of the values {Controlled, Uncontrolled, Stored}
- data_type:	see /3/
- priority:	a value in the range [0, 7]
- inhibit-time:	n*100 usec, n $\underline{>>}$ 0

An event whose class is 'Controlled' is called a controlled event. An event whose class is 'Uncontrolled' is called an uncontrolled event. An event whose class is 'Stored' is called a stored event. The following attributes are only defined for controlled events:

- error_type:	see /3/
- control_state:	one of the values {Enabled, Disabled}

6.2 Usage

An event can be used to model asynchronous behaviour such as a temperature exceeding a certain limit. The occurrence of an event is detected by the server and can be *notified* to the client(s). An event has a data_type attribute to supply additional information about what caused the event to occur such as the actual temperature that exceeded the limit.

Uncontrolled events can be used to implement events that are notified to any client that is "interested". Uncontrolled events are always notified when they occur.

Controlled events can be used to implement an event that can be notified to at most one client. The client can control whether the server notifies the event when it occurs.

Stored events can be used by a server to *store* locally a value of the data_type attribute of an event and optionally notify the client(s). A client, on his own initiative, can*read* the last value of an event that was stored by the server. Previously stored values are lost.

6.3 Local Services

Define Event

Parameter	*Request*
Argument	**Mandatory**
event_name	mandatory
data_type	mandatory
class	mandatory
controlled	selection
error_type	optional
uncontrolled	selection
stored	selection
user_type	mandatory
proiority	optional
inhibit-time	optional

This service creates an event with the requested attributes. The control state of a controlled event will be 'Disabled'. The attributes must not cause the data length of the used COB's to overflow the maximum.

- **NOTE:** The cooperating applications are responsible for using consistent attributes for the client and server of the event.

6.4 Remote Services

- **Event Class = Controlled:** There can be at most one client. There must be exactly one server.

- **Event Class = Uncontrolled:** There can be zero or more clients. There can be at most one server.

- **Event Class = Stored:** There can be zero or more clients. There must be exactly one server.

Notify Event

Through this service the server of event_name notifies the client(s) of event_name that the event has occurred and supplies its value. Event_name must be an uncontrolled event or a controlled event with control_state 'Enabled'. Value must match the data_type attribute of event_name. The service is unconfirmed.

Parameter	*Request / Indication*
Argument	**Mandatory**
event_name	mandatory
value	mandatory

Store Event

Through this service the server of event_name stores the value of event_name. Previously stored values for event_name are lost. Optionally the server immediately notifies this value to the client(s) of event_name.

Parameter	*Request / Indication*
Argument	**Mandatory**
event_name	mandatory
value	mandatory
immediate_notify	optional

Event_name must be a stored event. Value must match the data_type attribute of event_name. The service is local unless immediate notification is requested. In that case the service is unconfirmed.

Read Event

Through this service the client of event_name requests the server of event_name to supply the value as stored by the last Store Event service. The service is confirmed. The Remote Result parameter will confirm the value.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
event_name	mandatory	
Remote Result		**Mandatory**
value		mandatory

Event_name must be a stored event. Value must match the data_type attribute of event_name.

Set Event Control State

Through this service the client of event_name requests the server of event_name to set its control_state to the selected value. Event_name must be a controlled event. The service is confirmed. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally a value of the error_type attribute of event_name confirms the reason.

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
event_name	mandatory	
control_state	mandatory	
enabled	selection	
disabled	selection	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

6.5 Used COB's

- Uncontrolled Events:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
rx	tx	3	*

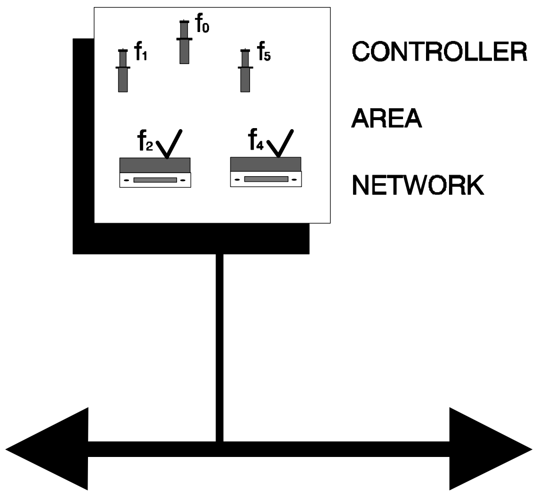
- Controlled Events:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
tx	rx	1	1
rx	tx	3	*

- Stored Events:

Client COB-Type	Server COB-Type	COB-Class	COB-Length
RTR-rx	RTR-tx	7	*

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS202-2
February 1996

CMS Protocol Specification

1. SCOPE

This document contains the protocol specification of the CAN-based Message Specification (CMS). CMS is part of the CAN Application Layer. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS202-1, CMS Service Specification

/3/: CiA/DS202-3, CMS Encoding Rules and Data Types

3. GENERAL DESCRIPTION

3.1 CMS Protocol Perspective

CMS defines a number of objects and remote services on these objects. In order to implement the remote services two (or more) CMS entities have to exchange information using a protocol.

3.2 CMS Protocol Descriptions

A protocol description describes the sequence of COB's and their format that are exchanged between the CMS Client(s) and Server(s) for a particular service on a CMS object, see /2/. All other COB attributes are described in /2/.

All CMS objects except a Multiplexed Variable use different COB's to implement the protocols for the services that are defined for that CMS object. Multiplexed Variables use the COB's of the Variable Set onto which they are multiplexed. There are two CMS service types, see /1/. Confirmed services use two COB's or one remote COB, whereas for unconfirmed services one COB will be sufficient.

The data length of the used COB's (indicated by the letter 'L' in the protocol descriptions) depends on the format of the application data (if any) that has to be transported in them. This format is specified by the data type or error type attribute of the corresponding CMS object (see /2/) and the CMS Encoding Rules (see /3/). These rules determine the number of bytes that are required to hold the application data.

The data length of the used COB's is the maximum as required by the format of the data type and error type attribute of the corresponding CMS object. For a Variable Set, the length of the used COB's is the maximum of the lengths required by each of the Multiplexed Variables that are multiplexed onto it. The length of the COB's for Domains (basic and multiplexed) is always 8.

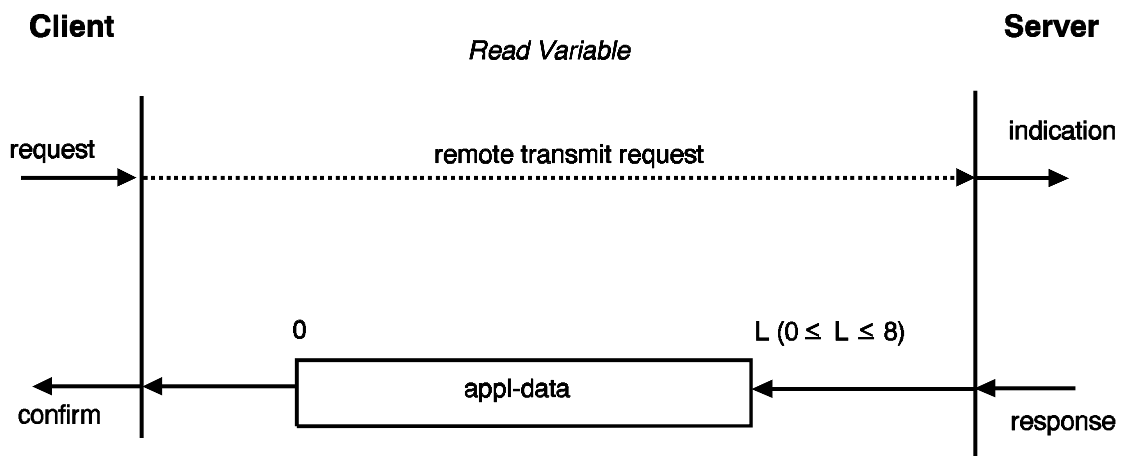
In the description of the COB data format, bytes are numbered from 0 to and including 7. Bits within a byte are numbered from 0 to and including 7. Byte 0 is transmitted first, byte 7 is transmitted last. Within a byte, bit 0 is the least significant bit, bit 7 is the most significant bit. In the protocol descriptions, [a, b] denotes the range of integers from a to b with a and b included. If $a > b$, the range is empty.

The terms 'lsb' and 'msb' stand for 'least significant byte' and 'most significant byte' respectively and are used to define how an integer number is stored in more than one byte. The order of significance is from lsb to msb.

4. CMS VARIABLE PROTOCOLS

4.1 Read-Only Access, Basic Variables

One confirmed service (Read Variable) is defined.



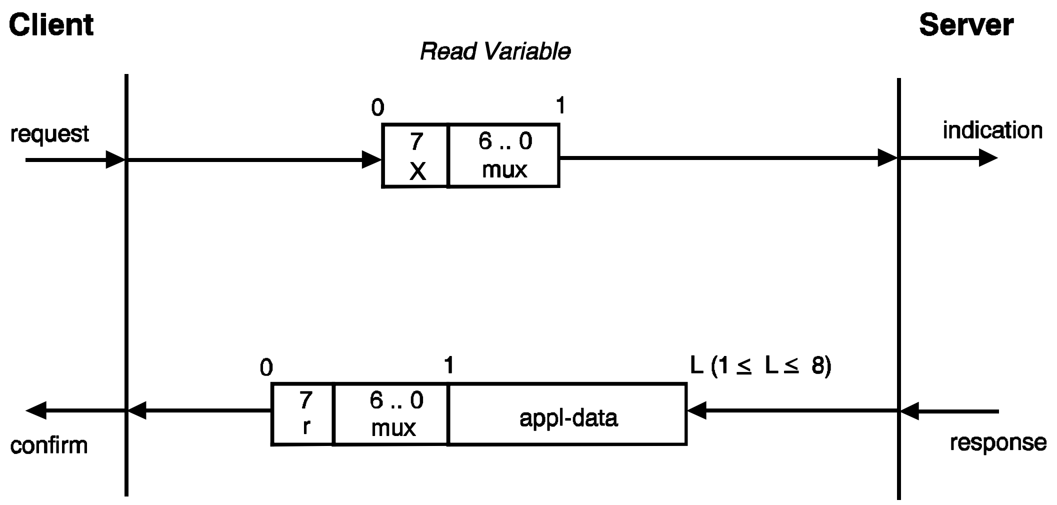
Read Variable Protocol

- **appl-data:** up to L bytes of application data representing a value of the data type attribute of the basic variable

4.2 Read-Only Access, Multiplexed Variables

One confirmed service (Read Variable) is defined.

Read Variable Protocol



- **mux:** multiplexor, a value between 0 and 127 (inclusive)
- **X:** not used, always 0
- **r:** result

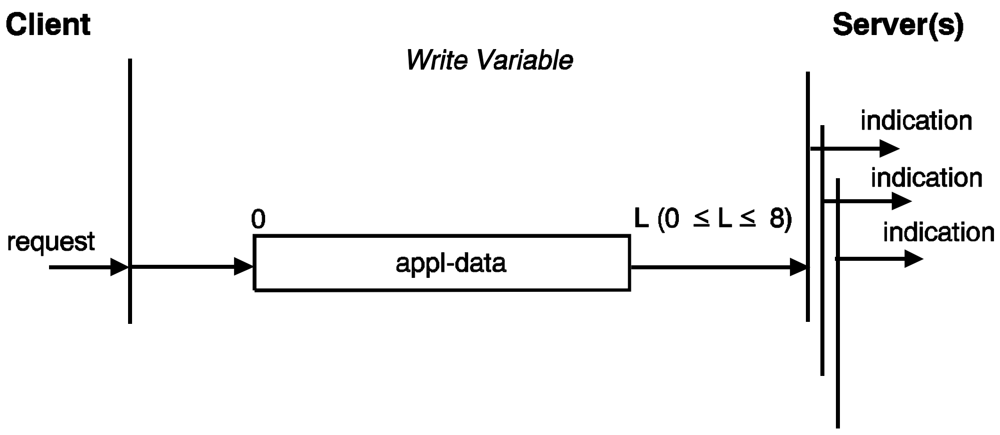
0:	Success
1:	Failure

- **appl-data:** up to L-1 bytes of application data. In case r = 0, it represents a value of the data type attribute of the multiplexed variable identified by mux. In case r = 1, it represents a value of the error type attribute of the multiplexed variable identified by mux.

4.3 Write-Only Access, Basic Variables

One unconfirmed service (Write Variable) is defined.

Write Variable Protocol



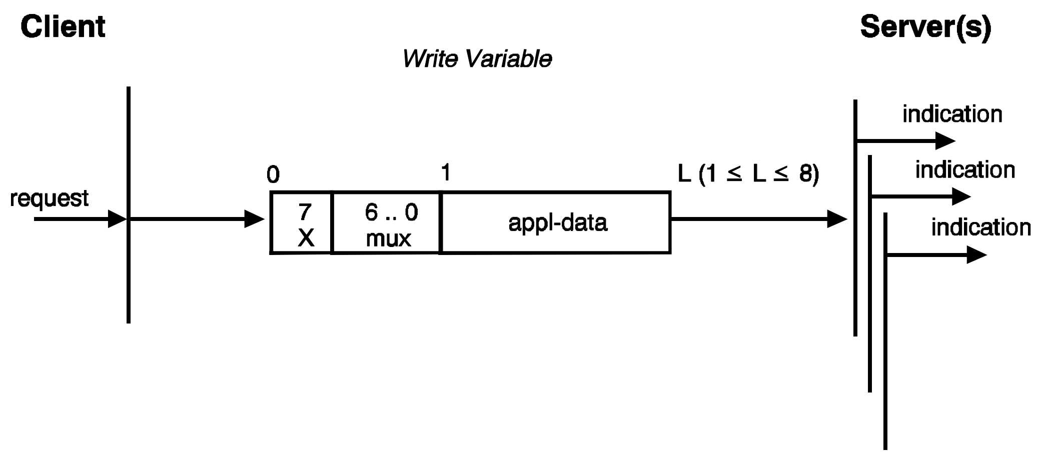
- **appl-data:** up to L bytes of application data representing a value of the data type attribute of the basic variable.

4.4 Write-Only Acces, Multiplexed Variables

One unconfirmed service (Write Variable) is defined.

Write Variable Protocol

- **mux:** multiplexor, a value between 0 and 127 (inclusive)

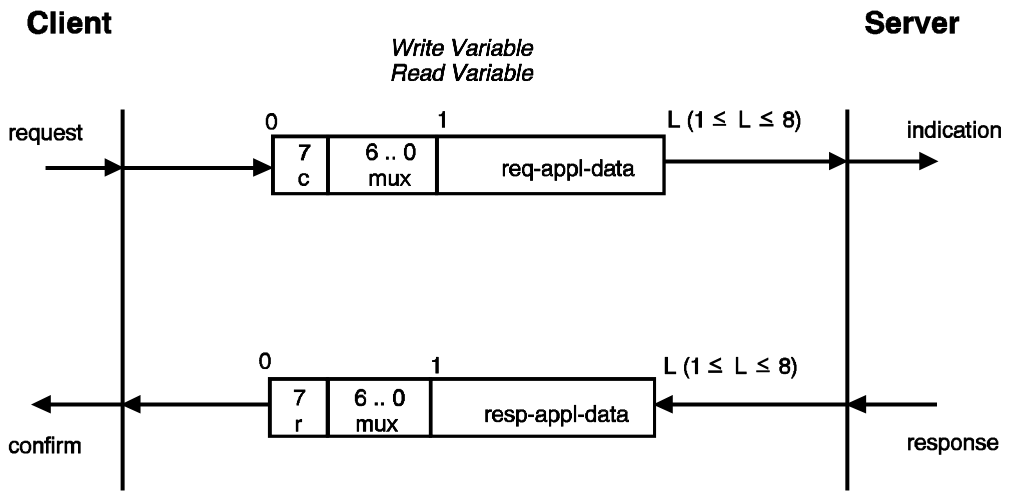


- **X:** not used, always 0
- **appl-data:** up to L-1 bytes of application data representing a value of the data type attribute of the multiplexed variable identified by mux.

4.5 Read/Write Access, Basic and Multiplexed Variables

Two confirmed services (Read Variable and Write Variable) are defined.

Read/Write Variable Protocol



- **mux:** multiplexor, only valid for multiplexed variables. If valid, a value between 0 and 127 (inclusive), otherwise 0
- **c:** command specifier

0:	write
1:	read

- **req-appl-data:** only valid when c = 0, otherwise reserved for further use by CiA. If valid it contains up to L-1 bytes of application data representing a value of the data type attribute of the basic variable, respectively the multiplexed variable identified by mux.

- **r:** result

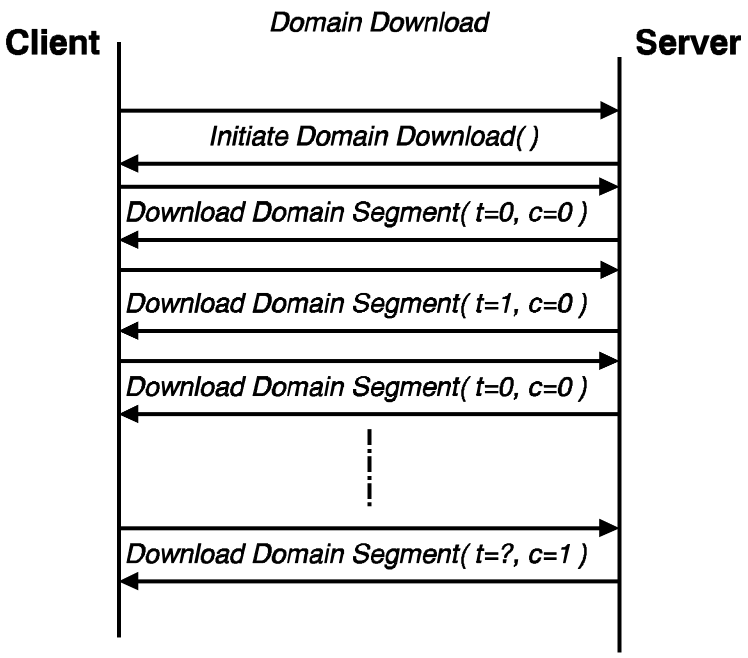
0:	Success
1:	Failure

- **resp-appl-data:** up to L-1 bytes of application data. In case of a write response and r = 0, it represents the same value as passed with the write request. In case of a read response and r = 0, it represents a value of the data type attribute of the variable respectively the multiplexed variable identified by mux. In case r = 1, it represents a value of the error type attribute of the basic variable respectively the multiplexed variable identified by mux.

5. CMS BASIC DOMAIN PROTOCOLS

Six confirmed services (Domain Download, Domain Upload, Initiate Domain Upload, Initiate Domain Download, Download Segment, and Upload Segment) and one unconfirmed service (Abort Domain Transfer) are defined for basic domains.

5.1 Download Domain Protocol

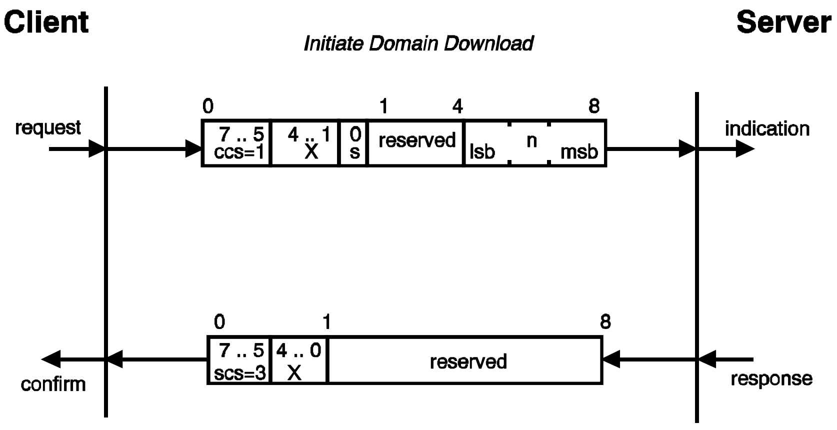

Client
Domain Download
Server
Initiate Domain Download( )
Download Domain Segment(t=0, c=0)
Download Domain Segment(t=1, c=0)
Download Domain Segment(t=0, c=0)
Download Domain Segment(t=?, c=1)

This protocol is used to implement the 'Domain Download' service for basic domains, see /2/. Basic domains are downloaded as a sequence of 'Download Domain Segment' services preceded by an 'Initiate Domain Download' service. The sequence is terminated by:

- a 'Download Domain Segment' response/confirm with the c-bit set to 1, indicating the succesful completion of the download sequence.
- an 'Abort Domain Transfer' request/indication, indicating the unsuccessful completion of the download sequence.
- a new 'Initiate Domain Download' request/indication, indicating the unsuccessful completion of the download sequence and the start of a new download sequence.

If in the download of two consecutive segments the toggle bit does not alter, this must be treated as if an invalid COB was received (see Annex I). If such an error is reported to the application, the application may decide to abort the download.

Initiate Domain Download Protocol



This protocol is used to implement the 'Initiate Domain Download' service for basic domains, see /2/.

- **ccs:** client command specifier
  1: initiate download request

- **scs:** server command specifier
  3: initiate download response

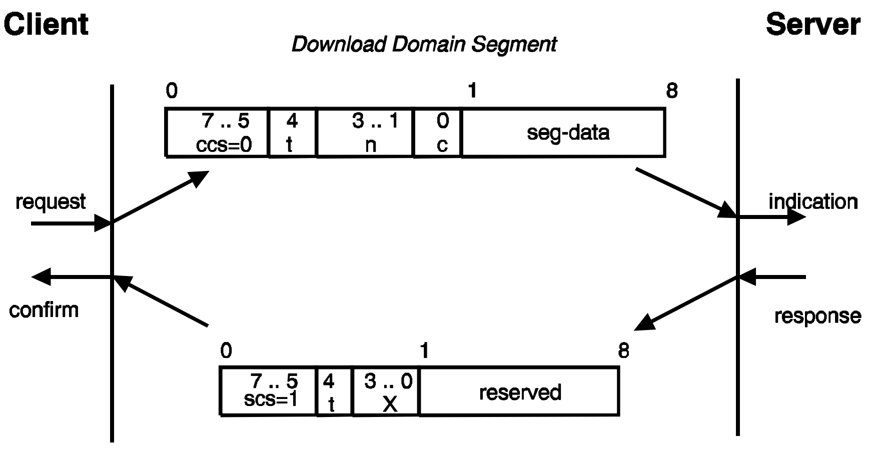
- **s:** size indicator
  0: data size is not indicated
  1: data size is indicated

- **n:** Only valid if s = 1, otherwise reserved for further use by CiA. If valid it contains the number of bytes to be downloaded

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA

Download Domain Segment Protocol



This protocol is used to implement the 'Download Domain Segment' service for basic domains, see /2/.

- **ccs:** client command specifier
  0: download segment request

- **scs:** server command specifier
  1: download segment response

- **seg-data:** at most seven bytes of segment data to be downloaded. Their meaning has to be specified by the application.

- **n:** indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.

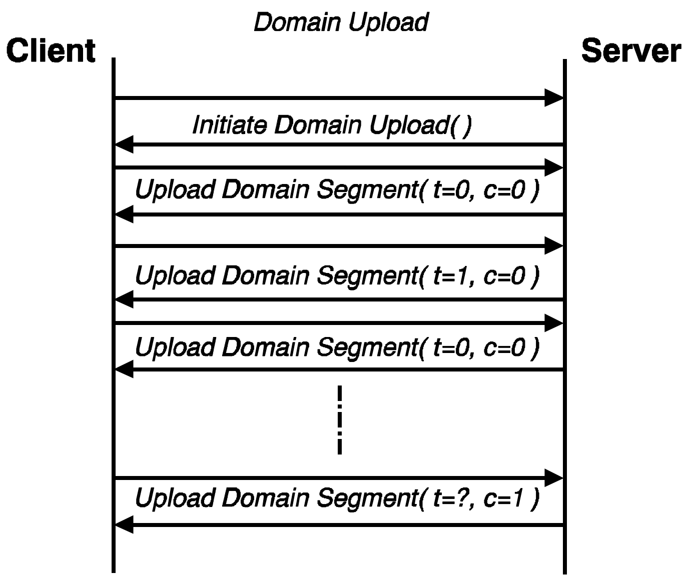
- **c:** indicates whether there are still more segments to be downloaded.
  0 more segments to be downloaded
  1: no more segments to be downloaded

- **t:** toggle bit. This bit must alternate for each subsequent segment that is downloaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA

5.2 Upload Domain Protocol


Domain Upload
Client
Server
Initiate Domain Upload( )
Upload Domain Segment(t=0, c=0)
Upload Domain Segment(t=1, c=0)
Upload Domain Segment(t=0, c=0)
Upload Domain Segment(t=?, c=1)

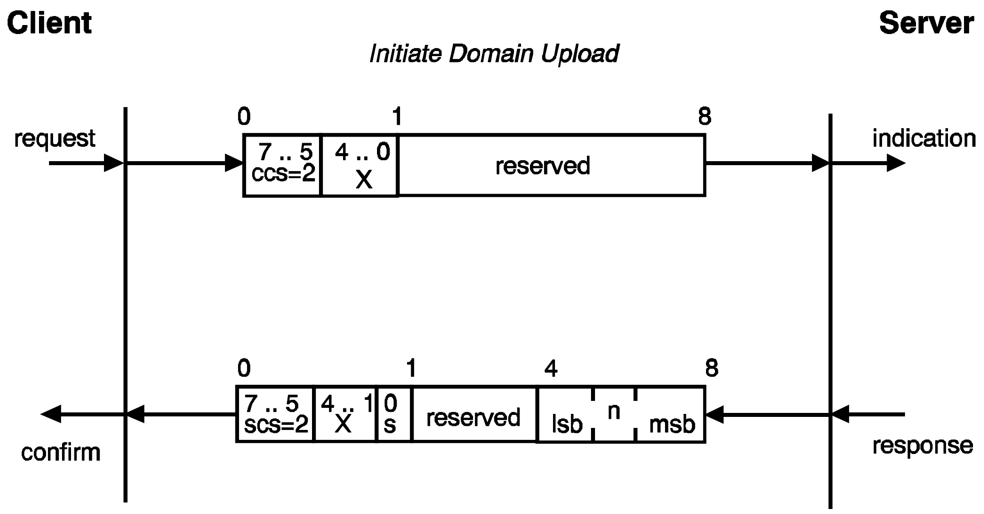
This protocol is used to implement the 'Domain Upload' service for basic domains, see /2/. Basic domains are uploaded as a sequence of 'Upload Domain Segment' services preceded by an 'Initiate Domain Upload' service. The sequence is terminated by:

- an 'Upload Domain Segment' response/confirm with the c-bit set to 1, indicating the succesful completion of the upload sequence.
- an 'Abort Domain Transfer' request/indication, indicating the unsuccessful completion of the upload sequence.
- a new 'Initiate Domain Upload' request/indication, indicating the unsuccessful completion of the upload sequence and the start of a new sequence.

If in the upload of two consecutive segments a toggle bit error occurs, this must be treated as if an invalid COB was received (see Annex I). If such an error is reported to the application, the application may decide to abort the upload.

Initiate Domain Upload Protocol

This protocol is used to implement the 'Initiate Domain Upload' service for basic domains, see /2/.



- **ccs:** client command specifier
  2: initiate upload request
- **scs:** server command specifier
  2: initiate upload response
- **s:** size indicator
  0: data size is not indicated
  1: data size is indicated

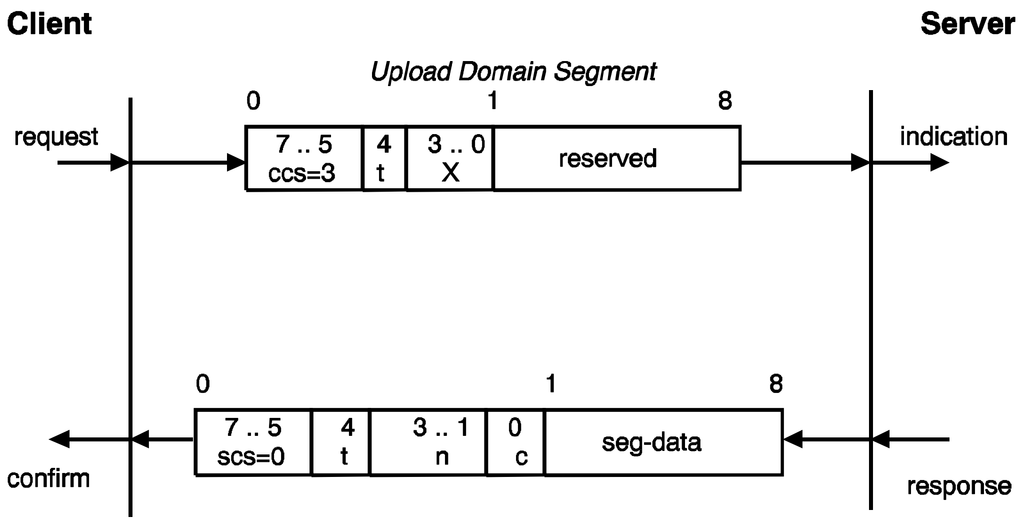
- **n:** Only valid if s = 1, otherwise reserved for further use by CiA. If valid it contains the number of bytes to be uploaded.

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA

Upload Domain Segment Protocol

This protocol is used to implement the 'Upload Domain Segment' service for basic domains, see /2/.



- **ccs:** client command specifier
  3: upload segment request

- **scs:** server command specifier
  0: upload segment response

- **t:** toggle bit. This bit must alternate for each subsequent segment that is uploaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

- **c:** indicates whether there are still more segments to be uploaded.
  0: more segments to be uploaded
  1: no more segments to be uploaded

- **seg-data:** at most seven bytes of segment data to be uploaded. Their meaning has to be specified by the application.

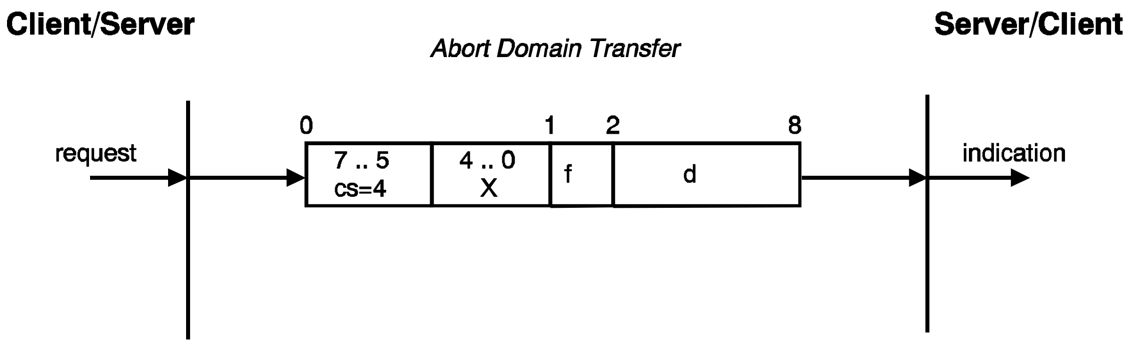
- **n:** indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.

- **X:** not used, always 0
- **reserved:** reserved for further use by CiA

5.3 Abort Domain Transfer

Abort Domain Transfer Protocol

This protocol is used to implement the 'Abort Domain Transfer' Service for basic domains, see /2/.



- **cs:** command specifier
  4: abort domain transfer

- **f:** indicates the reason for the failure.

0:	unspecified error
1:	application request
2:	no resources
3..127:	reserved for further use by CiA
128..255:	implementation specific error codes

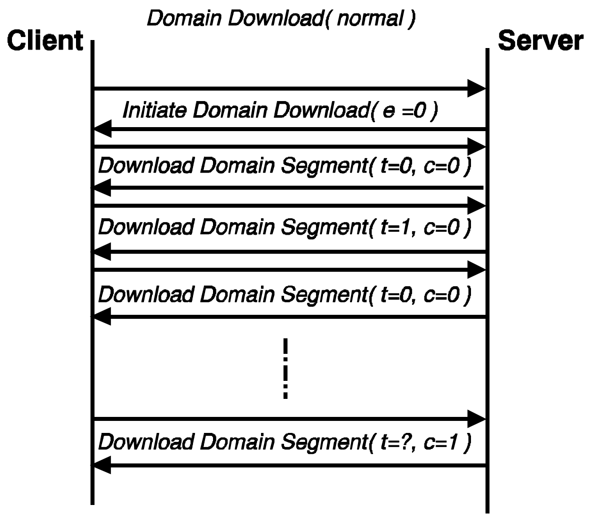
- **X:** not used, always 0
- **d:** only valid if f = 1 or f $\geq$ 128, otherwise reserved for further use by CiA. If valid it contains application specific information about the reason for the abort.

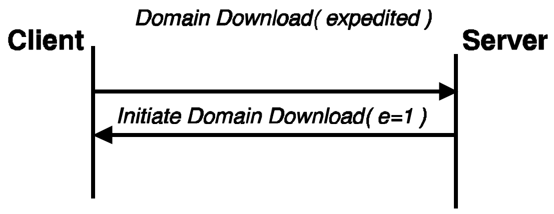
6. CMS MULTIPLEXED DOMAIN PROTOCOLS

Six confirmed services (Domain Download, Domain Upload, Initiate Domain Upload, Initiate Domain Download, Download Segment, and Upload Segment) and one unconfirmed service (Abort Domain Transfer) are defined for multiplexed domains.

The format of the multiplexor field in the COB's of Multiplexed Domains is determined by the multiplexor data type attribute of the corresponding Multiplexed Domain (see /2/) and the CMS Encoding Rules (see /3/). The multiplexor field has a fixed length of 3 bytes. If the encoded value of the multiplexor uses n bytes ($0 \leq n \leq 3$), it is located in bytes [1, n]. Bytes [1+n, 3] are not to be interpreted.

6.1 Download Domain Protocol


Domain Download(normal)
Client
Server
Initiate Domain Download(e =0)
Download Domain Segment(t=0, c=0)
Download Domain Segment(t=1, c=0)
Download Domain Segment(t=0, c=0)
Download Domain Segment(t=?, c=1)



This protocol is used to implement the 'Domain Download' service for multiplexed domains, see /2/. Multiplexed domains are downloaded as a sequence of zero or more 'Download Domain Segment' services preceded by an 'Initiate Domain Download' service. The sequence is terminated by:

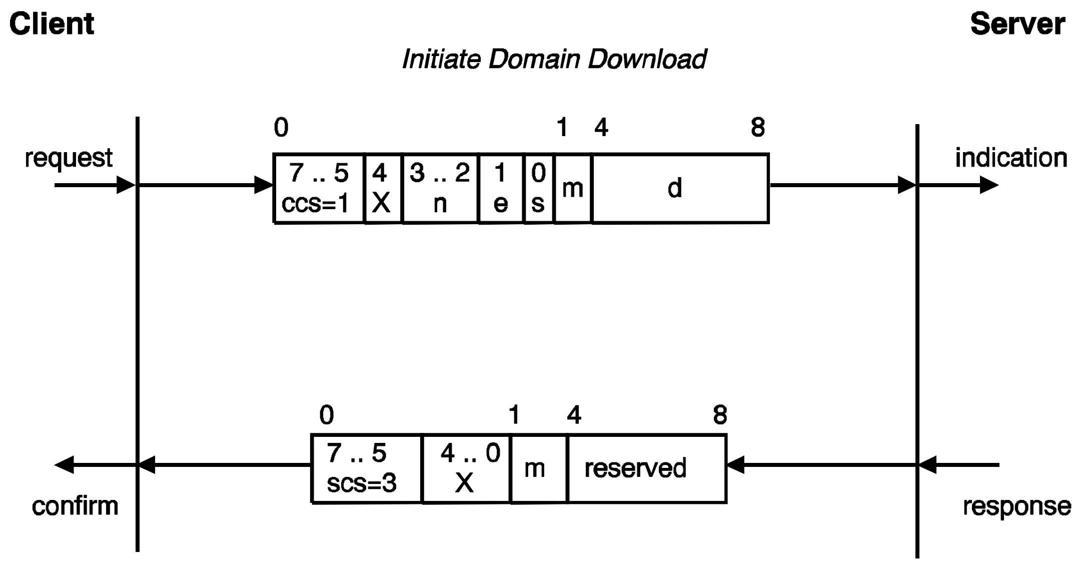
- an 'Initiate Domain Download' request/indication with the e-bit set to 1 followed by an 'Initiate Domain Download' response/confirm, indicating the successful completion of an expedited download sequence.
- a 'Download Domain Segment' response/confirm with the c-bit set to 1, indicating the succesful completion of a normal download sequence.
- an 'Abort Domain Transfer' request/indication, indicating the unsuccessful completion of the download sequence.

- a new 'Initiate Domain Download' request/indication, indicating the unsuccessful completion of the download sequence and the start of a new download sequence.

If in the download of two consecutive segments the toggle bit does not alter, this must be treated as if an invalid COB was received (see Annex I). If such an error is reported to the application, the application may decide to abort the download.

Initiate Domain Download Protocol

This protocol is used to implement the 'Initiate Domain Download' service for multiplexed domains, see /2/.



- **ccs:** client command specifier
  1: initiate download request

- **scs:** server command specifier
  3: initiate download response

- **n:** Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in d that do not contain data. Bytes [8-n, 7] do not contain data.

- **e**: transfer type
  0: normal transfer
  1: expedited transfer

- **s:** size indicator
  0: data set size is not indicated
  1: data set size is indicated

- **m**: multiplexor. It represents a value of the multiplexor data type attribute of the multiplexed domain.

- **d:** data

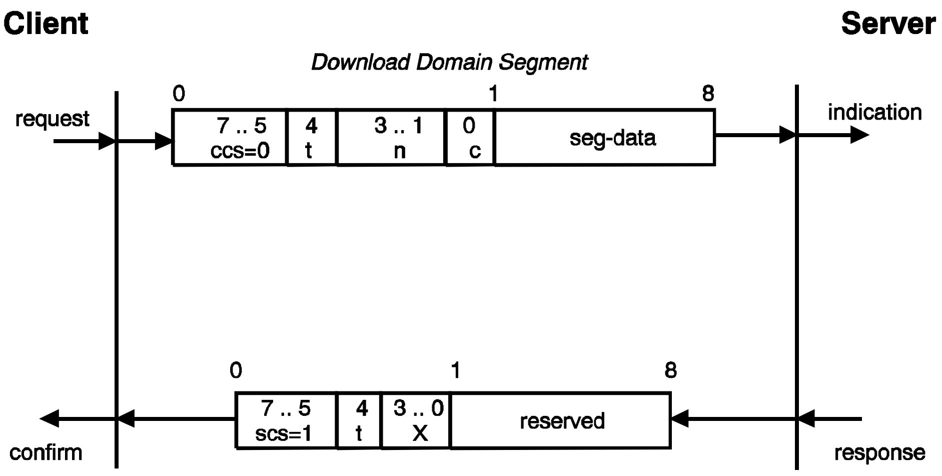
e = 0, s = 0:	d is reserved for further use by CiA.
e = 0, s = 1:	d contains the number of bytes to be downloaded. Byte 4 contains the lsb and byte 7 contains the msb.
e = 1:	d contains the data to be downloaded

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA

Download Domain Segment Protocol

This protocol is used to implement the 'Download Domain Segment' service for multiplexed domains, see /2/.



- **ccs:** client command specifier
  0: download segment request

- **scs:** server command specifier
  1: download segment response

- **seg-data:** at most seven bytes of segment data to be downloaded. Their meaning has to be specified by the application.

- **n:** indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.

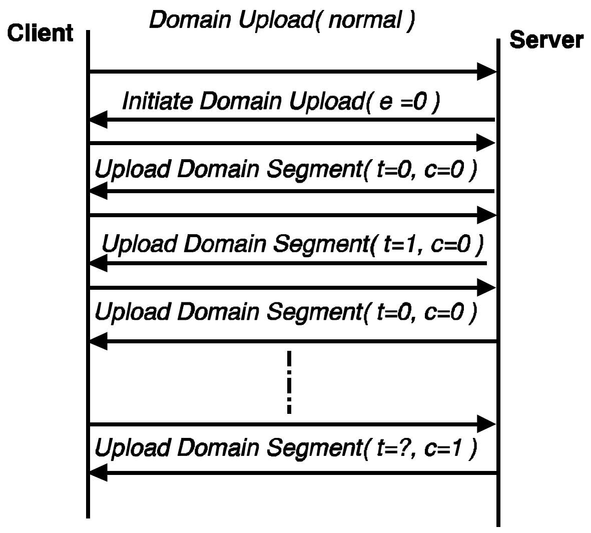
- **c:** indicates whether there are still more segments to be downloaded.
  0 more segments to be downloaded
  1: no more segments to be downloaded

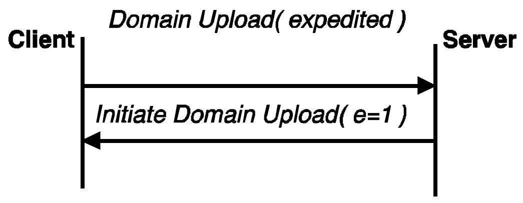
- **t:** toggle bit. This bit must alternate for each subsequent segment that is downloaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA

6.2 Upload Domain Protocol


Client
Domain Upload(normal)
Server
Initiate Domain Upload(e =0)
Upload Domain Segment(t=0, c=0)
Upload Domain Segment(t=1, c=0)
Upload Domain Segment(t=0, c=0)
Upload Domain Segment(t=?, c=1)



This protocol is used to implement the 'Domain Upload' service for multiplexed domains, see /2/. Multiplexed domains are uploaded as a sequence of zero or more 'Upload Domain Segment' services preceded by an 'Initiate Domain Upload' service. The sequence is terminated by:

- an 'Initiate Domain Upload' response/confirm with the e-bit set to 1, indicating the successful completion of an expedited upload sequence.

- an 'Upload Domain Segment' response/confirm with the c-bit set to 1, indicating the succesful completion of a normal upload sequence.

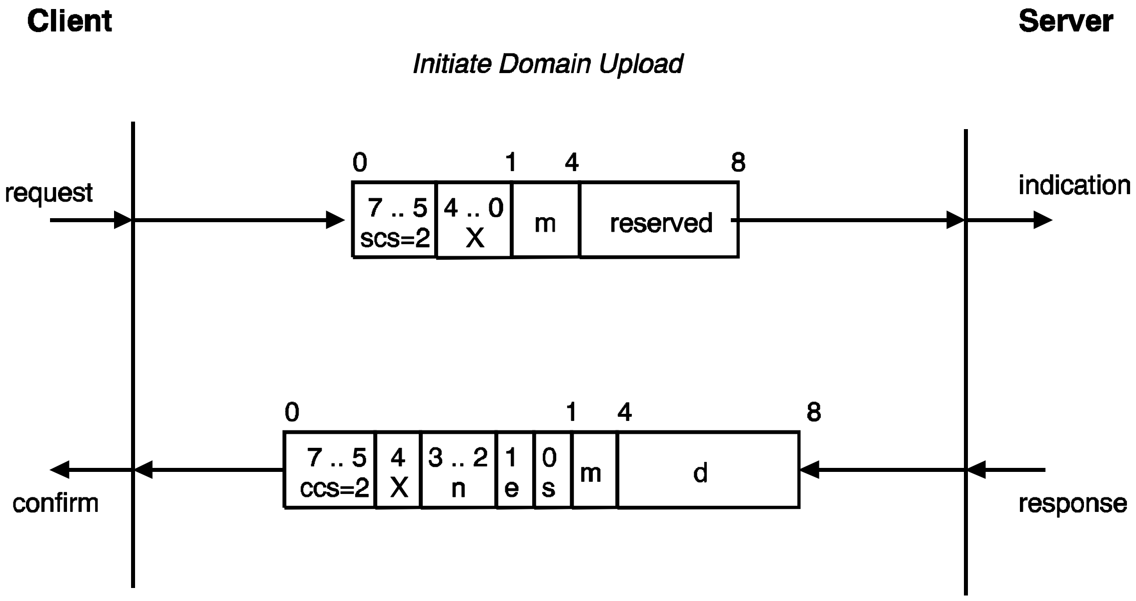
- an 'Abort Domain Transfer' request/indication, indicating the unsuccessful completion of the upload sequence.

- a new 'Initiate Domain Upload' request/indication, indicating the unsuccessful completion of the upload sequence and the start of a new sequence.

If in the upload of two consecutive segments the toggle bit does not alter, this must be treated as if an invalid COB was received (see Annex I). If such an error is reported to the application, the application may decide to abort the upload.

Initiate Domain Upload Protocol

This protocol is used to implement the 'Initiate Domain Upload' service for multiplexed domains, see /2/.



- **ccs:** client command specifier
  2: initiate upload request

- **scs:** server command specifier
  2: initiate upload response

- **n:** Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in d that do not contain data. Bytes [8-n, 7] do not contain segment data.

- **e**: transfer type
  0: normal transfer
  1: expedited transfer
- **s:** size indicator
  0: data set size is not indicated
  1: data set size is indicated

- **m**: multiplexor. It represents a value of the multiplexor data type attribute of the multiplexed domain.

- **d:** data

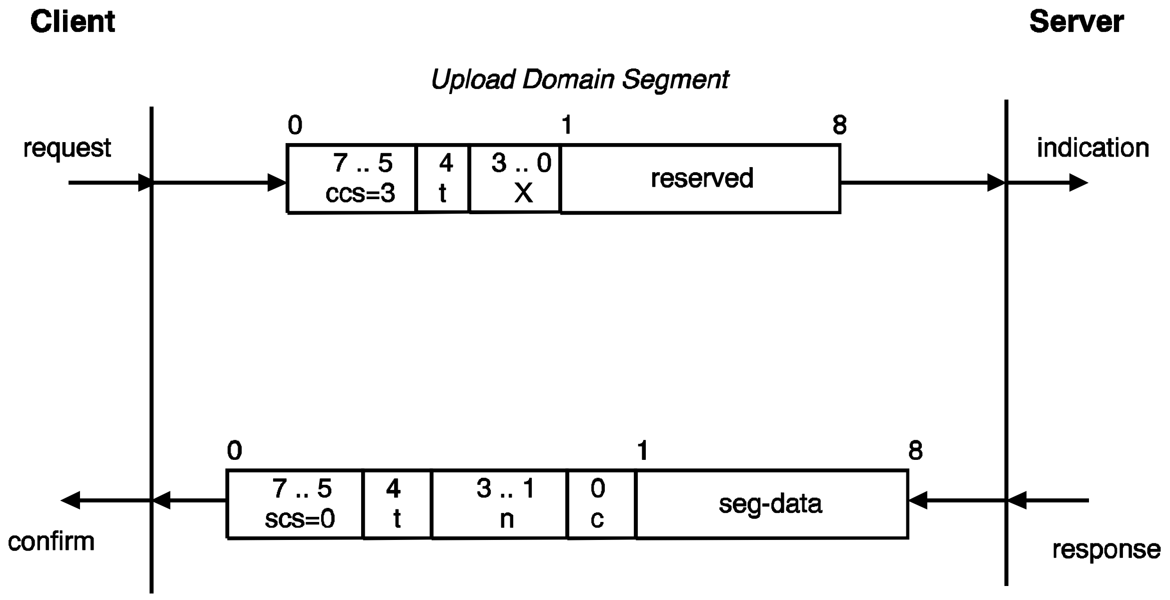
e = 0, s = 0:	d is reserved for further use by CiA.
e = 0, s = 1:	d contains the number of bytes to be uploaded. Byte 4 contains the lsb and byte 7 contains the msb.
e = 1:	d contains the data to be uploaded

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA

Upload Domain Segment Protocol

This protocol is used to implement the 'Upload Domain Segment' service for multiplexed domains, see /2/.



- **ccs:** client command specifier

3:	upload segment request

- **scs:** server command specifier

0:	upload segment response

- **t:** toggle bit. This bit must alternate for each subsequent segment that is uploaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

- **c:** indicates whether there are still more segments to be uploaded.

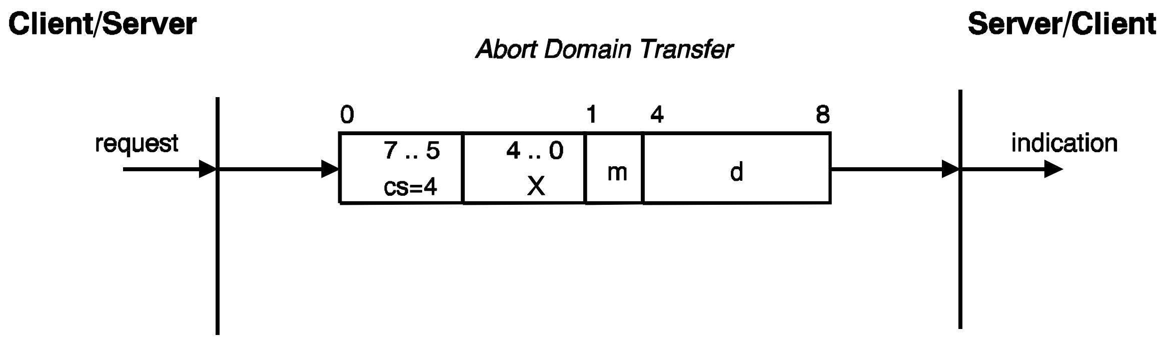
0:	more segments to be uploaded
1:	no more segments to be uploaded

- **seg-data:** at most seven bytes of segment data to be uploaded. Their meaning has to be specified by the application.
- **n:** indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
- **X:** not used, always 0
- **reserved:** reserved for further use by CiA

6.3 Abort Domain Transfer

Abort Domain Transfer Protocol

This protocol is used to implement the 'Abort Domain Transfer' Service for multiplexed domains, see /2/.



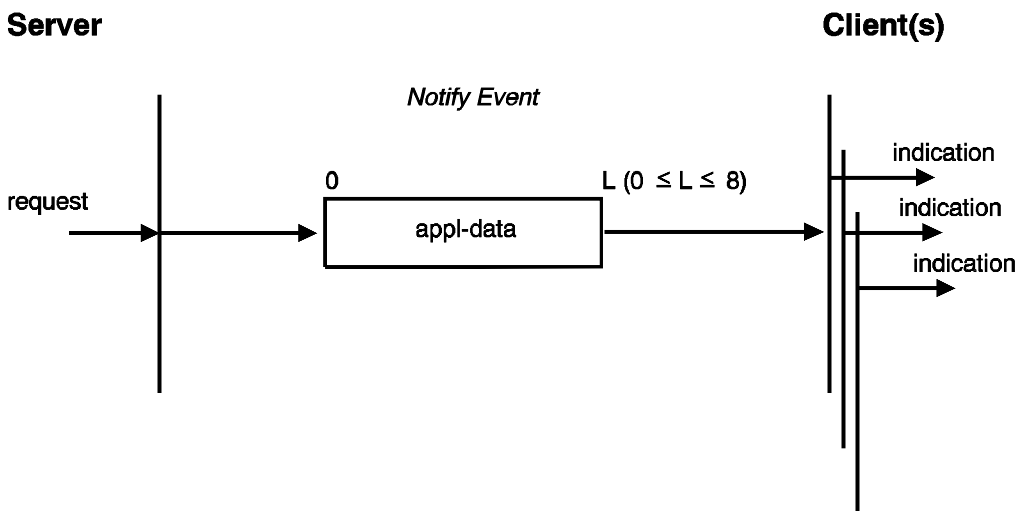
- **cs:** command specifier
  4: abort domain transfer
- **X:** not used, always 0
- **m**: multiplexor. It represents a value of the multiplexor data type attribute of the multiplexed domain.
- **d:** contains application specific data about the reason for the abort.

7. CMS EVENT PROTOCOLS

7.1 Uncontrolled Event

One unconfirmed service (Notify Event) is defined.

Notify Event Protocol

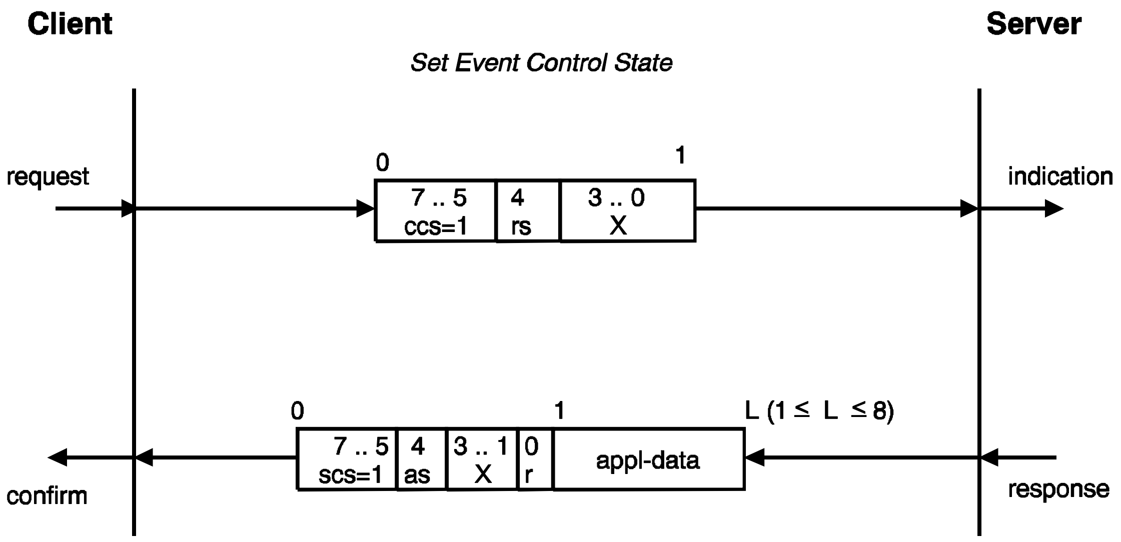


- **appl-data:** up to L bytes of application data representing a value of the data type attribute of the event.

7.2 Controlled Event

One confirmed (Set Event Control State) and one unconfirmed service (Notify Event) is defined.

Set Event Control State Protocol



- **ccs:** client command specifier
  1: Set_event_control_state

- **scs:** server command specifier
  1: Set_event_control_state

- **rs:** requested control status, only valid if ccs = 1, otherwise 0
  0: Disabled
  1: Enabled

- **X:** not used, always 0

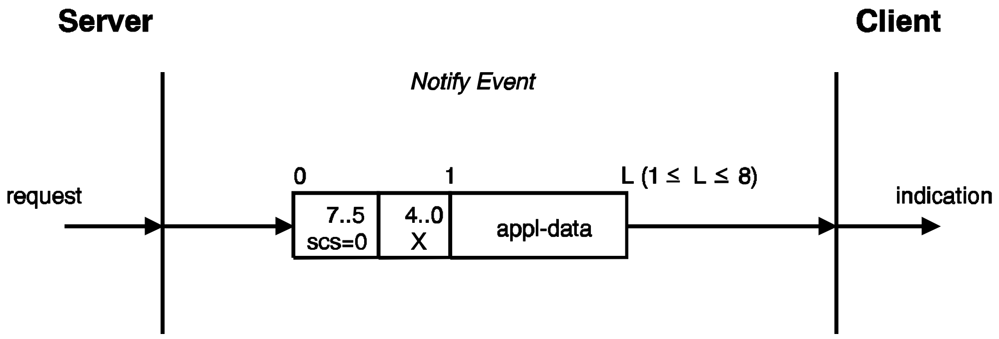
- **as:** actual control status, only valid if scs = 1 and r = 0, otherwise 0
  0: Disabled
  1: Enabled

- **r:** result, only valid if scs = 1, otherwise 0
  0: Success
  1: Failure

- **appl-data:** only valid if r = 1. If valid it contains up to L-1 bytes of application data representing a value of the error type attribute of the event.

- **X:** not used, always 0

Notify Event Protocol



- **scs:** server command specifier
  0: Notify-Event

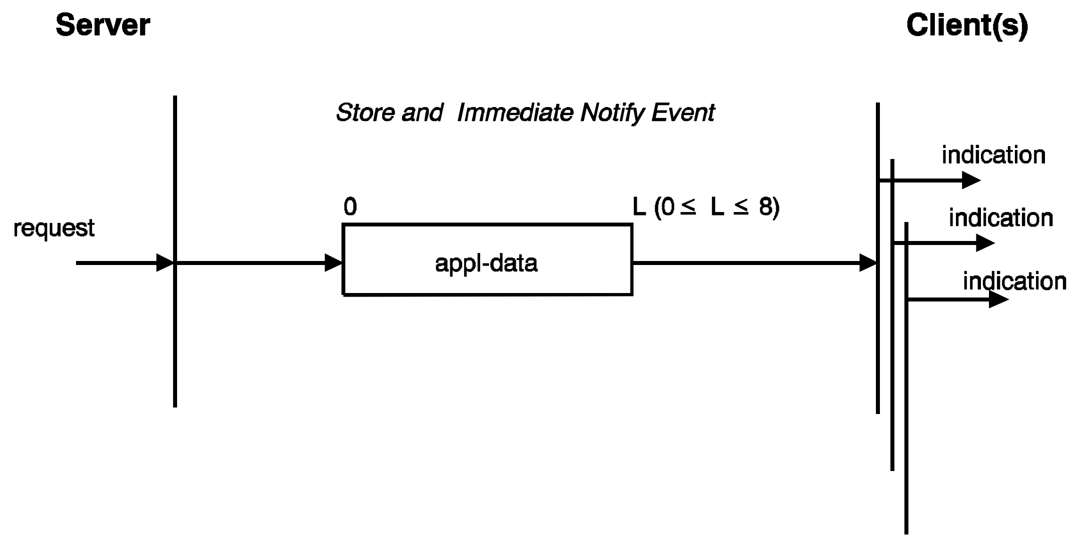
- **appl-data:** up to L-1 bytes of application data representing a value of the data type attribute of the event.

- **X:** not used, always 0

7.3 Stored Events

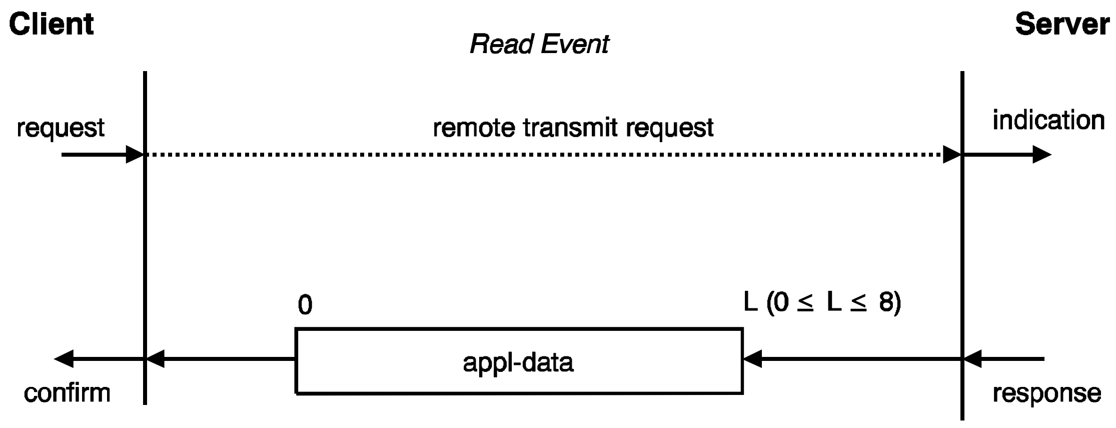
One unconfirmed service (Store and Immediate Notify) and one confirmed service (Read Event) is defined.

Store and Immediate Notify Protocol



- **appl-data:** up to L bytes of application data representing a value of the data type attribute of the stored event

Read Event Protocol



- **appl-data:** up to L bytes of application data representing a value of the data type attribute of the stored event

ANNEX I

IMPLEMENTATION RULES

When implementing the CMS protocols, the following rules have to be obeyed to guarantee interoperability. These rules deal with the following implementation aspects:

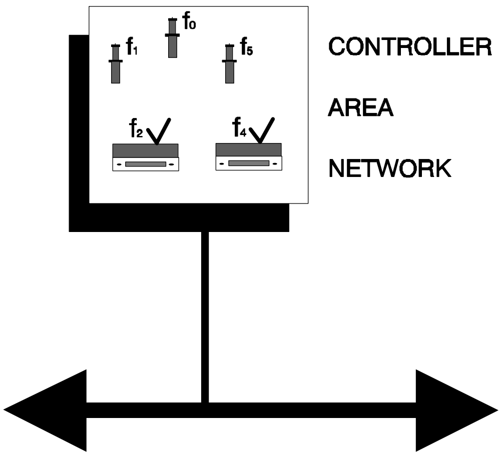
Invalid COB's

A COB is invalid if it has a COB-ID that is used by the CMS Protocol, but it contains invalid parameter values according to the CMS Protocol. This can only be caused by errors in the lower layers (see /1/) or implementation errors. Invalid COB's must be handled**locally** in an implementation specific way that does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. As far as the CMS Protocol is concerned, an invalid COB must be ignored.

Time-out's

Since COB's may be ignored, the response of a confirmed CMS service may never arrive. To resolve this situation, an implementation may, after a certain amount of time, indicate this to the service user (time-out). **A time-out is not a confirm of that CMS service**. A time-out indicates that the service has not completed yet. The application must deal with this situation. Time-out values are considered to be implementation specific and do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. However, it is recommended that an implementation provides facilities to adjust these time-out values to the requirements of the application.

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS202-3
February 1996

CMS Data Types and Encoding Rules

1. Scope

This document contains the encoding rules that are used to transfer CMS data values across the CAN Network and definitions application specifc extended data types. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. References

/1/ CiA/DS202-1, CMS Service Specification

/2/ CiA/DS207, Application Layer Naming Conventions

/3/ ANSI/IEEE Standard 754-1985 for Binary Floating-PointArithmetic.
Reprinted in: ACM SIGPLAN Notices 22(2), 9-25 (1987).

3. General Description

To be able to exchange meaningful data across the CAN network, the format of this data and its meaning have to be known by the sender and receiver(s). CMS models this by the concept of data types.

The CMS encoding rules define the representation of values of data types and the CAN network transfer syntax for the repesentations. Values are represented as bit sequences. Bit sequences are transferred in sequences of octetts (bytes). For numerical data types the CMS encoding is little endian style.

Applications require data types beyond the basic data types. Using the compound data type mechanism the list of available data types can be extended. Some general extended data types are defined.

4. Data Type Definitions

A data type determines a relation between values and encodings for data of that type. Data types are assigned names in their type definitions. The syntax of data and data type definitions is as follows.

```
<data definition>         ::=<type name> <data name>
<type definition>         ::=<constructor> <type name>
<constructor>             ::=<compound constructor> |
                             <basic constructor>
<compound constructor>    ::=<array constructor>|
                             <structure contructor>
<array constructor>       ::=ARRAY [<array length>] OF <type name>
<structure constructor>   ::=STRUCT OF <component list>
<component list>          ::=<component> |
                             <component>, <component list>
<component>               ::=<type name> <component name>
<basic constructor>       ::=BOOLEAN |
                             VOID<bit size> |
                             INTEGER<bit size> |
                             UNSIGNED<bit size> |
                             REAL32 |
                             NIL
<array length>            ::=positive integer
<bit size>                ::=1|2|...|64
<type name>               ::=symbolic name (see /2/)
<component name>          ::=symbolic name (see /2/)
<data name>               ::=symbolic name (see /2/)
```

Recursive definitions are not allowed.

The data type defined by <type definition> is called basic (resp.~compound) when the constructor is <basic constructor>(resp. <compound constructor>).

5. Bit Sequences

5.1 Definitions

A bit can take the values 0 or 1. Let $b_0,...,b_{n-1}$ be bits, n a non-negative integer. Then

$$b = b_0\ b_1\ ...\ b_{n-1}$$

is called a bit sequence of length $|b| = n$. The empty bit sequence of length 0 is denoted ε.

Examples: 10110100, 1, 101, etc. are bit sequences.

The inversion operator ($\neg$) on bit sequences assigns to a bit sequence

$$b = b_0\ b_1\ ...\ b_{n-1}$$

the bit sequence

$$\neg b = \neg b_0\ \neg b_1 ...\ \neg b_{n-1}$$

Here $\neg 0 = 1$ and $\neg 1 = 0$ on bits.

The basic operation on bit sequences is concatenation.

Let $a = a_0...a_{m-1}$ and $b = b_0\ ...\ b_{n-1}$ be bit sequences. Then the concatenation of a and b, denoted ab, is

$$ab = a_0\ ...\ a_{m-1}\ b_0\ ...\ b_{n-1}$$

Example: (10)(111) = 10111 is the concatenation of 10 and 111.

The following holds for arbitrary bit sequences a and b:

$$|ab| = |a| + |b|$$

and

$$\varepsilon a = a\varepsilon = a$$

5.2 Transfer Syntax

For transmission across a CAN network a bit sequence is reordered into a sequence of octetts. Here and in the following hexadecimal notation is used for octetts. Let $b=b_0...b_{n-1}$ be a bit sequence with $n \leq 64$. Denote l a non-negative integer such that $8(l-1) < n \leq 8l$. Then b is transferred in l octetts assembled as shown in Figure 1. The bits b_i, $i \geq n$ of the highest numbered octett are don't care bits.

Octett 1 is transmitted first and octett l is transmitted last. Hence the bit sequence is transferred as follows across the CAN network:

$$b_7, b_6,...,b_0,b_{15},...,b_8,...$$

	7	6	...	0
1	b_7	b_6	...	b_0
2	b_{15}	b_{14}	...	b_8
l	$b_{8l\text{-}1}$	$b_{8l\text{-}2}$	...	$b_{8l\text{-}8}$

Figure 1: Transfer Syntax for Bit Sequences

Example: The bit sequence 0011 1000 01 is transferred in two octetts: First $1c_h$ and then 02_h.

6. Basic Data Types

For basic data types <type name> equals the literal string of the associated constructor, e.g.,

BOOLEAN BOOLEAN

is the type definition for the Boolean data type.

6.1 NIL

Data of basic data type NIL is represented by ε.

6.2 Boolean

Data of basic data type BOOLEAN attains the values TRUE or FALSE. The values are represented as bit sequences of length 1. The value TRUE (resp. FALSE) is represented by the bit sequence 1 (resp.0).

6.3 Void

Data of basic data type VOIDn is represented as bit sequences of length$n$. The value of data of type VOIDn is undefined. The bits in the a sequence of data of type VOIDn must either be specified explicitly or else marked "don't care".

Data of type VOIDn is useful for reserved fields and for aligning components of compound values on octett boundaries.

6.4 Unsigned Integer

Data of basic data type UNSIGNEDn has values in the non-negative integers. The value range is 0, ..., 2^n-1. The data is represented as bit sequences of length n. The bit sequence

$b = b_0 \ldots b_{n-1}$

is assigned the value

$\text{UNSIGNEDn}(b) = b_{n-1}(2^n\text{-}1) + \ldots + b_1\, 2^1 + b_0\, 2^0$

Note that the bit sequence starts on the left with the least significant bit.

Example: The value 266 with data type UNSIGNED16 is transferred in two octetts across the bus, first $0a_h$ and then 01_h.

6.5 Signed Integer

Data of basic data type INTEGERn has values in the integers. The value range is -2^{n-1}, ..., $2^{n-1}-1$. The data is represented as bit sequences of length n. The bit sequence $b = b_0 .. b_{n-1}$ is assigned the value

$$\text{INTEGERn}(b) = b_{n-2}\, 2^{n-2} + ...+ b_1\, 2^1 + b_0\, 2^0 \quad \text{if } b_{n-1} = 0$$

and, performing two's complement arithmetic,

$$\text{INTEGERn}(b) = \text{INTEGERn}(\text{^b}) - 1 \quad \text{if } b_{n-1} = 1$$

Note that the bit sequence starts on the left with the least significant bit.

Example: The value -266 with data type INTEGER16 is transferred in two octetts across the bus, first $f6_h$ and then fe_h.

6.6 Floating Point Number

Data of basic data type REAL32 has values in the real numbers.
The data is represented as bit sequences of length 32. The encoding of values follows the IEEE 754-1985 Standard for floating point numbers, see the reprint /3/.

A bit sequence of length 32 either has a value (finite non-zero real number, ± 0, $\pm$ _) or is NaN (not-a-number). The bit sequence $b = b_0 \ldots b_{31}$ is assigned the value (finite non-zero number)

$$\text{REAL32}(b) = (-1)^S\, 2^{E - 127}\, (1 + F)$$

Here $S=b_{31}$ is the sign. $E = b_{30}\, 2^7 + \ldots + b_{23}\, 2^0$, $0 < E < 255$, is the un-biased exponent. $F = 2^{-23}\, (b_{22}\, 2^{22} + \ldots + b_1\, 2^1 + b_0\, 2^0)$ is the fractional part of the number. E =0 is used to represent $\pm$ 0. E =255 is used to represent infinities and NaN's. Note that the bit sequence starts on the left with the least significant bit.

Example: $6.25 = 2^{E\ -127}\ (1 + F)$ with $E = 129 = 2^7 + 2^0$ and $F = 2^{-1} + 2^{-4} = 2^{-23}(2^{22}+2^{19})$. Hence the number is represented as:

6.25:   0000 0000 0000 0000 0001 0011 0000 0010

Figure 2 shows example encodings for REAL32 as sequences of four octetts for transfer across the CAN network.

7. Compound Data Types

Type definitions of compound data types expand to a unique list of type definitions involving only basic data types. Correspondingly, data of compound type *'type_name'* are ordered lists of component data named component_i of basic type *'basic_type_i'*.
Compound data types constructors are ARRAY and STRUCT OF.

```
STRUCT OF
        <basic_type_1>       <component_1>,
        <basic_type_2>       <component_2>,
        ...                  ...
        <basic_type_N>       <component_N>
<type_name>

ARRAY [<length>]  OF  <basic_type>   <type_name>
```

The bit sequence representing data of compound type is obtained by concatenating the bit sequences representing the component data. Assume that the components *'component_i'* are represented by bit sequences b_i, for $i = 1, \ldots, N$ Then the compound data is represented by the concatenated sequence $b_1 \ldots b_N$.

Example: Consider the data type

```
STRUCT OF
        INTEGER10        i,
        UNSIGNED5        u
NewData
```

Assume $i = -423$ and $u = 30$. Let $b(i)$ and $b(u)$ denote the bit sequences representing the values of i and u, respectively. Then:

$b(i)$	=	1001101001
$b(u)$	=	01111
$b(iu) = b(i)\, b(u)$	=	1001101001 01111

The value of the structure is transferred with two octetts, first 59_h and then 79_h.

8. Extended Data Types

The extended data types consist of the basic data types and the compound data types defined in the following subsections.

8.1 Octett String

The data type *OctettString<length>* is defined below. *length* is the length of the octett string.

```
ARRAY [<length>]   OF UNSIGNED8        OctettString<length>
```

8.2 Visible String

The data type *VisibleString<length>* is defined below. The admissable values of data of type VisibleChar are 0_h and the range from 20_h to $7E_h$. The data are interpreted as ASCII characters. *length* is the length of the visible string.

```
UNSIGNED8                             VisibleChar
ARRAY[<length>]  OF VisibleChar       VisibleString<length>
```

8.3 Date

The data type *Date* is defined below. It follows from the definition and the encoding rules that data of type *Date* is represented as bit sequences of length 56.

```
STRUCT OF
        UNSIGNED16          ms,
        UNSIGNED6           min,
        VOID2               reserved_1,
        UNSIGNED5           hour,
        VOID2               reserved_2,
        BOOLEAN             su,
        UNSIGNED5           day_of_month,
        UNSIGNED3           day_of_week,
        UNSIGNED6           month,
        VOID2               reserved_3,
        UNSIGNED7           year,
        VOID1               reserved_4
Date
```

Figure 1 contains descriptions and value ranges for the components of data of type *Date*. The components reserved*_i, i* =1,...,4, are reserved with all bits equal 0.

Component	Description	Value Range
ms	milliseconds	0,...,59999
min	minutes	0,...,59
hour	hour	0,...,23
su	standard or summer time	0 = standard, 1 = summer
day_of_month	day of month	1,...,31
day_of_week	day of week	1 = monday, 7 = sunday
month	month	1,...,12
year	year modulo centuries	0,...,99

Figure 1:Descriptions for *Date*

8.4 Time of Day

The data type *TimeOfDay* represents absolute time. It follows from the definition and the encoding rules that *TimeOfDay* is represented as bit sequences of length 48.

Component ms is the time in milliseconds after midnight. Component days is the number of days since January 1, 1984.

```
STRUCT OF
        UNSIGNED28     ms,
        VOID4          reserved_1,
        UNSIGNED16     days
TimeOfDay
```

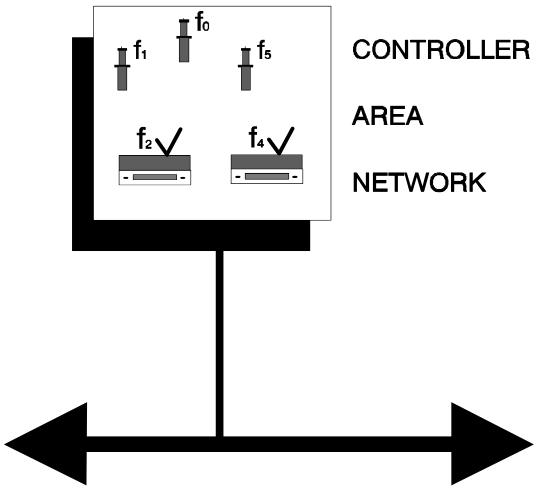
8.5 Time Difference

The data type *TimeDifference* represents a time difference. It follows from the definition and the encoding rules that *TimeDifference* is represented as bit sequences of length 48.

Time differences are sums of numbers of days and milliseconds. Component ms is the number milliseconds. Component days is the number of days.

```
STRUCT OF
        UNSIGNED28     ms,
        VOID4          reserved_1,
        UNSIGNED16     days
TimeDifference
```

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS203-1
February 1996

NMT Service Specification

1. SCOPE

This document contains the Network Management Service Specification. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS203-2, NMT Protocol Specification

/3/: CiA/DS207, Application Layer Naming Conventions

3. GENERAL DESCRIPTION

3.1 NMT PERSPECTIVE

NMT is one of the application layer entities in the CAN Reference Model, see /1/.

The NMT aids in the development of distributed applications. Due to the fact that an application is distributed, certain events have to be handled (e.g failures of parts of the applicaton) that would not occur if the same application had not been distributed.

The application has to deal with these network management aspects, although they have nothing to do with the goal of the application (e.g controlling a process). These aspects are the consequence of building a distributed application and must be compared to the advantages of building a distributed application.

3.2 NMT Objects and Services

A CAN network consists of modules that are connected by one physical CAN bus. The NMT uses three objects to model a CAN network:

- **the network object.** The network object represents the set of all modules in a CAN network. A network can contain at most 255 modules. The network object may exist on one module only, called the NMT Master.

- **the remote node object.** Each module in the network that is managed by the NMT services is represented by a remote node object on the NMT Master.

- **the node object.** Each module that is managed by the NMT services is represented by a node object on that module (including the NMT Master). A module where a node object exists is called an NMT Slave.

Each NMT Slave and its node object is uniquely identified in the network by its NMT Address. The syntax of an NMT Addres is defined in /3/. The NMT Address of an NMT Slave cannot be changed by the NMT services but can be configured via the LMT Service Element (see /1/) or in an application specific way that does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

For each NMT Slave there must exist one remote node object with the same NMT Address on the NMT Master. A node object and the remote node object that have the same NMT Address are called peers. Each remote node object communicates with its peer via the NMT Protocol as defined in /2/. The NMT Protocol uses a Node-ID to address an NMT Slave. The syntax of a Node-ID is defined in /3/. A unique Node-ID is assigned to the node object of each NMT Slave and its peer by the NMT Master. Peers have the same Node-ID. The NMT model of a CAN network is depicted in fig. 1. Note that it is possible that a module is an NMT Master and an NMT Slave at the same time.

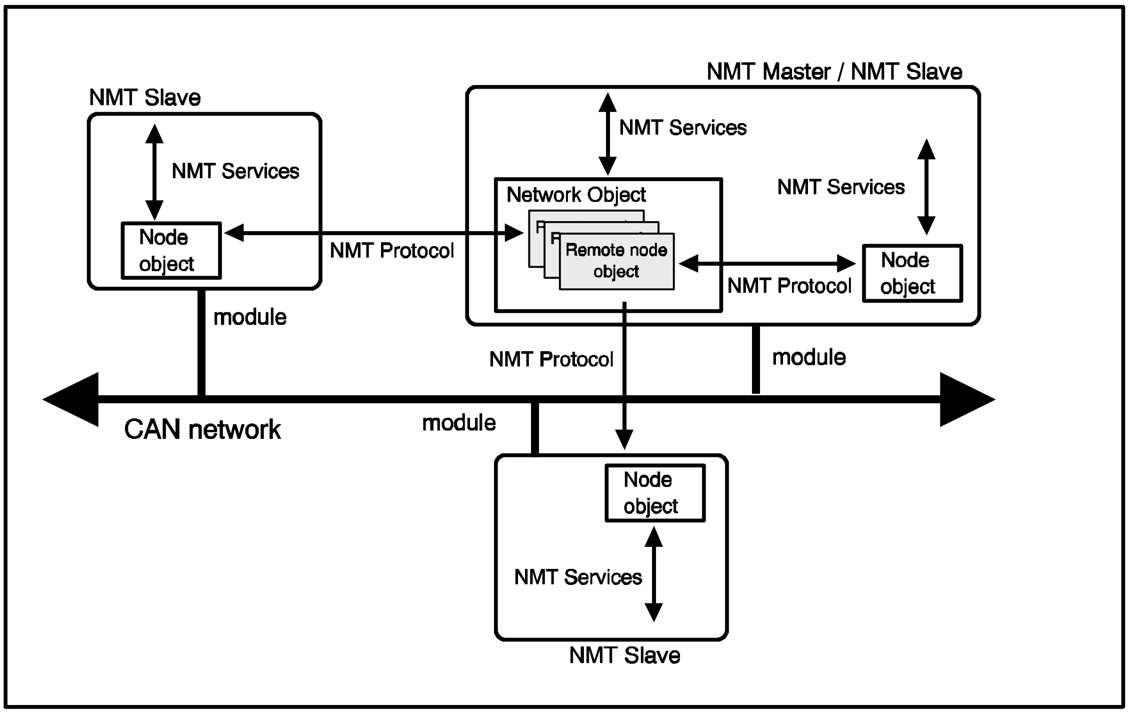


Fig. 1: The NMT Model

The NMT offers the following services:

- **Module Control Services:** through these services, the NMT Master initializes NMT Slaves that want to take part in the distributed application and allows them to communicate with each other through the CMS Service Element (see /1/). To this purpose, an NMT Slave has to define all the CMS objects it needs. The COB's required by the CMS protocol for these CMS objects have to obtain COB identifiers and inhibit-times. These can be assigned statically by the application or dynamically by the Distributor Service Element (see /1/). Once the COB identifiers have been obtained, the NMT Master can indicate the NMT Slave that it may or may not access the network through the CMS Service Element. Through the Module Control Services the NMT Master controls the sequence of these actions for each NMT Slave. Through the Module Control services, the NMT Master and NMT Slave also negotiate about parameters for the NMT Protocol.

- **Error Control Services:** through these services, the NMT detects failures in a CAN network. Local failures are caused by errors detected in the Data Link or Physical Layer (see /1/) of a module or by other application specific conditions on that module that prevent it from taking part in the distributed application. These conditions do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. Remote failures are failures detected by the Node Guarding Protocol (see section 6 of this docu ment).

- **Configuration Control Services:** through these services, the NMT can up and download configuration data from respectively to a module in the CAN network. The meaning of the configuration data that is up- or downloaded is application specific and does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. It may be executable code, parameters, data or anything else.

3.3 NMT Capabilities

NMT capabilities indicate categories of network management functionality that may or may not be present in the network. Capabilities that affect all modules in a network are called network capabilities and can only be configured on the NMT Master. Capabilities that affect only one module in the network are called node capabilities and can only be configured on an NMT Slave. The following capabilities are defined:

- Network Management capability. This network capability implements the mandatory module control services on the NMT Master. These services can only be executed with NMT Slaves that have the:

- **Node Management capability.** This node capability implements the mandatory module control services on an NMT Slave. These services can only be executed if the NMT Master has the Network Management capability.

- **Network Error capability.** This network capability implements the mandatory error control services on the NMT Master. These services can only be executed with NMT Slaves that have the:

- **Node Error capability.** This node capability implements the mandatory error control services on an NMT slave. These services can only be executed if the NMT Master has the Network Error capability

- **Network Configuration capability.** This network capability implements the mandatory configuration control services on the NMT Master. These services can only be executed with NMT Slaves that have the:

- **Node Configuration capability.** This node capability implements the mandatory configuration control services on an NMT Slave. These services can only be executed if the NMT Master has the Network Configuration capability

How to configure NMT capabilities on an NMT Master and NMT Slave does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

3.4 Network Classes

The network class indicates the network capabilities that have been configured on the NMT Master:

- **Class 0:** no Network Management capability. As a consequence, module control, error control, configuration control, and dynamic identifier/inhibit time distribution is not possible for any of the modules in the network.

- **Class 1:** Network Management capability, no Network Error capability, no Network Configuration capability. This is a network where error control and configuration control is not possible for any of the modules in the network. Module control and dynamic identifier/inhibit-time distribution is possible.

- **Class 2:** Network Management capability, Network Error capability, no Network Configuration capability. This is a network where module control, error control, and dynamic identifier/inhibit-time distribution is possible. Configuration control is not possible for any of the modules in the network.

- **Class 3:** Network Management capability, no Network Error capability, Network Configuration capability. This is a network where module control, configuration control, and dynamic identifier/inhibit-time distribution is possible. Error control is not possible for any of the modules in the network.

- **Class 4:** Network Management capability, Network Error capability, Network Configuration capability. This is a network where module control, error control, configuration control and dynamic identifier/inhibit-time distribution is possible.

3.5 Node Classes

The node class indicates the node capabilities that have been configured on an NMT Slave:

- **Class 0:** no Node Management capability. This is a module that cannot be managed through the module control services of the NMT Master. As a consequence, error control, configuration control, and dynamic identifier/in hibit-time distribution is not possible for this module.

- **Class 1:** Node Management capability, no Node Error capability, no Node Configuration capability. This is a module that can be managed through the module control services of the NMT Master and for which dynamic identifier/inhibit-time distribution is possible. Error control and configuration control is not possible for this module.

- **Class 2:** Node Management capability, Node Error capability, no Node Configuration capability. This is a module that can be managed through the module and error control services of the NMT Master and for which dynamic identifier/inhibit-time distribution is possible. Configuration control is not possible for this module.

- **Class 3:** Node Management capability, no Node Error capability, Node Configuration capability. This is a module that can be managed through the module and configuration control services of the NMT Master and for which dynamic identifier/inhibit-time distribution is possible. Error control is not possible for this module.

- **Class 4:** Node Management capability, Node Error capability, Node Configuration capability. This is a module that can be managed through the module-, error-, and configuration control services of the NMT Master and for which dynamic identifier/inhibit-time distribution is possible.

3.6 NMT Service Descriptions

The NMT services are described in a tabular form that contains the parameters of each service primitive that is defined for that service. The primitives that are defined for a particular service determine the service type (e.g unconfirmed, confirmed, etc.). How to interprete the tabular form and what service types exist is defined in /1/. In the service descriptions, [a, b] denotes the range of integers from a to b with a and b included. If a > b, the range is empty.

4. NMT OBJECTS

4.1 Network Object

Network Attributes:

- **remote node set:** the set of remote node objects that form the network
- **class:** the network class. A value in the range [0, 4].

4.2 Remote Node Object

Remote Node Attributes

- **NMT Address:** see /3/. This attribute uniquely identifies the remote node object in the remote node set.
- **state:** one of the values {DISCONNECTED, CONNECTED, PREPARED, OPERATIONAL}. This attribute indicates the state of the remote node object. The state is controlled by the NMT services according to the state diagrams in section 6 of this document.
- **Node-ID:** a value in the range [1, 255]. This attribute uniquely identifies the remote node object in the remote node set of the network object if and only if the state of the remote node object is not DISCONNECTED. It is identical to the Node-ID attribute of its peer.

4.3 Node Object

Node Attributes

- **NMT Address:** see /3/. This attribute uniquely identifies the NMT Slave and its node object in the network.
- **state:** one of the values {DISCONNECTED, CONNECTING, PREPARING, PREPARED, OPERATIONAL}. This attribute indicates the state of the NMT

Slave and its node object. The state is controlled by the NMT services according to the state diagrams in the section 6 of this document.

- **Node-ID:** a value in the range [1, 255]. This attribute uniquely identifies the NMT Slave and its node object in the network if and only if the state of the node object is neither DISCONNECTED nor CONNECTING. It is identical to the Node-ID attribute of its peer.

- **class:** the node class. A value in the range [0, 4].

In the remainder of this document, the attributes of a node object of an NMT Slave are also considered to be attributes of the NMT Slave. E.g the state of an NMT Slave denotes the state of its node object.

5. NMT SERVICES

There can be atmost one confirmed NMT service outstanding in the complete network.

5.1 Module Control Services

The mandatory module control services need to be implemented on the NMT Master if and only if the Network Management capability has been configured on the NMT Master. The mandatory module control services need to be implemented on an NMT Slave if and only if the Node Management capability has been configured on that NMT Slave.

Create Network

Parameter	*Request*
Argument class	**Mandatory** mandatory

Through this service the NMT Master creates a network object with the requested attributes. The service will only be executed if no network object exists. After completion of the service, the remote node set will be empty. The service is local and mandatory.

Add Remote Node

Parameter	*Request*
Argument NMT_Address	**Mandatory** mandatory

Through this service the NMT Master creates a remote node object with the requested attributes and inserts it in the remote node set of the network object. The service will only be executed if a network object exists and if there are less than 255 remote node objects. After completion of the service, the state of the remote node object will be DISCONNECTED. The service is local and mandatory.

Remove Remote Node

Through this service the NMT Master removes the remote node object identified by NMT_Address from the remote node set of the network object. The service will only be

Parameter	*Request*
Argument NMT_Address	**Mandatory** mandatory

executed if NMT_Address identifies a remote node object whose state is DISCONNECTED. The service is local and mandatory.

Create Node

Parameter	*Reauest*
Argument NMT_Address class	**Mandatory** mandatory mandatory

Through this service an NMT Slave creates a node object with the requested attributes. The service will only be executed if no node object already exists. After completion of the service, the state of the NMT Slave will be DISCONNECTED. The service is local and mandatory.

Delete Node

Parameter	*Request*
Argument	**Mandatory**

Through this service an NMT Slave deletes its node object. The service will only be executed if the state of the NMT Slave is DISCONNECTED. The service is local and mandatory.

Identify Remote Nodes

Through this service the NMT Master requests all NMT Slaves whose NMT Address meets the NMT_Address_selection and whose state is CONNECTING, to identify them selves through the 'Identify Node' service. The service is unconfirmed and optional.

Parameter	*Request/Indication*
Argument NMT_Address_selection	**Mandatory** mandatory

Identify Node

Parameter	*Request/Indication*
Argument	**Mandatory**

Through this service an NMT Slave indicates the NMT Master that there is an NMT Slave whose state is CONNECTING. The service will only be executed if the state of the NMT Slave is CONNECTING. The service is unconfirmed and optional.

Connect Node

Parameter	*Request*
Argument download	**Mandatory** optional

Through this service the NMT Slave sets its state from DISCONNECTED to CONNECTING. The NMT Master may optionally be requested to download a configuration to the NMT Slave. The service will only be executed if the state of the NMT Slave is DISCONNECTED. The service is local and mandatory.

Connect Remote Node

Through this service the NMT Master sets the state of the NMT Slave identified by NMT_Address from CONNECTING to PREPARING. The service will only be executed if NMT_Address identifies a remote node object whose state is DISCONNECTED.

The service is confirmed and mandatory. The Remote Result parameter will confirm the success or failure. If the state of the NMT Slave is not CONNECTING the service will fail. In case of success the following holds:

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
NMT_Address	mandatory	
Remote Result		**Mandatory**
success		selection
download		optional
failure		selection
reason		optional

- the state of the remote node object identified by NMT_Address will be CONNECTED
- the state of the NMT Slave identified by NMT_Address will be PREPARING.
- the Node-ID attribute of the remote node object identified by NMT_Address and its peer have been assigned a value.

In case of successs it will be confirmed whether the NMT Slave needs a configuration to be downloaded. In case of a failure, the state of the remote node object identified by NMT_Address and its peer will be DISCONNECTED and optionally the reason may be confirmed.

Prepare Remote Node

Through this service the NMT Master sets the state of the NMT Slave identified by Node_ID from PREPARING to PREPARED. The service will only be executed if Node_ID identifies a remote node object whose state is CONNECTED.

Prior to the state transition, the NMT Slave is allowed to obtain identifiers and inhibit times from the DBT Service Element (see /1/) for the COB's required by the CMS protocol for the CMS objects as defined on that NMT Slave. The NMT Master may optionally request that previously obtained identifiers and inhibit times are discarded. If this is not requested, the NMT Slave itself may decide whether or not to discard them. If the NMT Slave does not obtain identifiers and inhibit times from the DBT Service Element, this parameter must be ignored by the NMT Slave.

The service is confirmed and mandatory. The Remote Result parameter will confirm the success or failure of the request. If the state of the NMT Slave is not PREPARING the

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Node_ID	mandatory	
discard	optional	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

service will fail. If the NMT Slave does not obtain identifiers and inhibit times from the DBT Service Element, the service will succeed. In case of success, the state of the remote node object identified by Node-ID and its peer will be PREPARED. In case of a failure, the state of the remote node object identified by Node_ID and its peer will be DISCONNECTED and optionally the reason may be confirmed.

Start Remote Node

Parameter	*Indicatio/Requestn*
Argument	**Mandatory**
Node_ID	selection
all	selection

The service will only be executed for the selected remote node objects whose state is PREPARED. Through this service the NMT Master sets the state of the selected NMT Slaves from PREPARED to OPERATIONAL. Only NMT Slaves whose state is OPER ATIONAL may execute services of the CMS Service Element (see /1/).

The service is unconfirmed and mandatory. If the state of an NMT Slave is not PREPARED no state transition will occur on the NMT Slave. After completion of the service, the state of the selected remote node objects will be OPERATIONAL.

Stop Remote Node

The service will only be executed for the selected remote node objects whose state is OPERATIONAL. Through this service the NMT Master sets the state of the selected NMT

Slaves from OPERATIONAL to PREPARED. NMT Slaves whose state is not OPERATIONAL may not execute services of the CMS Service Element (see /1/).

Parameter	*Request/Indication*
Argument	**Mandatory**
Node _ID	selection
all	selection

The service is unconfirmed and mandatory. If the state of an NMT Slave is not OPERATIONAL no state transition will occur on the NMT Slave. After completion of the service, the state of the selected remote node objects will be PREPARED.

Disconnect Remote Node

Parameter	*Request/Indication*
Argument	**Mandatory**
Node_ID	selection
all	selection

The service will only be executed for the selected remote node objects. Through this service the NMT Master sets the state of the selected NMT Slaves to DISCONNECTED independent of their present state and undefines their Node-ID attribute.

The service is unconfirmed and mandatory. After completion of the service, the state of the selected remote node objects will be DISCONNECTED and their Node-ID attributes are undefined.

Disconnect Node

Parameter	*Request*
Argument	**Mandatory**

Through this service, the NMT Slave sets the state of the node object to DISCON NECTED independent of its present state and undefines its Node-ID attribute. The service will only be executed if a node object exist. The service is local and mandatory.

5.2 Error Control Services

The mandatory error control services need to be implemented on the NMT Master if and only if the Network Error capability has been configured on the NMT Master. The mandatory error control services need to be implemented on an NMT Slave if and only if the Node Error capability has been configured on that NMT Slave.

Network Event

Parameter	*Indication*
Argument	**Mandatory**
remote_error	selection
Node_ID	mandatory
local error	selection
state	mandatory
occurred	selection
resolved	selection

The service is provider initiated and mandatory and does not affect the state of the network object. A network object must exist. Through this service, the NMT service provider on the NMT Master indicates that one of the following has occurred:

- a remote error occurred or has been resolved for the remote node object identified by Node_ID and its peer.
- a local error occurred or has been resolved on the NMT Master.

Node Event

The service is provider initiated and mandatory and does not affect the state of the network object. A node object must exist. Through this service, the NMT service provider on an NMT Slave indicates that one of the following has occurred:

- a remote error occurred or has been resolved for the NMT Master.
- a local error occured or has been resolved on the NMT Slave.

Parameter	*Indication*
Argument	**Mandatory**
remote_error	selection
local error	selection
state	mandatory
occurred	selection
resolved	selection

5.3 Configuration Control Services (non-segmented)

When using these services the NMT is responsible for the segmentation of the configuration. All non-segmented configuration control services are optional.

Configuration Download

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Node_ID	mandatory	
data	mandatory	
size	optional	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

Through this service the NMT Master downloads configuration data from the NMT Master to the NMT Slave identified by Node_ID. The data and optionally its size are indicated.

The service will only be executed if Node_ID identifies a remote node object. The service is confirmed and optional. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason may be indicated.

Configuration Upload

Through this service the NMT Master uploads configuration data from the the NMT Slave identified by Node_ID to the NMT Master.

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Node_ID	mandatory	
Remote Result		**Mandatory**
success		selection
data		mandatory
size		optional
failure		selection
reason		optional

The service will only be executed if Node_ID identifies a remote node object. The service is confirmed and optional. The Remote Result parameter will indicate the success or failure of the request. In case of success, the data and optionally its size are confirmed. In case of a failure, optionally the reason may be confirmed.

5.4 Configuration Control Services (segmented)

When using these services the application is responsible for the segmentation of the configuration. The mandatory segmented configuration control services need to be implemented on the NMT Master if and only if the Network Configuration capability has been configured on the NMT Master. They need to be implemented on an NMT Slave if and only if the Node Configuration capability has been configured on that NMT Slave.

Initiate Configuration Download

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Node_ID	mandatory	
size	optional	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

Through this service the NMT Master prepares the NMT Slave identified by Node_ID for downloading a configuration from the NMT Master. Optionally the size of the configuration to be downloaded may be indicated.

The service will only be executed if Node_ID identifies a remote node object. The service is confirmed and mandatory. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason may be confirmed.

Initiate Configuration Upload

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Node_ID	mandatory	
Remote Result		**Mandatory**
success		selection
size		optional
failure		selection
reason		optional

Through this service the NMT Master prepares the NMT Slave identified by Node_ID for uploading a configuration to the NMT Master. The service will only be executed if Node_ID identifies a remote node object. The service is confirmed and mandatory. The Remote Result parameter will indicate the success or failure of the request. In case of success, optionally the size of the configuration to be uploaded is confirmed. In case of a failure, optionally the reason may be confirmed.

Download Configuration Segment

Through this service the NMT Master transfers the data of the next segment to the NMT Slave identified by Node_ID. The data and optionally its size are indicated. The continue parameter indicates whether there are still more segments to be downloaded or that this was the last segment to be downloaded. A successful 'Initiate Configuration Download' service must have been executed prior to this service.

The service will only be executed if Node_ID identifies a remote node object. The service is confirmed and mandatory. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason is confirmed. In case of success, the NMT Slave identified by Node_ID has accepted the segment data and is ready to accept the next segment.

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Node_ID	mandatory	
data	mandatory	
size	optional	
continue	mandatory	
more	selection	
last	selection	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

Upload Configuration Segment

Parameter	Request/Indication	Response/Confirm
Argument	**Mandatory**	
Node_ID	mandatory	
Remote Result		**Mandatory**
success		selection
data		mandatory
size		optional
continue		mandatory
more		selection
last		selection
failure		selection
reason		optional

Through this service the NMT Master requests the NMT Slave identified by Node_ID to supply the data of the next segment to the NMT Master. A successful 'Initiate Configuration Upload' service must have been executed prior to this service.

The service will only be executed if Node_ID identifies a remote node object. The service is confirmed and mandatory. The Remote Result parameter will indicate the success or failure of the request. In case of a failure, optionally the reason may be confirmed. In case of success, the data and optionally its size are confirmed. The continue parameter confirms whether there are still more segments to be uploaded or that this was the last segment to be uploaded.

Abort Configuration Transfer

This service aborts the up- or download of a configuration to or from the NMT Slave identified by Node_ID. The service may be executed at any time by the NMT Master. An NMT Slave will only execute this service as a response to any of the other configuration control services. Optionally the reason may be indicated. On the NMT Master the service will only be executed if Node_ID identifies a remote node object. On the NMT Slave the service will only be executed if a node object exists and the Node_ID parameter must be ignored.

Parameter	*Request/Indication*
Argument	**Mandatory**
Node_ID	mandatory
reason	optional

The service is unconfirmed and mandatory. If the NMT Master has a confirmed configuration service outstanding for the NMT Slave identified by Node_ID, the Abort Indication is taken to be the Confirm of that service.

Verify Configuration

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Node-ID	mandatory	
check_sum	mandatory	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

Through this service the NMT Master requests the NMT Slave identified by Node_ID to verify if check_sum matches with the last configuration that was successfully downloaded. The value of the check_sum is application specific and does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

The service will only be executed if Node_ID identifies a remote node object. The service is confirmed and optional. The Remote Result parameter will confirm the succes or failure of the verification. In case of a failure optionally the reason may be confirmed.

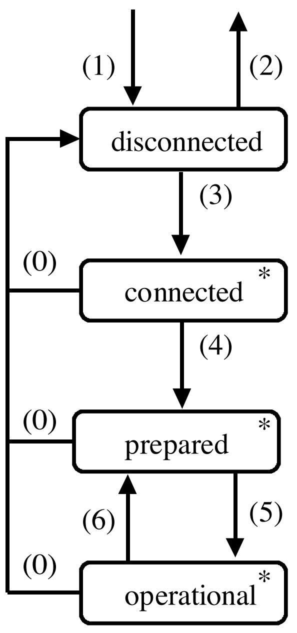
6. STATE TRANSITION DIAGRAMMS

Each remote node object on the NMT Master and its peer on an NMT Slave maintain a state transition diagram, see fig.2 and fig.3. The state transitions in these diagrams are caused by the NMT services as indicated in these diagrams. Some services can only cause a state transition in one diagram, while others may cause a state transition in both diagrams. Services that do not cause a state transition are not drawn in the state transition diagrams. An error response/confirm indicates that for the service involved, the 'failure' selection in the service specification will be selected.

Depending on the NMT services that were executed, the NMT Master assumes that the node object of an NMT Slave is in a certain state. To detect if this assumption is true, the NMT Master regularly retrieves the state of an NMT Slave and compares it to the state of its peer. This meachanism is called Node Guarding and the protocol the Node Guarding Protocol. If the comparison fails or if the state of an NMT Slave could not be retrieved at all, this is indicated to the NMT Master through the 'Network Event' service as a remote error. If the node state of an NMT Slave has not been retrieved during a certain period of time by the NMT Master, this is indicated to the NMT Slave through the 'Node Event' service as a remote error. Note that the 'Network Event' and 'Node Event' services do not cause a state transition.

The Node Guarding Protocol is active if and only if the NMT Master has the Network Error capability, the NMT Slave has the Node Error capability, and if the (remote) node state is not DISCONNECTED or CONNECTING, see fig.2 and fig.3. Note that the 'Disconnect Remote Node' service causes the Node Guarding Protocol to stop for that NMT Slave, since the state of the remote node object becomes DISCONNECTED.

If the Node Guarding Protocol functions again normally after a remote error has been reported and no state transition has occurred, this is indicated to the NMT Master and NMT Slave through the 'Network Event' and the 'Node Event' service respectively.

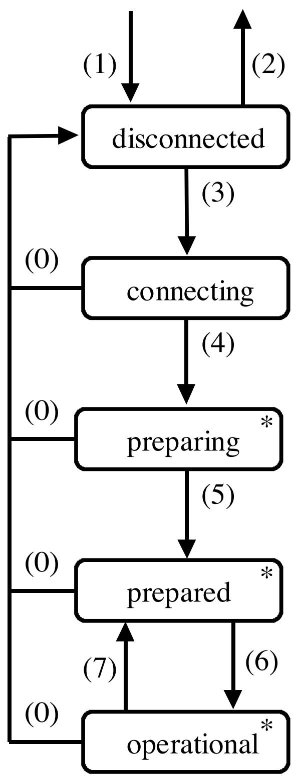


* = Node Guarding active if configured

(0) = Disconnect_Remote_Node request
(0) = Error Confirm

(1) = Add_Remote_Node request
(2) = Remove_Remote_Node request
(3) = Connect_Remote_Node confirm
(4) = Prepare_Remote_Node confirm
(5) = Start_Remote_Node request
(6) = Stop_Remote_Node request

Fig. 2: Remote Node State Diagram



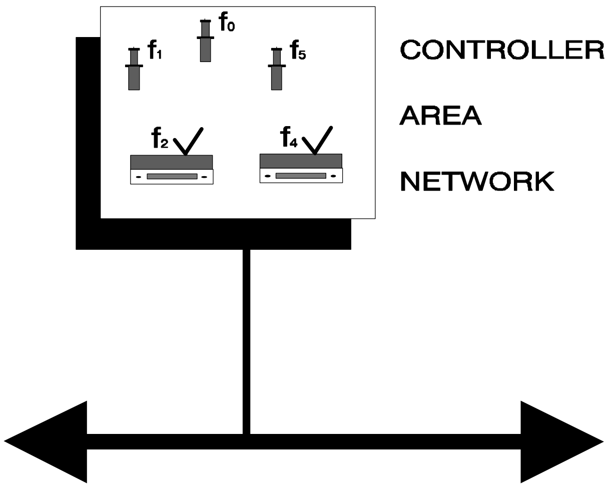
* = Node Guarding active if configured

(0) = Disconnect_Remote_Node indication
(0) = Disconnect_Node request
(0) = Error response

(1) = Create_Node request
(2) = Delete_Node request
(3) = Connect_Node request
(4) = Connect_Remote_Node response
(5) = Prepare_Remote_Node response
(6) = Start_Remote_Node indication
(7) = Stop_Remote_Node indication

Fig. 3: Node State Diagram

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS203-2
February 1996

NMT Protocol Specification

1. SCOPE

This document contains the protocol specification of the Network Management (NMT). NMT is part of the CAN Application Layer. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS203-1, NMT Service Specification

/3/: CiA/DS207, Application Layer Naming Conventions

3. GENERAL DESCRIPTION

3.1 NMT Protocol Perspective

The Network Management (NMT) service element in the CAN Reference Model (see /1/), provides the NMT services. The NMT Protocol is executed between between the NMT Master and each of the NMT Slaves (see /2/) to implement these services.

3.2 NMT Slave Synchronization

Since in the NMT Protocol all NMT Slaves use the same COB to send information to the NMT Master, there must be only one NMT Slave at a time that communicates with the NMT Master. For all protocols except the Identify Node protocol, the NMT Master takes the initiative. The Identify Node protocol does not transfer information and will therefore cause no synchronization conflicts. For all other services an NMT Slave is only allowed to respond if it has first been addressed by the NMT Master through its unique NMT Address or Node-ID. Since there can be atmost one confirmed NMT service outstanding at a time (see /2/), the synchronization is established.

3.3 NMT Protocol Descriptions

A protocol description specifies the sequence of COB's and their format that are exchanged between the NMT Master and NMT Slave(s) for a particular NMT service. In the description of the COB data format, bytes are numbered from 0 to and including 7. Bits within a byte are numbered from 0 to and including 7. Byte 0 is transmitted first, byte 7 is transmitted last. Within a byte, bit 0 is the least significant bit, bit 7 is the most significant bit. In the protocol descriptions [a, b] denotes the range of integers from a to b with a and b included. If a > b, the range is empty.

The terms 'lsb' and 'msb' stand for 'least significant byte' and 'most significant byte' respectively and are used to define how an integer number is stored in more than one byte for the NMT Protocol. The order of significance is from lsb to msb.

The NMT protocols transfer the NMT Address (both module-name and module-ID) of an NMT Slave. The transfer syntax of these attributes is defined in /3/.

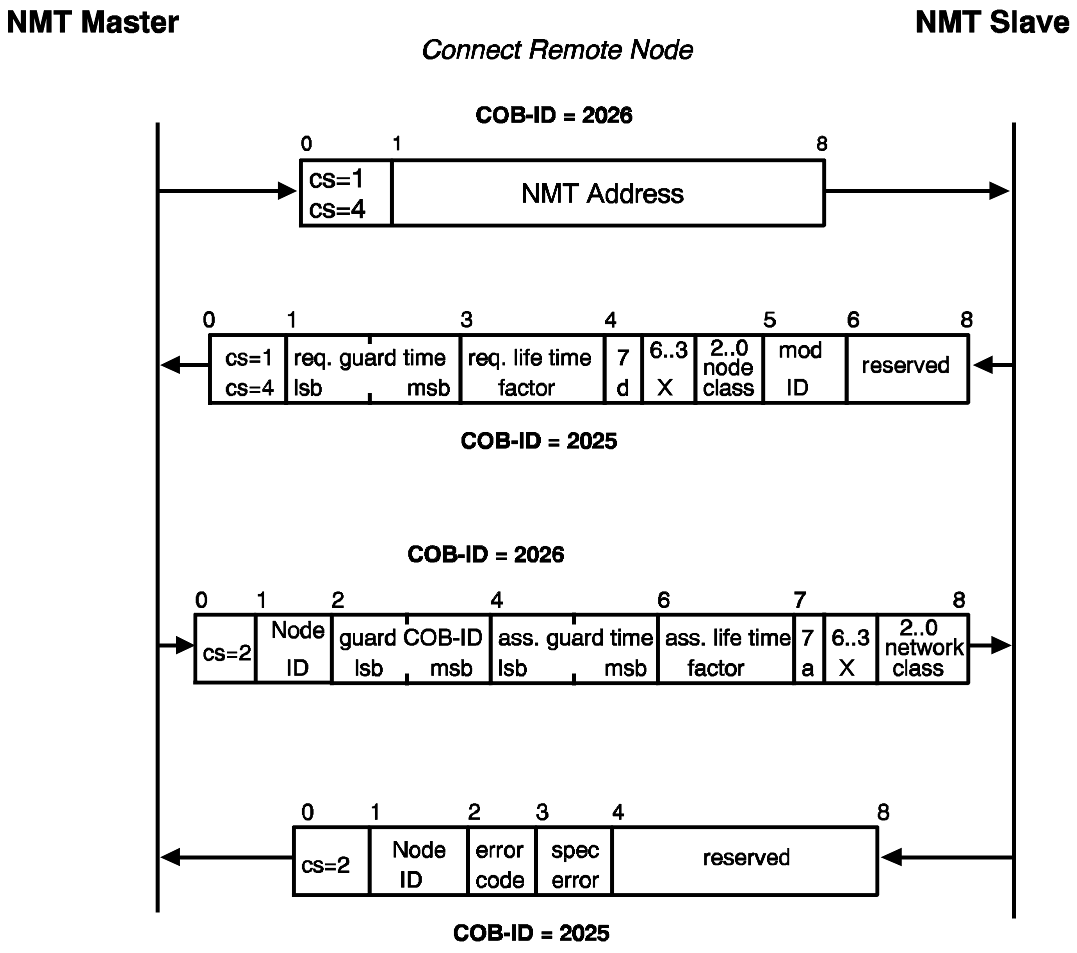
3.4 Usage of Command Specifiers

The command specifier is transmitted within the first data byte of the NMT protocol in COB-ID=2026/2025. For each service a unique value is specified for the command specifier. Unused values are reserved up to 191 for further use by CiA. Values from 192 to 255 are user specific.

4. MODULE CONTROL PROTOCOLS

4.1 Node Connect Protocol

This protocol is used to implement the 'Connect Remote Node' service, see /2/. This protocol also serves to exchange and negotiate parameters for the other NMT protocols.



- **cs:** NMT command specifier.
  1: select an NMT Slave by the module-name of its NMT-Address. Bytes [1, 7] contain the module name, see /3/. The selected NMT Slave responds with the same cs in case of a positive response.
  2: assign NMT protocol parameters
  4: select an NMT Slave by the module-ID of its NMT-Address. Byte 1 contains the module-ID, see /3/. Bytes [2, 7] are reserved for further use by CiA. The selected NMT Slave responds with the same cs in case of a positive response.

- **mod-ID:** the module-ID of the NMT Address of the NMT Slave, see /3/.

- **req. guard time:** the guard time in milli-seconds for the Node Guarding Protocol as requested by the NMT Slave. It is valid if and only if the node class indicates that the Node Error capability has been configured on the NMT Slave, see /2/, otherwise it is reserved for further use by CiA.

- **req. life time factor:** when multiplied with the requested guard time gives the life time for the Node Guarding Protocol. It is valid if and only if the node class indicates that the Node Error capability has been configured on the NMT Slave, see /2/, otherwise it is reserved for further use by CiA.

- **node class:** indicates the node capabilities that have been configured on the NMT Slave according to the definition in /2/.

- **d:** indicates whether or not the NMT Slave needs a configuration to be downloaded
  0: no download requested
  1: download requested

- **Node-ID:** the value of the Node-ID attribute that the NMT Master assigns to the NMT Slave, see /2/. The Node-ID is equal to the module-ID which is passed to the NMT-Master with the NMT-Address attribute when the corresponding remote node object is created.

- **guard COB-ID:** the value of the COB-ID for the Node Guarding Protocol. It must be a value between 1761 and 2015 inclusive. It is valid if and only if the network class indicates that the Network Error capability has been configured on the NMT Master, see /2/, otherwise it is reserved for further use by CiA.

- **ass. guard time:** the guard time in milli-seconds for the Node Guarding Protocol as assigned by the NMT Master. It is valid if and only if the network class indicates that the Network Error capability has been confirgured on the NMT Master, see /2/, otherwise it is reserved for further use by CiA.

- **ass. life time factor:** when multiplied with the assigned guard time gives the life time for the Node Guarding Protocol as assigned by the NMT Master. It is valid if and only if the network class indicates that the Network Error capability has been confirgured on the NMT Master, see /2/, otherwise it is reserved for further use by CiA.

- **network class:** indicates the network capabilities that have been configured on the NMT Master according to the definition in /2/.

- **a**: abortion flag used by the NMT master for signalling an abort of the Connect Remote Node Protocol if the NMT master detects a protocol inconsistency during the first request-response cycle; the NMT slave has to respond with error code 253.

0:	second request valid
1:	abort Connect Remote Node request

- **error code:**

0:	protocol successfully completed
1..252:	reserved for further use by CiA
253:	protocol error indication by NMT master
254:	request not allowed by the node state of NMT Slave
255:	other error occurred

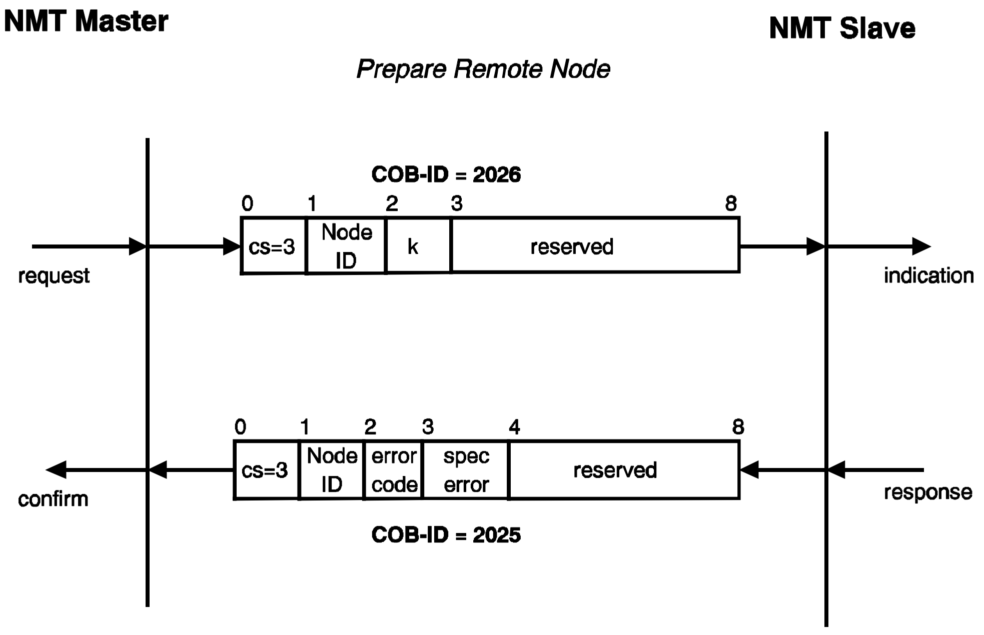
- **specific error code:** a value between 0 and 255 inclusive. If the error code equals 255, it gives an implementation specific error code, otherwise it is reserved for further use by CiA.

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA.

4.2 Node Prepare Protocol

This protocol is used to implement the 'Prepare Remote Node' service, see /2/.



- **cs:** NMT command specifier

3:	prepare

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol.

- **k:** indicates whether the COB-ID's previously obtained from the DBT must be discarded or not

0:	discard previous obtained identifiers
1:	the NMT Slave may decide to keep the old identifiers

- **error code:**

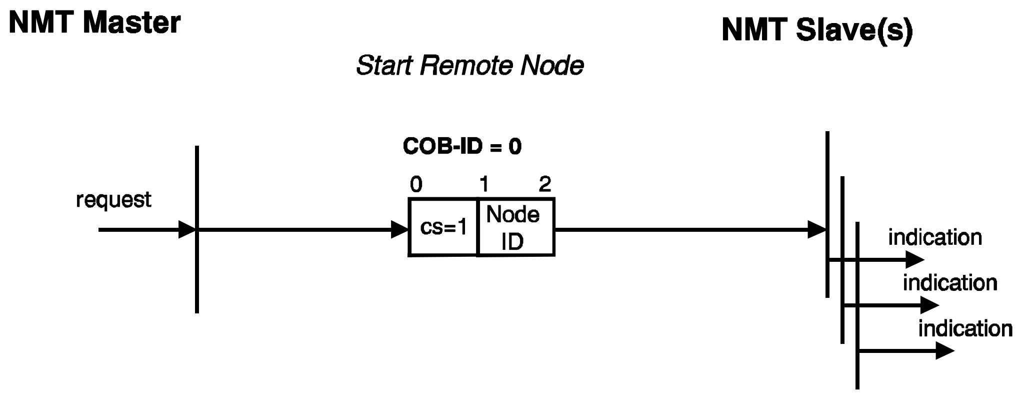
0:	Prepare Protocol completed successfully
1:	DBT protocol error occurred
2:	DBT Master is not available
3..253:	reserved for further use by CiA
254:	request not allowed by the node state of the NMT Slave
255:	other error occurred

- **specific error code:** a value between 0 and 255 inclusive. If the error code equals 1 it gives the error code of the DBT protocol. If error code equals 255, it gives an implementation specific error code. Otherwise it is reserved for further use by CiA.

- **reserved:** reserved for further use by CiA.

4.3 Node Start Protocol

This protocol is used to implement the 'Start Remote Node' service, see /2/.

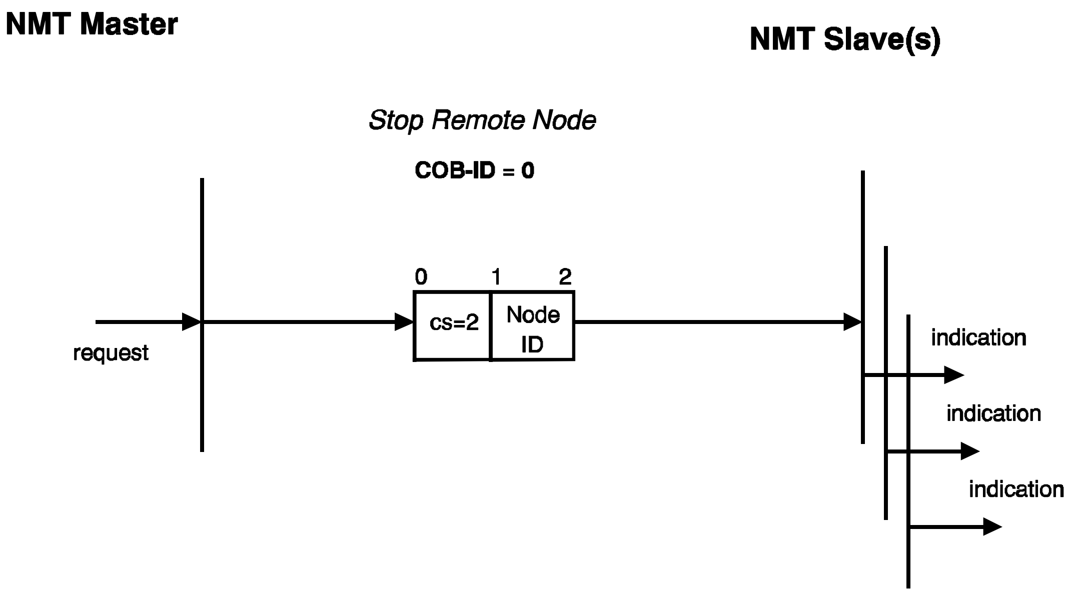


- **cs:** NMT command specifier
  1: start

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol, or 0. If 0, the protocol addresses all NMT Slaves.

4.4 Node Stop Protocol

This protocol is used to implement the 'Stop Remote Node' service, see /2/.

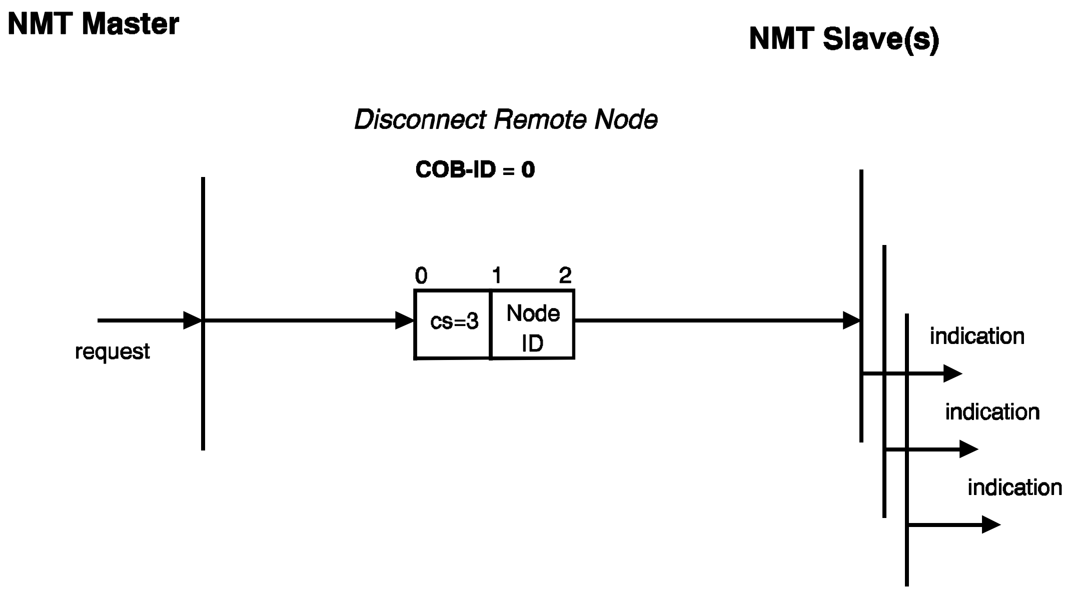


- **cs:** NMT command specifier
  2: stop

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol, or 0. If 0, the protocol addresses all NMT Slaves.

4.5 Node Disconnect Protocol

This protocol is used to implement the 'Disconnect Remote Node' service, see /2/.

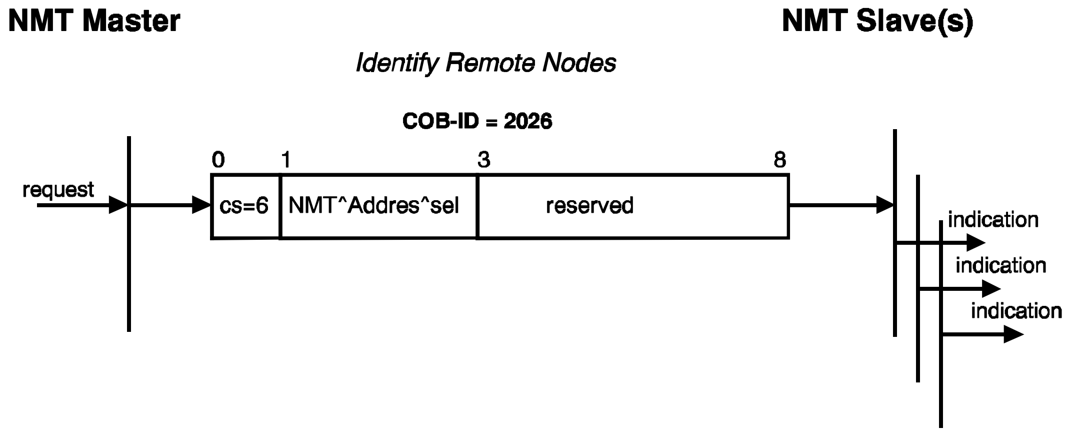


- **cs:** NMT command specifier
  3: disconnect

- Node-ID: the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol, or 0. If 0, the protocol addresses all NMT Slaves.

4.6 Identify Remote Nodes Protocol

This protocol is used to implement the 'Identify Remote Nodes' service, see /2/.



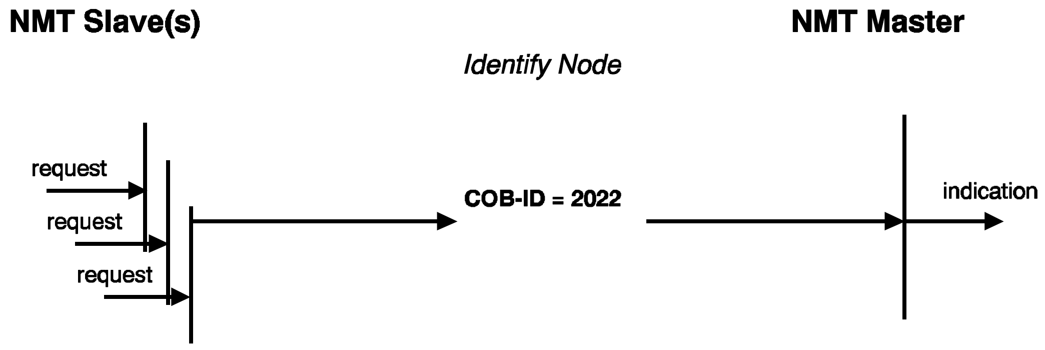
- **cs:** NMT command specifier
  6: identify

- **NMT_Address_sel:** selects a range of module-ID's. Byte 1 contains the lower boundary. Byte 2 contains the upper boundary. The boundaries are included in the range. All NMT Slaves whose NMT Address has a module-ID that lies within this range, are requested to identify themselves, see /2/.

- **reserved:** reserved for further use by CiA.

4.7 Identify Node Protocol

This protocol is used to implement the 'Identify Node' service, see /2/.

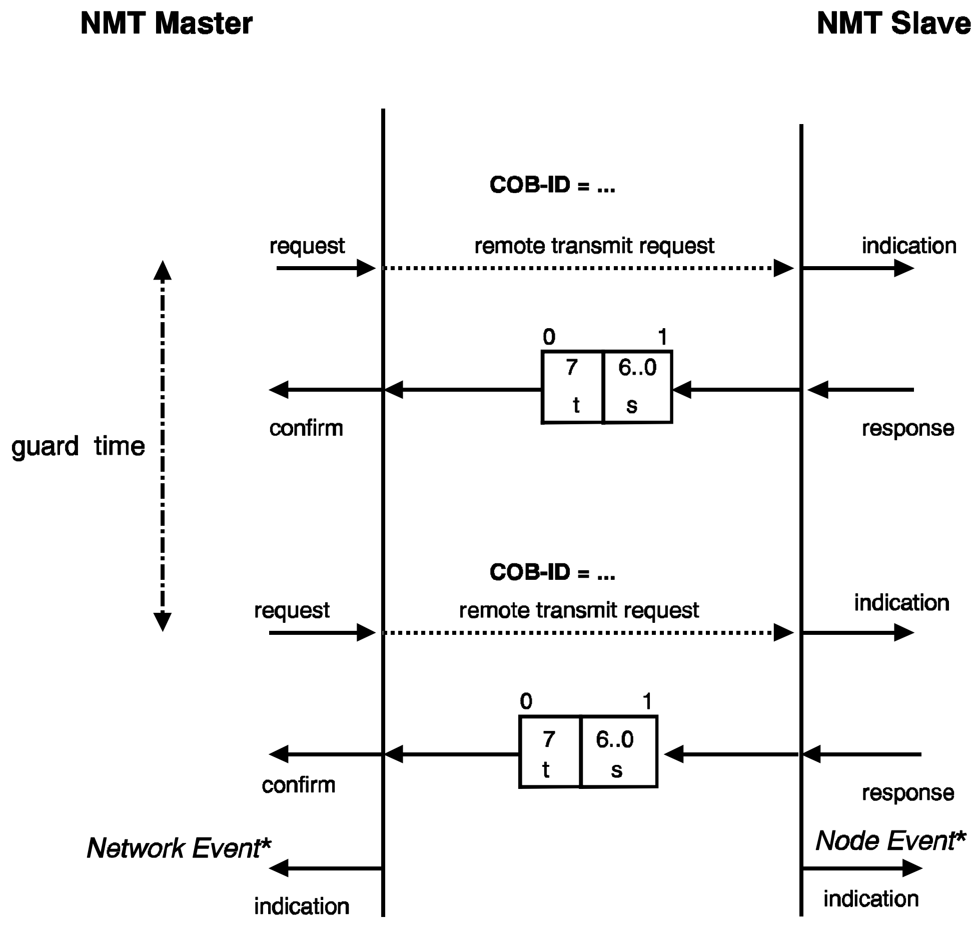


- NOTE: there is no protocol data to allow several NMT Slaves to execute this protocol at the same time. For this service, the NMT Slave takes the initiative.

5. ERROR CONTROL PROTOCOLS

5.1 Node Guarding Protocol

This protocol is used to detect remote errors in the network, see /2/. Each NMT Slave uses one remote COB for the Node Guarding Protocol. This protocol implements the provider initiated Error Control services. The state diagrams in /2/ determine when this protocol is active or inactive.



* if Node Guarding Protocol Error

The NMT Master polls each NMT Slave at regular time intervals. This time-interval is called the guard time and may be different for each NMT Slave. The response of the NMT Slave contains the state of the node object of that NMT Slave, see /2/. If this state does not match the state of the corresponding remote node object or if no response is received by the NMT Master, it will be indicated that a remote error has occurred through the 'Network

Event' service, see /2/. If the NMT Slave hasn't been polled during its life-time, it will be indicated that a remote error has occurred through the 'Node Event' service, see /2/. The life-time can be different for each NMT Slave. If it has been indicated that a remote error has occurred and the errors in the guarding protocol have disappeared, it will be indicated that the remote error has been resolved through the 'Network Event' and 'Node Event' services, see /2/.

The guard time, the life time, and the COB-ID for the Node Guarding Protocol are negotiated between the NMT Master and each NMT Slave in the Node Connect Protocol.

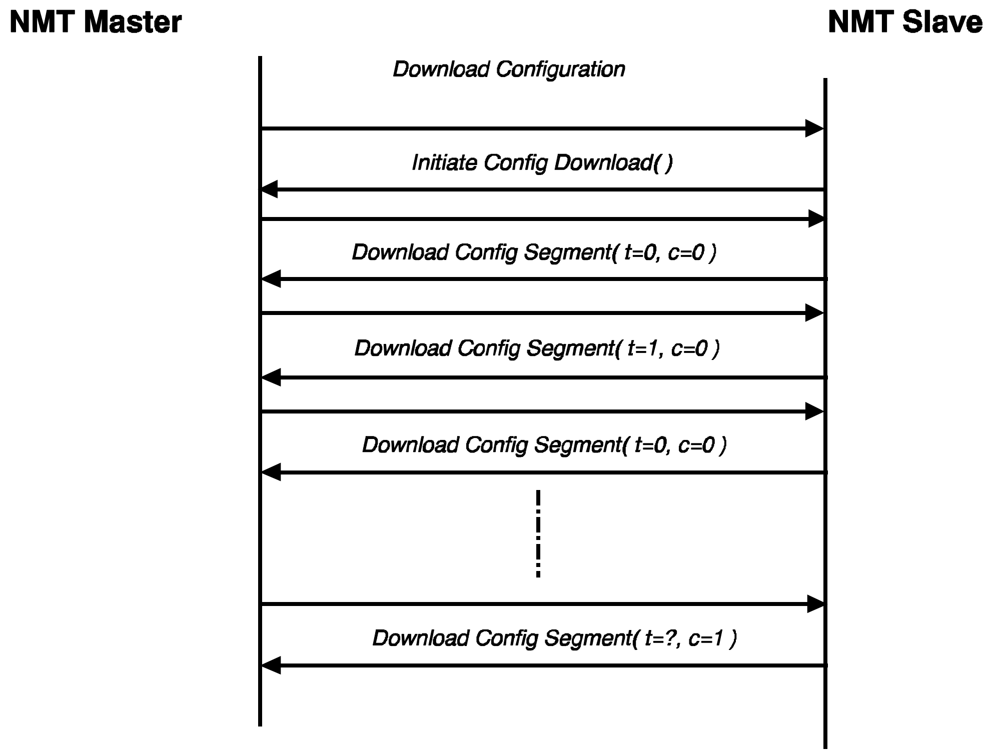
- s: the state of the node object on the NMT Slave

1:	DISCONNECTED
2:	CONNECTING
3:	PREPARING
4:	PREPARED
5:	OPERATIONAL

- t: toggle bit. The value of this bit must alternate between two consecutive responses from the NMT Slave. If it does not alter, it will be indicated that a remote error has occurred through the 'Network Event' service, see /2/. The value of the toggle-bit of the first reponse after the Guarding Protocol becomes active, is 0.

6. CONFIGURATION CONTROL PROTOCOLS

6.1 Download Configuration Protocol


NMT Master
NMT Slave
Download Configuration
Initiate Config Download( )
Download Config Segment(t=0, c=0)
Download Config Segment(t=1, c=0)
Download Config Segment(t=0, c=0)
Download Config Segment(t=?, c=1)

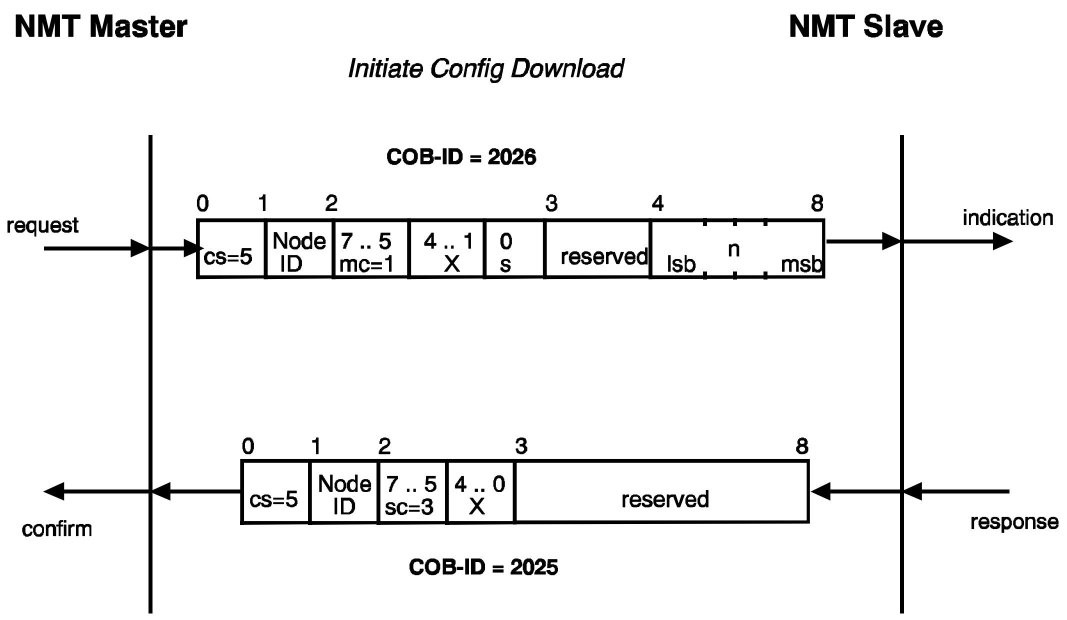
This protocol is used to implement the 'Configuration Download' service, see /2/. Configurations are downloaded as a sequence of 'Download Configuration Segment' services preceded by an 'Initiate Configuration Download' service. The sequence can be terminated by:

- a 'Download Configuration Segment' response/confirm with the c-bit set to 1, confirming the succesful completion of the download sequence.
- an 'Abort Configuration Transfer' request/indication indicating the unsuccesful completion of the download sequence.
- a new 'Initiate Configuration Download' request/indication indicating the unsuccesful completion of the download sequence and starting a new sequence.

If in the download of two consecutive segments the toggle bit does not alter, this must be treated as if an invalid COB was received (see Annex I). If such an error is reported to the application, the application may decide to abort the download.

Initiate Configuration Download Protocol

This protocol is used to implement the 'Initiate Configuration Download' service, see /2/.



- **cs:** NMT command specifier
  5: Transfer Configuration

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol.

- **mc:** NMT Master command
  1: initiate download request

- **sc:** NMT Slave command
  3: initiate download response

- **s:** size indicator
  0: configuration size is not indicated
  1: configuration size is indicated

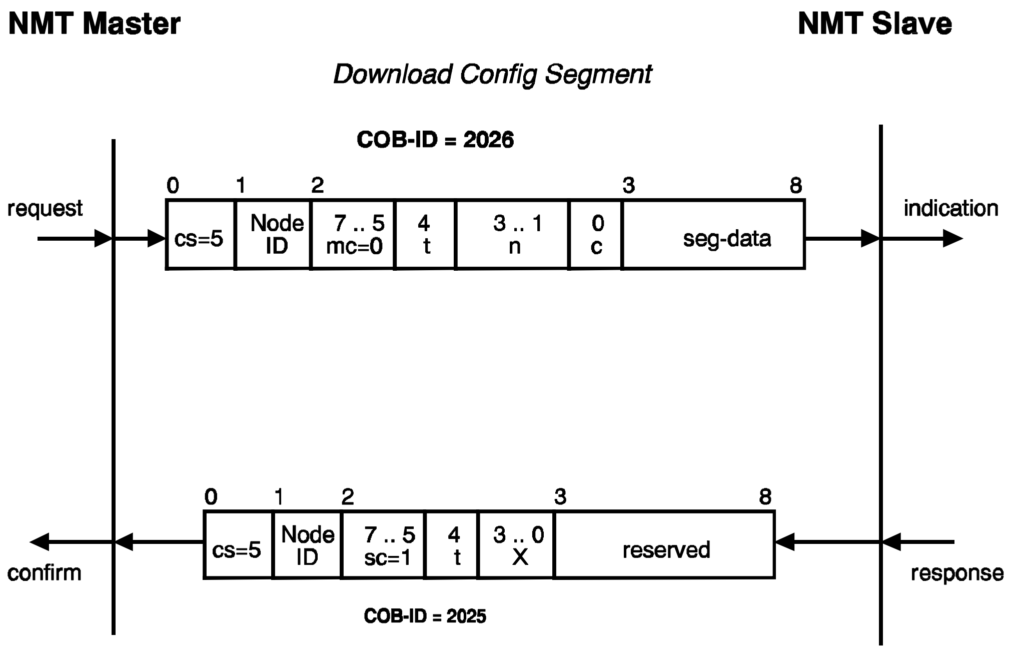
- **n:** only valid if s = 1, otherwise reserved for further use by CiA. If valid it contains the number of bytes to be downloaded

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA.

Download Configuration Segment Protocol

This protocol is used to implement the 'Download Configuration Segment' service, see /2/.



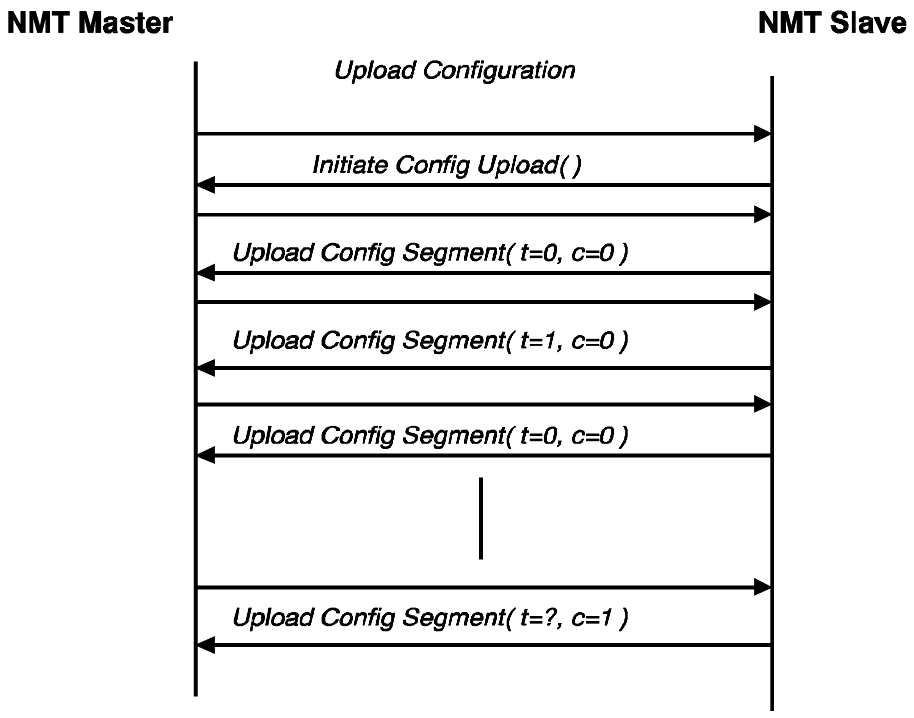
- **cs:** NMT command specifier
  5: Transfer Configuration
- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol.
- **mc:** NMT Master command
  0: download segment request
- **sc:** NMT Slave command
  1: download segment response
- **c:** indicates whether there are still more segments to be downloaded.
  0: more segments to be downloaded
  1: no more segments to be downloaded
- **seg-data:** contains at most five bytes of segment data to be downloaded. Its meaning has to be specified by the application.
- **n:** contains the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data.

- **t:** toggle bit. This bit must alternate for each subsequent segment that is downloaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA.

6.2 Upload Configuration Protocol


NMT Master
NMT Slave
Upload Configuration
Initiate Config Upload( )
Upload Config Segment(t=0, c=0)
Upload Config Segment(t=1, c=0)
Upload Config Segment(t=0, c=0)
Upload Config Segment(t=?, c=1)

This protocol is used to implement the 'Configuration Upload' service, see /2/. Configurations are uploaded as a sequence of 'Upload Configuration Segment' services preceded by an 'Initiate Configuration Upload' service. The sequence can be terminated by:

- an 'Upload Configuration Segment' response/confirm with the c-bit set to 1, indicating the succesful completion of the upload sequence.

- an 'Abort Configuration Transfer' request/indication indicating the unsuccess ful completion of the upload sequence.

- a new 'Initiate Configuration Upload' request/indication indicating the unsuc cesful completion of the upload sequence and starting a new sequence.

If in the upload of two consecutive segments the toggle bit does not alter, this must be treated as if an invalid COB was received (see Annex I). If such an error is reported to the application, the application may decide to abort the upload.

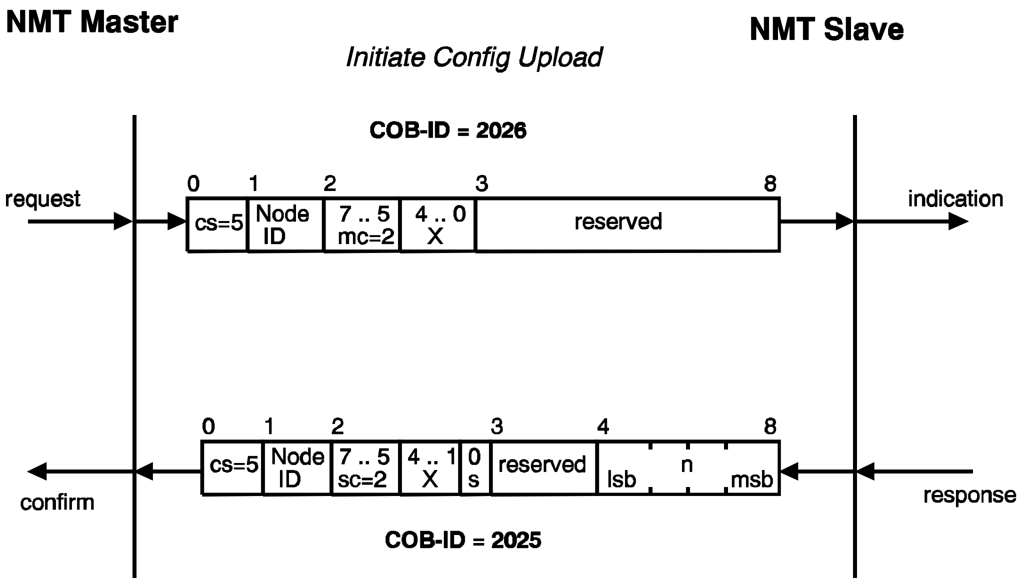
Initiate Configuration Upload Protocol

This protocol is used to implement the 'Initiate Configuration Upload' service, see /2/.

- **cs:** NMT command specifier
  5: Transfer Configuration

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol.

- **mc:** NMT Master command
  2: initiate upload request



- **sc:** NMT Slave command
  2: initiate upload response

- **s:** size indicator
  0: configuration size is not indicated
  1: configuration size is indicated

- **n:** Only valid if s = 1, otherwise reserved for further use by CiA. If valid it contains the number of bytes to be uploaded

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA.

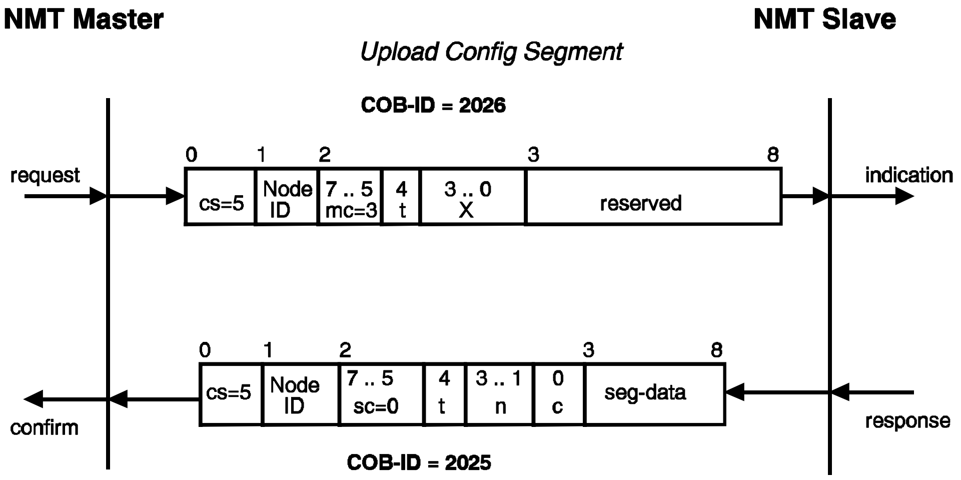
Upload Configuration Segment Protocol

This protocol is used to implement the 'Upload Configuration Segment' service, see /2/.

- **cs:** NMT command specifier
  5: Transfer Configuration

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol.

- **mc:** NMT Master command
  3: upload segment request



- **sc:** NMT Slave command
  0: upload segment response

- **c:** indicates whether there are still more segments to be uploaded.
  0 : more segments to be uploaded
  1 : no more segments to be uploaded

- **seg-data:** contains at most five bytes of segment data to be uploaded. Its meaning has to be specified by the application.

- **n:** contains the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data.

- **t:** toggle bit. This bit must alternate for each subsequent segment that is uploaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

- **X:** not used, always 0

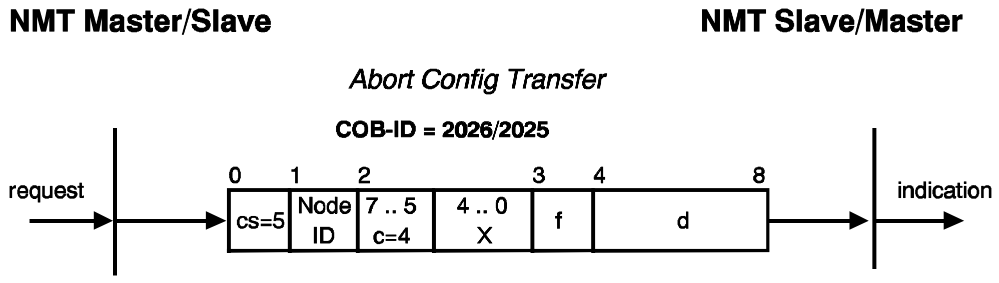
- **reserved:** reserved for further use by CiA.

6.3 Abort Configuration Transfer Protocol

This protocol is used to implement the 'Abort Configuration Transfer' service, see /2/. The NMT Master sends COB-ID 2026. The NMT Slave sends COB-ID 2025.

- **cs:** NMT command specifier
  5: Transfer Configuration

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol.



- **c:** command
  4: abort configuration transfer request

- **f:** indicates the reason for the failure.

0:	unspecified error
1:	application request
2:	no resources
3..127:	reserved for further use by CiA
128..255:	implementation specific error codes

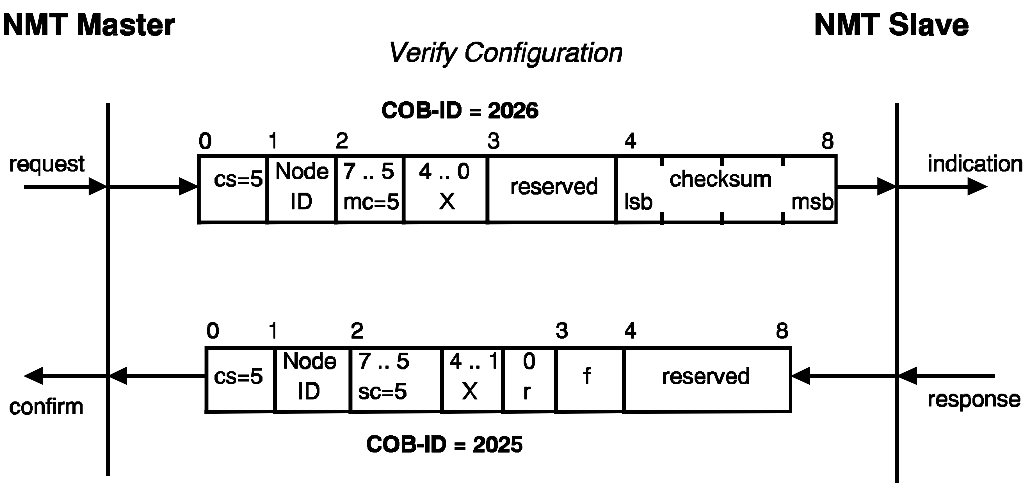
- **d:** only valid if f = 1 or f > 128, otherwise reserved for further use by CiA. If valid it contains application specific information about the reason for the abort.

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA.

6.4 Verify Configuration Protocol

This protocol is used to implement the 'Verify Configuration' service, see /2/.



- **cs:** NMT command specifier
  5: Transfer Configuration

- **Node-ID:** the Node-ID of the NMT Slave as assigned by the NMT Master in the Node Connect Protocol.

- **mc:** NMT Master command
  5: verify configuration

- **sc:** NMT Slave command
  5: verify configuration

- **checksum:** the check-sum to be verified

- **r:** result
  0: verification successful
  1: verification failed

- **f:** Only valid if r = 1. If valid it indicates the reason for the failure.

0:	unspecified error
1:	checksum mismatch
2:	no configuration present
3..127:	reserved for further use by CiA
128..255:	implementation specific error codes

- **X:** not used, always 0

- **reserved:** reserved for further use by CiA.

ANNEX I

IMPLEMENTATION RULES

When implementing the NMT protocols, the following rules have to be followed to guarantee interoperability. These rules deal with the following implementation aspects:

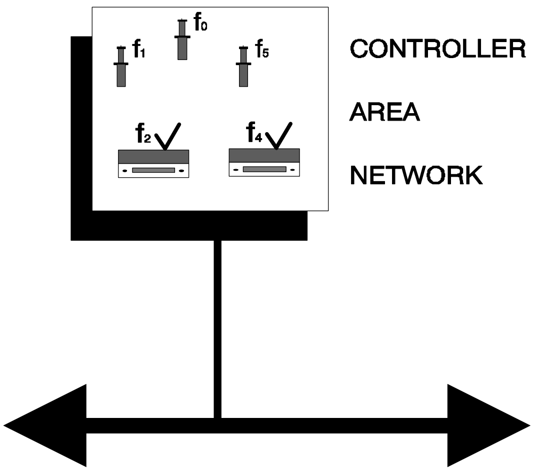
Invalid COB's

A COB is invalid if it has a COB-ID that is used by the NMT Protocol, but it contains invalid parameter values according to the NMT Protocol. This can only be caused by errors in the lower layers (see /1/) or implementation errors. Invalid COB's must be handled locally in an implementation specific way that does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. As far as the NMT Protocol is concerned, an invalid COB must be ignored.

Time-out's

Since COB's may be ignored, the response of a confirmed NMT service may never arrive. To resolve this situation, an implementation may, after a certain amount of time, indicate this to the service user (time-out). A time-out is not a confirm of that NMT service. A time-out indicates that the service has not completed yet. The application must deal with this situation. Time-out values are considered to be implementation specific and do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. However, it is recommended that an implementation provides facilities to adjust these time-out values to the requirements of the application.

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS204-1
February 1996

DBT Service Specification

1. SCOPE

This document contains the Distributor Service Specification. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS204-2, DBT Protocol Specification

/3/: CiA/DS207, Application Layer Naming Conventions

/4/: CiA/DS203-1, NMT Service Specification

/5/: CiA/DS202-1, CMS Service Specification

3. GENERAL DESCRIPTION

3.1 DBT PERSPECTIVE

The essential issue in creating an *open system* where modules from independent suppliers can cooperate via CAN, is how the *identifiers and inhibit-times* are assigned to the COB's that a module uses. Identifiers are used by the CAN Datalink Layer and inhibit-times are defined by the CMS service element of the CAN Application Layer, see /5/. The identifiers and inhibit-times must be distributed in a way that:

- prevents a *conflict* (i.e different functions that use the same identifiers).
- prevents a *mismatch* (i.e different identifiers for the same COB).
- offers the *system integrator* control of the *dynamic behaviour* of the system since the identifier and inhibit-time of a COB determines its priority respectively its maximum access time to the CAN bus.

Three methods can be distinguished to distribute identifiers and inhibit-times to a module (**Note:** it is not required that the same method is used for all modules although this increases the probability of clashes and conflicts):

- If a **standard distribution** is used, the identifiers and inhibit-times are standardized by the module suppliers and system integrators and cannot be changed. A standard distribution requires standardizing all functions and their corresponding identifiers and can only succeed if sufficient identifiers are available and if the application has a limited scope (e.g one system or a specific application type).

- If a **static distribution** is used, the identifiers and inhibit-times are fixed by the module suppliers and may be changed by the system integrator through module specific measures such as setting dip-switches, adapting firmware, etc. A static distribution requires that the system integrator can assign all possible identifiers.

- If a **dynamic distribution** is used, the identifiers and inhibit-times are distributed *via the CAN network* through standard services and a protocol.

The DBT is a service element of the CAN Application Layer (see /1/) that offers dynamic distribution of identifiers and inhibit-times to the COB's that a module uses. The dynamic distribution does not necessarily take place every time the module is 'powered on'. Depending on the facilities of the module, distribution may only be required once e.g when the module is installed in the network.

3.2 DBT Objects and Services

The CMS service element defines for each CMS object the COB's that must be used for the protocols of that object. Each module in the network that acts as a Client or Server of a CMS object must either transmit or receive these COB's and is called a *user* of these COB's. A user is uniquely identified in the network via its *Node-ID*, see /4/.

The DBT uses four objects to model its functionality:

- **the COB Database.** The COB Database contains zero or more COB Definitions. The COB Database may exist on one module only, called the *DBT Master*.

- **a COB Definition.** A COB Definition defines all attributes of a COB. A COB Definition is uniquely identified in the COB Database by its identifier (COB-ID). A COB Definition is created by the DBT Master. For each user a COB Definition contains a User Definition created by that user. A COB definition can optionally contain Predefinitions created by the DBT Master.

- **a User Definition.** A User Definition defines how one module uses that COB. A User Definition is uniquely identified in the COB Database by the *name* of the COB and the Node-ID of that user (**Note:** different users may use different COB names for the same COB). The syntax of COB-names and Node-ID's are defined in /3/.

- **a Predefinition.** A Predefinition defines that all User Definitions with a certain COB-name must use the COB-ID of the COB Definition to which it belongs. A Predefinition is uniquely identified in the COB Database by the name of the COB.

The contents of the COB Database can be manipulated locally by the DBT Master or remotely (possibly via the network) by a *DBT Slave*. A DBT Slave communicates with the DBT Master via the DBT Protocol as depicted in Fig. 1. Note that it is possible that a module is a DBT Master and a DBT Slave at the same time. The DBT Protocol is specified in /2/.

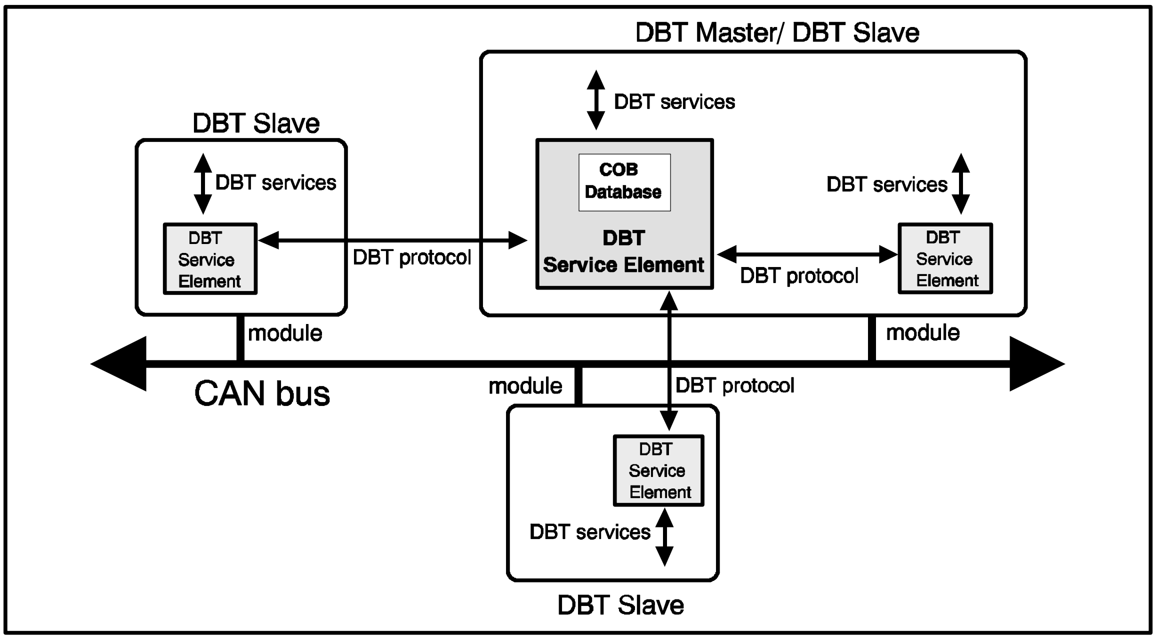


Fig. 1: The DBT Model

The DBT offers the following categories of services:

- **Distribution Control Services:** it is the task of the DBT Master to distribute a COB-ID and inhibit-time for a COB to all its users. The COB-ID and inhibit-time that the DBT Master distributes is determined by the requested values from the DBT Slave and *predefined values* from the DBT Master. The DBT Master can also prevent that a value will be distributed as a COB-ID and enforce a minimum inhibit-time for a certain COB-ID.

- **Consistency Control Services:** through these services, a DBT Slave can detect inconsistencies in the COB Database and inconsistencies between User Definitions created by different users.

3.3 DBT Slave Capabilities

DBT slave capabilities indicate categories of DBT functionality that may or may not be present in the DBT Slave.The following capabilities are defined:

- **Distribution capability**. This capability implements the mandatory distribution control services on a DBT Slave.

- **Consistency capability**. This capability implements the mandatory consistency control services on a DBT Slave.

How to configure DBT slave capabilities on a DBT Slave does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

3.4 DBT Slave Classes

The DBT slave class indicates the capabilities that have been configured on a DBT Slave:

- **Class 0:** no Distribution capability. As a consequence, consistency control, and dynamic distribution of identifiers and inhibit-times is not possible for this module.

- **Class 1:** Distribution capability, no Consistency capability. This is a module for which dynamic distribution of identifiers and inhibit-times is possible, but consistency control is not possible.

- **Class 2:** Distribution capability, Consistency capability. This is a module for which dynamic distribution of identifiers and inhibit-times and consistency control is possible.

3.5 DBT Service Descriptions

The DBT services are described in a tabular form that contains the parameters of each service primitive that is defined for that service. The primitives that are defined for a particular service determine the service type (e.g unconfirmed, confirmed, etc.). How to interprete the tabular form and what service types exist is defined in /1/. In the service descriptions, [a, b] denotes the range of integers from a to b with a and b included. If a > b, the range is empty.

All services assume that no errors occur in the Data Link and Physical Layer. These errors are resolved by the Network Management Service Element, see /1/.

4. DBT OBJECTS

All the DBT Objects, their attributes, and their relation are drawn in Fig. 2.

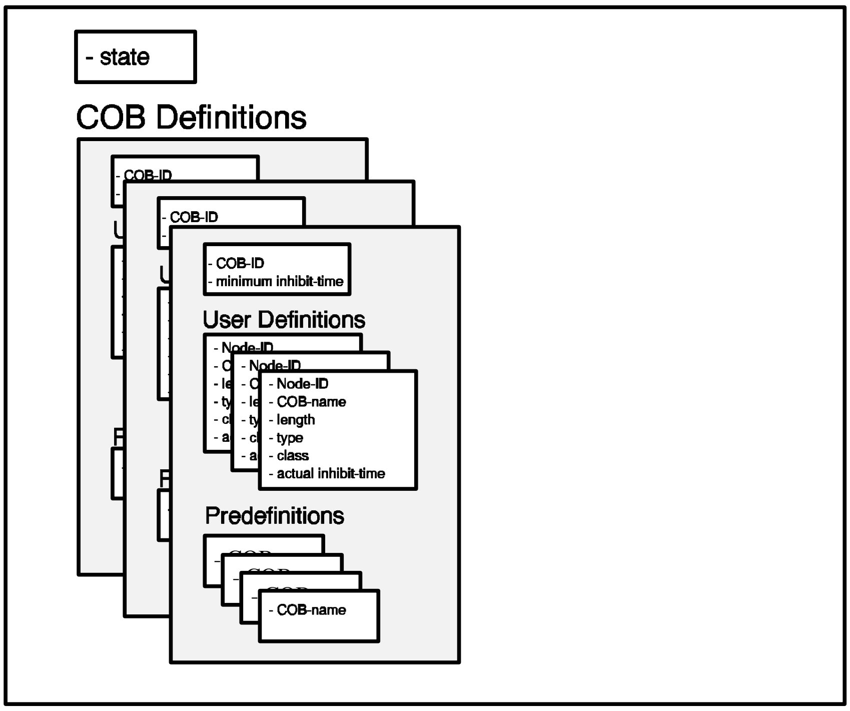


Fig. 2: The COB Database

4.1 COB Database

Attributes:

- **state:** one of the values {ENABLED, DISABLED}. This attribute indicates whether or not the DBT Master is capable of distributing COB-ID's and inhibit-times consistently for the COB's used by the CMS protocol.

- **COB definition set:** The set of all COB Definitions.

4.2 COB Definition

Attributes

- **COB-ID**: a value in the range [1, 1760], indicating the COB-ID of the COB.

- **minimum inhibit-time:** a value in the range [0, 65535] indicating the minimum value in units of 100 µsec for the inhibit-time that must be used by a user of the COB.

- **user definition set:** the set of User Definitons of this COB.
- **predefinition set:** the set of Predefinitons of this COB.

4.3 User Definition

Attributes

- **Node-ID:** see /4/. It identifies the user that created the User Definition.
- **COB-name:** see /3/. The name of the COB as used by the user that created the User Definition.
- **COB-length:** a value in the range [0, 8]. The COB-length indicates the number of data bytes of the COB as used by the user that created the User Definition.
- **COB-type:** one of the values {RECEIVE, TRANSMIT}. The COB-type indicates whether the COB is received (RECEIVE) or transmitted (TRANSMIT) by the user that created the User definition (**Note:** for a Remote-COB, RECEIVE indicates transmitting the request for the COB and receiving the data. TRANSMIT indicates that the request is received and the data transmitted).
- **COB-class:** see /5/. The COB-class relates the number of users that transmit and receive the COB as expected by the user that created the User Definition.
- **actual inhibit-time:** a value in the range [0, 65535]. It indicates the value of the inhibit-time in units of 100 μsec as actually used by the user that created the User Definition.

4.4 Predefinition

Attributes

- **COB-name:** see /3/. All User Definitions for a COB with this COB-name must be created in the user set of the COB Definition to which the Predefinition belongs.

5. DBT SERVICES

There can be atmost one confirmed DBT service outstanding in the complete network.

5.1 Distribution Control Services

The mandatory distribution control services need to be implemented on the DBT Master. The mandatory distribution control services need to be implemented on a DBT Slave if and only if the Distribution capability has been configured on that DBT Slave.

Create COB Database

Parameter	*Request*
Argument	**Mandatory**

Through this service, the DBT Master creates a COB Database. No COB Database may exist. After completion of the service the state of the COB Database will be ENABLED and it will contain no COB Definitions. The service is local and mandatory.

Enable Distribution

Parameter	*Request*
Argument	**Mandatory**

Through this service, the DBT Master sets the state of the COB Database to ENABLED. What conditions can cause this service to be invoked is determined by the DBT Master. The service is local and mandatory.

Disable Distribution

Parameter	*Request*
Argument	**Mandatory**

Through this service, the DBT Master sets the state of the COB Database to DISABLED. What conditions can cause this service to be invoked is determined by the DBT Master. The service is local and mandatory.

Create COB Definition

Parameter	*Request*
Argument	**Mandatory**
range	mandatory
low COB-ID	mandatory
high COB-ID	mandatory
min. inhibit-time	mandatory

Through this service, the DBT Master creates all COB Definitions in the COB Database with a COB-ID in the requested range, all with the same minimum inhibit-time. The state of the COB Database must be ENABLED. After completion of the service, the user and predefinition set of the created COB Definitions will be empty. The service is local and mandatory.

Delete COB Definition

Parmater	*Request*
Argument	
range	mandatory
low COB-ID	mandatory
high COB-ID	mandatory

Through this service, the DBT Master deletes all COB Definitions with a COB-ID in the requested range. The state of the COB Database must be ENABLED. The service is local and mandatory.

Create Predefinition

Through this service, the DBT Master creates a Predefinition with the requested attributes in the predefinition set of the COB Definition identified by COB-ID. No Predefinition and User Definition with the same COB-name may exist in the COB Database. The state of the COB Database must be ENABLED. The service is local and mandatory.

Parameter	*Request*
Argument	**Mandatory**
COB-ID	mandatory
COB-name	mandatory

Delete Predefinition

Parameter	*Request*
Argument	**Mandatory**
COB-name	mandatory

Through this service, the DBT Master deletes the Predefinition identified by COB-name from the COB Database. A Predefinition with the requested COB-name must exist in the predefinition set of a COB Definition in the COB Database. The state of the COB Database must be ENABLED. The service is local and mandatory.

Create User Definition

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
COB-name	mandatory	
COB-length	mandatory	
COB-class	mandatory	
COB-type	mandatory	
Node_ID	mandatory	
priority	mandatory	
inhibit-time	mandatory	
Remote Result		**Mandatory**
success		selection
COB-ID		mandatory
min. inhibit-time		mandatory
failure		selection
reason		optional

Through this service, a DBT Slave creates a new User Definition with the requested attributes in the user set of one of the COB Definitions in the COB Database. If there exists an *offending* COB Definition the service will fail. A COB Definition is offending if and only if the following conditions are met:

- its user set contains a User Definition or its predefinition set contains a Predefinition with the same COB-name
- its user set contains User Definitions with a different value for the COB-length and/or COB-class attributes

If there exists no offending COB Definition but there exists a *matching* COB definition, this definition will be selected. A COB Definition is matching if and only if at least one of the following conditions are met:

- its predefinition set contains a Predefinition with the same COB-name
- its user set contains a User Definition with the same COB-name

If there exists no offending and no matching COB Definition, a *free* COB definition will be selected according to the following rules given in order of precedence:

- a COB definition with an empty user set and an empty predefinition set whose COB-ID corresponds to the requested priority, see the table in annex I of this document.
- a COB definition with an empty user set and an empty predefinition set whose COB-ID exceeds the COB-ID's that correspond to the requested priority, see the table in annex I of this document.

If there exists no offending, no matching, and no free COB Definition, the service will fail.

The service is confirmed and mandatory. The state of the COB Database must be ENABLED. The Remote Result parameter will indicate the success or failure of the selection of a COB Definition. In case of success, the COB-ID of the selected COB Definition and its minimum inhibit-time attribute will be confirmed. The DBT Slave must use the maximum of the requested inhibit-time and the minimum inhibit-time attribute of the selected COB Definition. In case of a failure optionally the reason may be confirmed.

Delete User Definition

Through this service a DBT Slave deletes all User Definitions that were created by the user identified by Node_ID or all existing User Definitions from the COB Database. The state of the COB Database must be ENABLED unless all existing User Definitions are to be deleted.

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
scope	mandatory	
Node_ID	selection	
all	selection	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

The service is confirmed and mandatory. The Remote Result parameter will indicate the success or failure of the request. In case of success, the requested User Definitions were deleted. If all User Definitions have been deleted the state of the COB Database will be ENABLED. In case of a failure, optionally the reason may be confirmed.

5.2 Consistency Control Services

The mandatory consistency control services need to be implemented on the DBT Master. The mandatory consistency control services need to be implemented on a DBT Slave if and only if the Consistency capability has been configured on that DBT Slave.

Verify COB Class

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
Remote Result		**Mandatory**
success		selection
failure		selection
COB-ID		optional
reason		optional

Through this service a DBT Slave requests the DBT Master to verify for each COB Definition in the COB Database, if the number of User Definitions in its user set whose type attribute is RECEIVE respectively TRANSMIT, matches the class attribute of that COB Definition. The state of the COB Database must be ENABLED.

The service is confirmed and mandatory. The Remote Result parameter will confirm the success or failure of the verification. In case of a failure, optionally the COB-ID of a COB Definition for which the verification failed and optionally a reason is confirmed.

Get Checksum

Parameter	*Request/Indication*	*Response/Confirm*
Argument	**Mandatory**	
scope	mandatory	
Node_ID	selection	
all	selection	
Remote Result		**Mandatory**
success		selection
checksum		mandatory
failure		selection
reason		optional

Through this service, a DBT slave requests the DBT Master to calculate a checksum. The value of the checksum depends on the requested scope and equals either:

- the remainder of the whole division by 8191 of the sum of the COB-ID's of all COB Definitions in the COB Database that have at least one User Definition
- the remainder of the whole division by 8191 of the sum of the COB-ID's of the COB Definitions whose user set contains a User Definition created by the user identified by Node_ID.

The service is confirmed and mandatory. The Remote Result parameter will confirm the success or failure of the verification. In case of failure, optionally the reason is confirmed.

ANNEX I

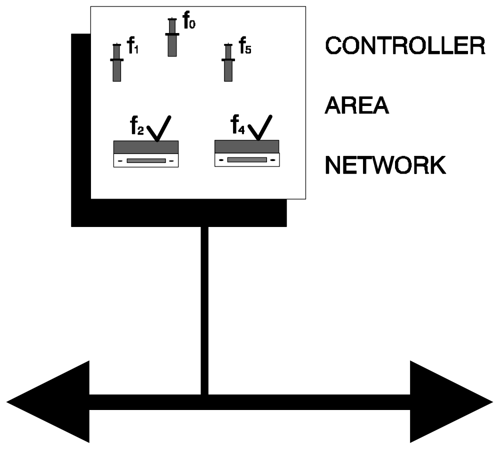
COB Identifier Table

The COB-ID table in Fig. 3 describes the usage of the COB identifiers by the protocols of the CAN Application Layer for Industrial Automation.

NMT start/stop services	0
CMS objects priority 0	1-220
. . . .	
CMS objects priority 7	1541-1760
NMT Node Guarding Protocol	1761-2015
reserved for further use by CiA	2016-2019
LMT Services	2020
LMT Services	2021
NMT Identify Node Protocol	2022
DBT services	2023
DBT services	2024
NMT services	2025
NMT services	2026
reserved for module self-test	2027
reserved for further use by CiA	2028-2031

Fig. 3: The COB-ID Table

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS204-2
February 1996

DBT Protocol Specification

1. SCOPE

This document contains the protocol specification of the Distributor (DBT). DBT is part of the CAN Application Layer. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS204-1, DBT Service Specification

/3/: CiA/DS207, Application Layer Naming Conventions

/4/: Robert Bosch GmbH, CAN Specification 2.0 Part B, September 1991

3. GENERAL DESCRIPTION

3.1 DBT Protocol Perspective

The Distributor (DBT) service element in the CAN Reference Model (see /1/), provides the DBT services. The DBT Protocol is executed between the DBT Master and each of the DBT Slaves (see /2/) to implement these services.

3.2 DBT Slave Synchronization

Since in the DBT Protocol all DBT Slaves use the same COB to send information to the DBT Master, there must be only one DBT Slave at a time that communicates with the DBT Master. This synchronization between the DBT Slaves is established by the NMT service element of the CAN Application Layer, see /1/.

3.3 DBT Protocol Descriptions

A protocol description specifies the sequence of COB's and their format that are exchanged between the DBT Master and DBT Slave for a particular DBT service.

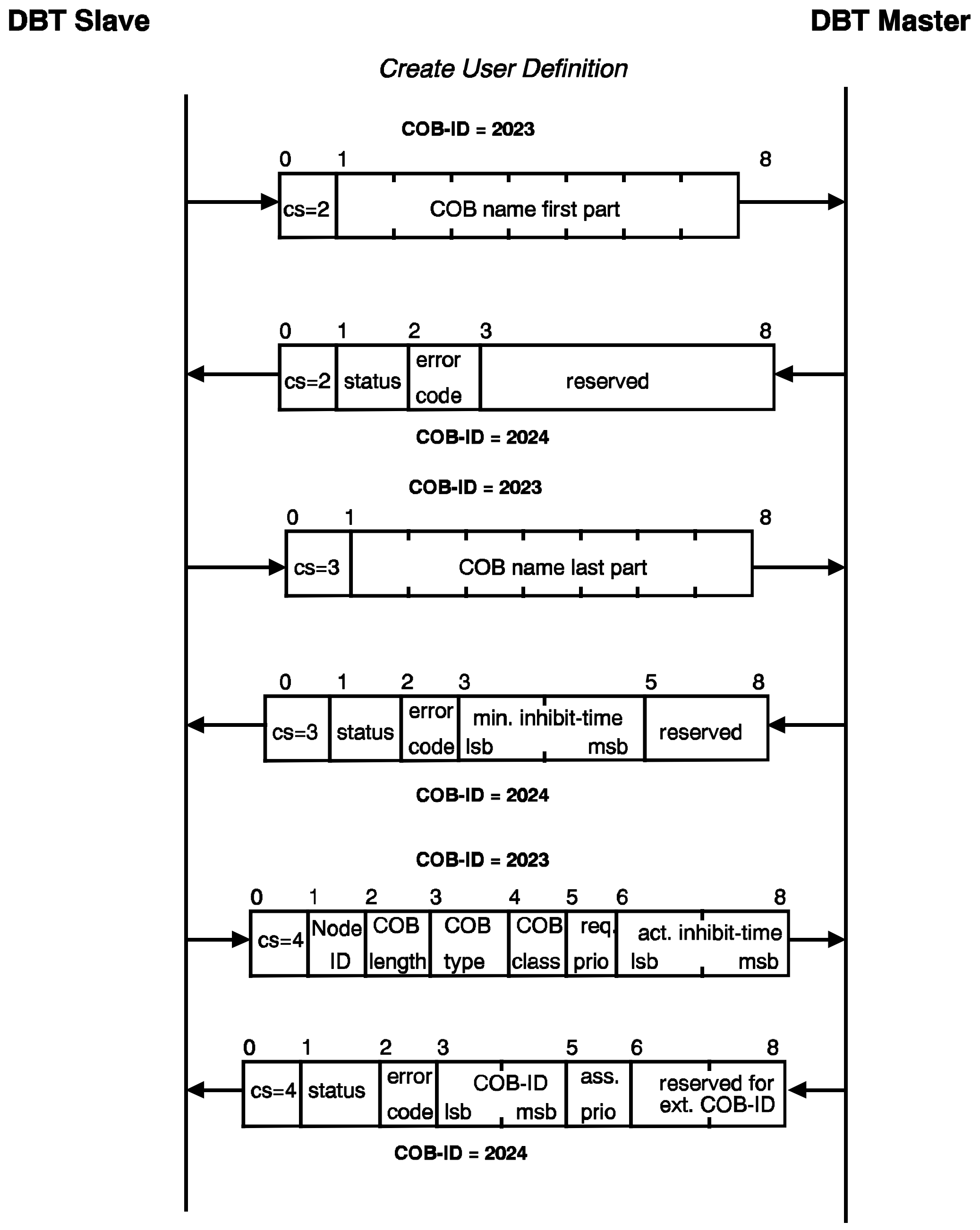
In the description of the COB data format, bytes are numbered from 0 to and including 7. Bits within a byte are numbered from 0 to and including 7. Byte 0 is transmitted first, byte 7 is transmitted last. Within a byte, bit 0 is the least significant bit, bit 7 is the most significant bit. In the protocol descriptions, [a, b] denotes the range of integers from a to b with a and b included. If a > b, the range is empty.

The terms 'lsb' and 'msb' stand for 'least significant byte' (lsb) and 'most significant byte' (msb) respectively and are used to define how an integer number is stored in more than one byte for the DBT Protocol. The order of significance is from lsb to msb.

4. DISTRIBUTION CONTROL PROTOCOLS

4.1 Create User Definition Protocol

This protocol is used to implement the 'Create User Definition' service, see /2/.



- **cs:** DBT command specifier
  2: COB name first part command
  3: COB name last part command
  4: COB attribute command

- **COB-name first part:** the first seven characters of the COB-name, see /3/.

- **COB-name last part:** the last seven characters of the COB-name, see /3/.

- **min. inhibit-time:** only valid if status = 0, otherwise reserved for further use by CiA. If valid it contains the value of the minimum inhibit-time attribute of the selected COB Definition, see /2/. If there is no COB definition in the COB Database whose user set or predefinition set contains a User Definition respectively a Predefinition for COB-name, its value must be 0

- **act. inhibit-time:** the value of the inhibit-time attribute of the User Definition, in units of 100 usec, see /2/.

- **Node-ID:** the value of the Node-ID attribute of the User Definition, see /2/.

- **COB-length:** the value of the COB-length attribute of the User Definition, see /2/.

- **COB-class:** the value of the COB-class attribute of the User Definition, see /2/.

- **COB-type:** the value of the COB-type attribute of the User Definition, see /2/.
  0: RECEIVE
  1: TRANSMIT

- **req. priority:** a value in the range [0, 7]. It indicates the priority of the CMS object that uses this COB for the CMS Protocol, see /2/.

- **status:** indicates the succes or failure of the service.
  0: service successful
  1: service not successful

- **error code:** only valid if status = 1, otherwise reserved for further use by CiA. If valid it indicates the reason for the failure.

0:	reserved for further use by CiA
1:	there are no matching, no offending and no free COB definitions in the COB Database
2:	COB Database is in the DISABLED state, see /2/
3:	there exists an offending COB Definition whose predefinition- or user set contains a definition with the same COB-name but whose user set contains definitions with a different COB-class, see /2/
4:	there exists an offending COB Definition whose predefinition- or user set contains a definition with the same COB-name but whose user set contains definitions with a different COB-length, see /2/
5:	reserved for other DBT services

6..253:	reserved for further use by CiA
254:	cs not expected by DBT Protocol
255:	other error occurred

- **COB-ID:** only valid if status = 0, otherwise reserved for further use by CiA. If valid it contains the identifier as distributed by the DBT Master

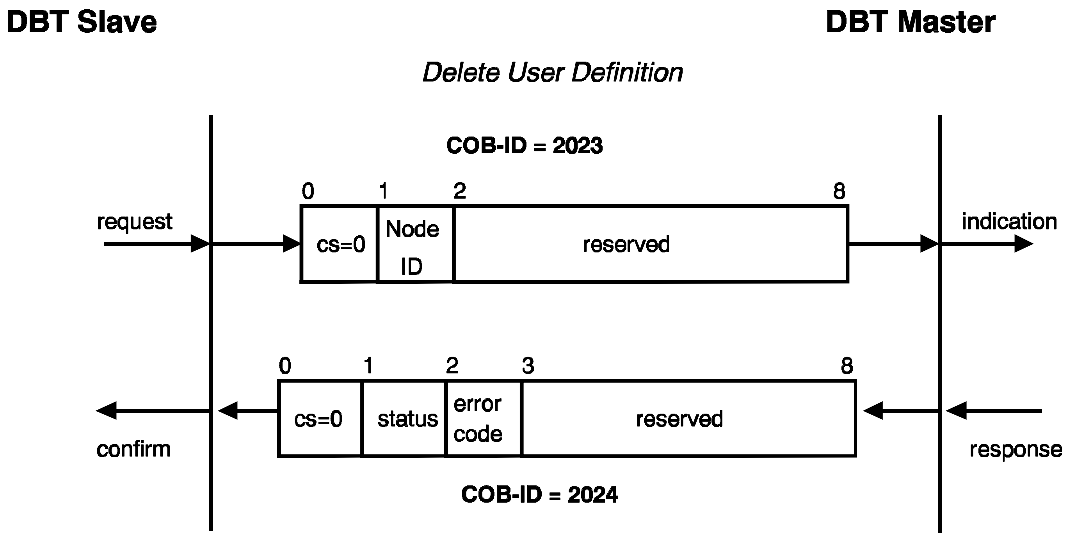
- **reserved for ext. COB-ID:** reserved to contain the value of the COB-ID if extended identifiers are used, see /4/. Its value must be 0.

- **ass. priority:** only valid if status = 0, otherwise reserved for further use by CiA. If valid it contains the priority of the CMS object according to the COB-ID as distributed by the DBT Master, see /2/.

- **reserved:** reserved for further use by CiA.

4.2 Delete User Definition Protocol

This protocol is used to implement the 'Delete User Definition' service, see /2/.



- **cs:** DBT command specifier
  0: delete COB command

- **Node-ID:** the value of the Node-ID attribute of the User Definitions that must be deleted (see /2/), or 0. If 0, all User Definitions must be deleted.

- **status:** indicates the success or failure of the service.
  0: service successful
  1: service not successful

- **error code:** only valid if status = 1, otherwise reserved for further use by CiA. If valid it indicates the reason for the failure.

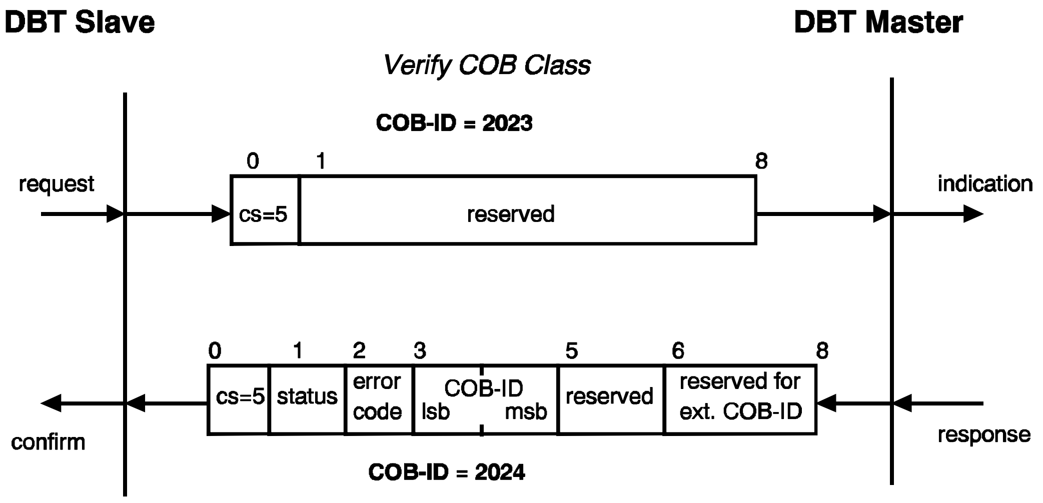
0:	reserved for further use by CiA
1:	reserved for other DBT services
2:	COB Database is in the DISABLED state, see /2/
3..5:	reserved for other DBT services
6..253:	reserved for further use by CiA
254:	reserved for other DBT services
255:	other error occurred

- **reserved:** reserved for further use by CiA.

5. CONSISTENCY CONTROL PROTOCOLS

5.1 Verify COB Class Protocol

This protocol is used to implement the 'Verify COB Class' service, see /2/.



- **cs:** DBT command specifier
  5: verfify COB class command

- **status:** indicates the success or failure of the service.
  0: service successful
  1: service not successful

- **error code:** Only valid if status = 1, otherwise reserved for further use by CiA. If valid it indicates the reason for the failure.

0:	reserved for further use by CiA
1:	reserved for other DBT services
2:	COB database is in the DISABLED state, see /2/
3..4:	reserved for other DBT services
5:	verification failed
6..253:	reserved for further use by CiA
254:	reserved for other DBT services
255:	other error occurred

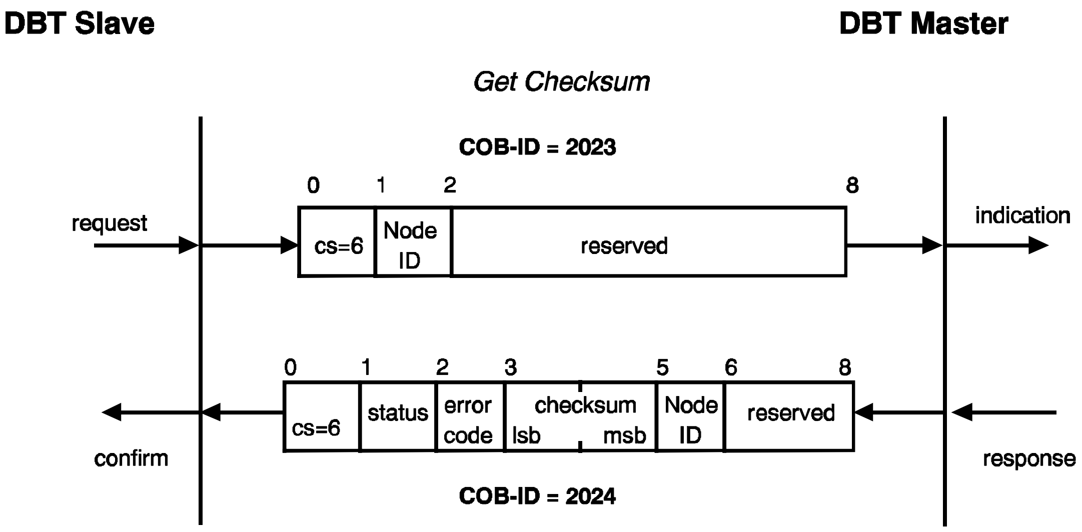
- **COB-ID:** only valid if status = 1, otherwise reserved for further use by CiA. If valid it contains an identifier for which the verification failed.

- **reserved for ext. COB-ID:** reserved to contain the value of a COB-ID for which the verification failed, if extended identifiers are used, see /4/. Its value must be 0.

- **reserved:** reserved for further use by CiA.

5.2 Get Checksum Protocol

This protocol is used to implement the 'Get Checksum' service, see /2/.



- **cs:** DBT command specifier
  6: get checksum command

- **Node-ID:** the value of the Node-ID attribute of the User Definitions for which the checksum must be calculated (see /2/), or 0. If 0, the checksum must be calculated for all User Definitions.

- **status:** indicates the success or failure of the service.
  0: service successful
  1: service not successful

- **error code:** only valid if status = 1, otherwise reserved for further use by CiA. If valid it indicates the reason for the failure.

0:	reserved for further use by CiA
1:	reserved for other DBT services
2:	COB Database is in the DISABLED state, see /2/
3..5:	reserved for other DBT services
6..253:	reserved for further use by CiA
254:	reserved for other DBT services
255:	other error occurred

- **checksum:** only valid if status = 0, otherwise reserved for further use by CiA. If valid it contains the requested checksum.

- **reserved:** reserved for further use by CiA.

ANNEX I

IMPLEMENTATION RULES

When implementing the DBT protocols, the following rules have to be followed to guarantuee inter-operability. These rules deal with the following implementation aspects:

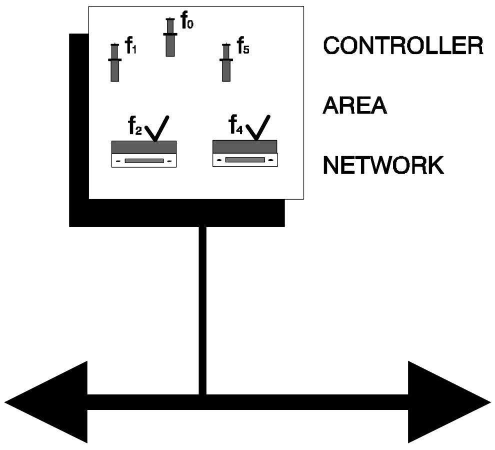
Invalid COB's

A COB is invalid if it has a COB-ID that is used by the DBT Protocol, but it contains invalid parameter values according to the DBT Protocol. This can only be caused by errors in the lower layers (see /1/) or implementation errors. Invalid COB's must be handled**locally** in an implementation specific way that does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. As far as the DBT Protocol is concerned, an invalid COB must be ignored.

Time-out's

Since COB's may be ignored, the response of a confirmed DBT service may never arrive. To resolve this situation, an implementation may, after a certain amount of time, indicate this to the service user (time-out). **A time-out is not a confirm of that DBT service**. A time-out indicates that the service has not completed yet. The application must deal with this situation. Time-out values are considered to be implementation specific and do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. However, it is recommended that an implementation provides facilities to adjust these time-out values to the requirements of the application.

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS205-1
February 1996

LMT Service Specification

1. SCOPE

This document contains the Layer Management Service Specification. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN in the OSI Reference Model

/2/: CiA/DS205-2, LMT Protocol Specification

/3/: CiA/DS207, Application Layer Naming Conventions

/4/: CiA/DS102, Version 2.0, CAN Physical Layer for Industrial Applications

3. GENERAL DESCRIPTION

3.1 LMT Perspective

LMT is one of the application layer entities in the CAN Reference Model (see /1/). LMT offers the possibility to inquire and change the settings of certain parameters of the local layers on a CAN module with LMT Slave capabilities by a CAN module with LMT Master capabilities via the CAN Network.
The following parameters can be inquired and/or changed by the use of LMT:

- NMT-address of the NMT Service Element
- bit timing parameters of the physical layer
- LMT address

By using LMT a LMT Slave can be configured for a CAN network without using any devices like DIP-switches for setting the parameters. There are several solutions available for LMT Slaves with and without a unique LMT-address or non-volatile storage.

3.2 LMT Objects and Attributes

LMT functionality is modelled using two objects (see figure 1). The LMT Master object exists exactly once in a CAN network supporting LMT. The LMT Master configures layer parameters of connected CAN modules by the use of LMT Slave objects residing on the individual modules.
Communication between LMT Master and LMT Slaves is accomplished by the LMT protocol.

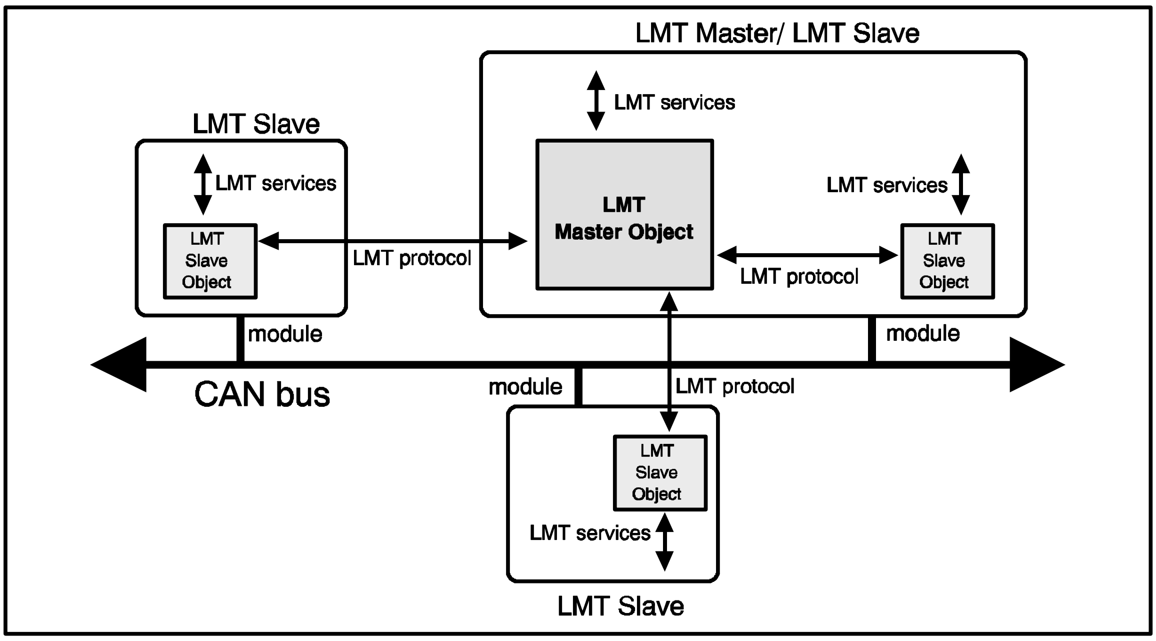


Fig. 1: The LMT Model

3.2.1 LMT Master Object

The module that configures other modules via a CAN network is called the LMT Master. There may be only one LMT Master in a network.
The LMT Master has no attributes.

3.2.2 LMT Slave Object

The module that is configured by the LMT Master via a CAN Network is called the LMT Slave. The number of LMT Slaves in a network is not limited.
The LMT Slave has the following attributes:

- **LMT Address**
  The LMT address uniquely identifies a LMT Slave. The format of the LMT address is specified in the 'Application Layer Naming Conventions' (see /3/). The LMT address of a LMT Slave can be inquired. It is valid only for LMT Slaves of class 2.

- **LMT Class**
  Each LMT class indicates the LMT capabilities that are available on a LMT-Slave.

Class 0:	No LMT Services are implemented.
Class 1:	All LMT Services with exception of Switch Mode Selective and Inquire LMT Address, Identify Remote Slaves, Identify Slaves are implemented.
Class 2:	All mandatory LMT Services are implemented.

- **LMT Mode**
  The LMT mode distinguishes between the LMT configuration phase and the operation phase of the module. In configuration mode all LMT services, in operation mode only the switch mode services are available. Any module not explicitly put into configuration mode is in operation mode.

3.3 LMT Modes and Services

LMT services can be functionally grouped in three areas:

- The switch mode services provide a way to logically connect the LMT Master and LMT Slave(s) for configuration purposes. They change the LMT mode attribute of the LMT Slave (see figure 2).
- The configuration services perform the actual task of configuring the layer parameters of an LMT Slave. The configuration services are only available in configuration mode.
- The inquiry services provide a way for the LMT Master to determine layer parameters. The inquiry services are available only in configuration mode.

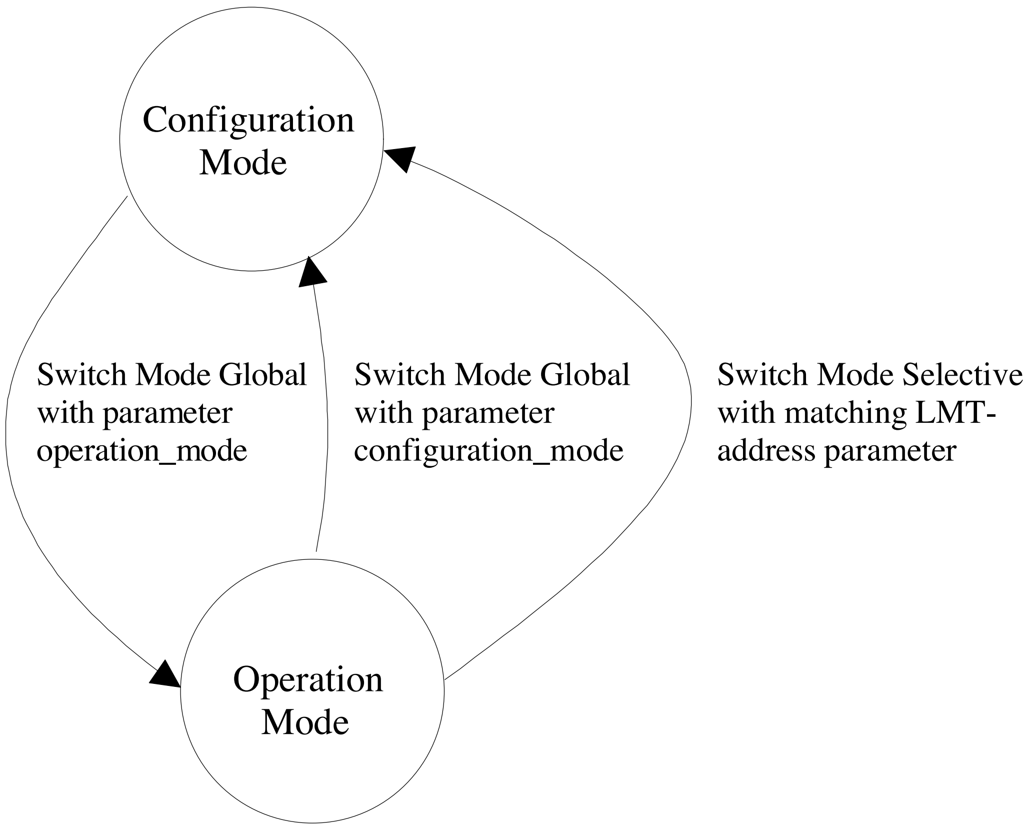

Configuration Mode
Switch Mode Global with parameter operation_mode
Switch Mode Global with parameter configuration_mode
Switch Mode Selective with matching LMT-address parameter
Operation Mode

Fig. 2: LMT modes and switching procedure

3.4 LMT Service Descriptions

The LMT services are described in a tabular form that contains the parameters of each service primitive (see /1/).

4. SWITCH MODE SERVICES

The Switch Mode Services control the mode attribute of a LMT Slave. LMT provides two ways to put a LMT Slave into configuration mode, Switch Mode Global and Switch Mode Selective. Switch Mode Selective switches exactly one LMT-Slave into configuration mode. Switch Mode Global switches all LMT Slaves into configuration mode.
Some LMT configuration and inquiry services require that only one LMT Slave is in configuration mode. To execute these services for a class 1 LMT Slave requires that only one LMT Slave is in the network.

Besides the LMT Switch Mode Services there may be other (local and module specific) means to change the mode of an LMT Slave, that are not within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications.

4.1 Switch Mode Global

This service is used to switch all LMT Slaves in the network between operation mode and configuration mode.

Parameter	*Request/Indication*
Argument	**Mandatory**
mode	mandatory
configuration_mode	selection
operation_mode	selection

This service is unconfirmed and mandatory for LMT class 1 and class 2 slaves.

4.2 Switch Mode Selective

This service is used to switch the class 2 LMT Slave, whose LMT address attribute equals LMT_address, into configuration mode.

Parameter	*Request/Indication*
Argument	**Mandatory**
LMT_address	mandatory

This service is unconfirmed and mandatory for LMT class 2 slaves.

5. CONFIGURATION SERVICES

The configuration services are available only in configuration mode. Some of the services require that exactly one LMT Slave is in configuration mode.

5.1 Configure NMT Address

Through this service the LMT Master configures the NMT-address parameter of a LMT Slave.

Parameter	*Request/Indication*	*Response/Confirmation*
Argument	**Mandatory**	
NMT-addressNMT	mandatory	
module_name	selection	
module_ID	selection	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

This service allows only one LMT Slave in configuration mode. This service is confirmed and mandatory for LMT class 1 and class 2 slaves. The remote result parameter confirms the success or failure of the service. In case of a failure optionally the reason is confirmed.

5.2 Configure Bit Timing Parameters

Through the Configure Bit Timing Parameters service the LMT Master sets the new bit timing on a LMT Slave.

Parameter	*Request/Indication*	*Response/Confirmation*
Argument	**Mandatory**	
table_selector	mandatory	
table_index	mandatory	
Remote Result		
success		**Mandatory**
failure		selection
reason		selection
		optional

By means of the table_selector the bit timing parameter table to be used is specified. In the bit timing parameter table the bit timing parameters for different baud rates are specified. With table_selector value ´0´ the standard CiA bit timing parameter table is referenced (see /4/). The table_index selects the entry (baud rate) in the selected table.

This service allows only one LMT Slave in configuration mode. The service has to be followed by an Activate Bit Timing Parameters service to activate the configured parameters. After execution of the Configure Bit Timing Parameters service the node may not execute any remote CAL services except the services Configure Bit Timing Parameters, Activate Bit Timing Parameters and Switch Mode.

This service is confirmed and mandatory for LMT class 1 and class 2 slaves. The remote result parameter confirms the success or failure of the service. In case of a failure optionally the reason is confirmed.

5.3 Activate Bit Timing Parameters

Through the Activate Bit Timing Parameters service the LMT Master activates the bit timing as defined by the Configure Bit Timing Parameters service.

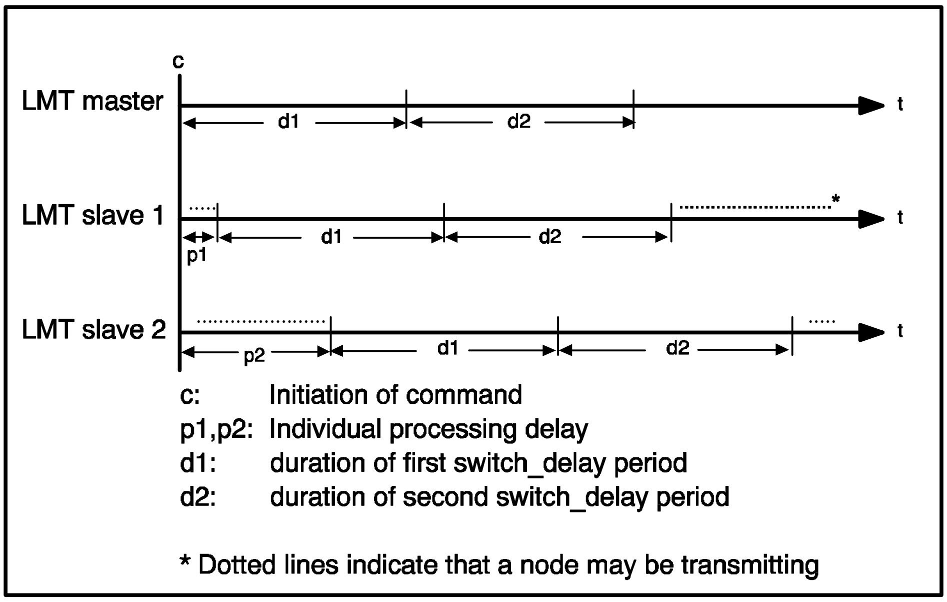
Parameter	*Request/Indication*
Argument switch_delay	**Mandatory** mandatory

The switch_delay parameter specifies the length of two delay periods of equal length, which are necessary to avoid operating the bus with differing bit timing parameters. Each node performs the actual switch of the bit timing parameters ´switch_delay´ milliseconds after the reception of the command. After performing the switch, a node does not transmit any messages before the second time ´switch_delay´ has passed.

This service is unconfirmed and mandatory for LMT class 1 and class 2 slaves.

Note

Nodes may have different processing times for performing the Activate Bit Timing Parameters command and messages that are transmitted before this command may still be in the receive queue of a node. This means that a node may still transmit CAN messages with the old bit timing during the duration of the processing delay. Therefore switch_delay has to be longer than the longest processing time of any node in the network to avoid that a node already switches while another node still transmits using the old bit timing parameters. After the time specified by switch_delay has passed the first time, every node must perform the switch during the second duration of switch_delay. Therefore after switch_delay has passed the second time, all nodes are guaranteed to be listening with the new bit timing parameters. The diagram in figure 3 shows the location of the two switch_delay periods.


c
LMT master
d1
d2
t
LMT slave 1
d1
d2
p1
*
t
LMT slave 2
p2
d1
d2
t
c: Initiation of command
p1,p2: Individual processing delay
d1: duration of first switch_delay period
d2: duration of second switch_delay period
* Dotted lines indicate that a node may be transmitting
Fig. 3: Definition of the two switch_delay periods

5.4 Store Configured Parameters

The Store Configured Parameters service is used to actuially store the configured parameters into non-volatile storage.

Parameter	*Request/Indication*	*Response/Confirmation*
Argument	**Mandatory**	
Remote Result		**Mandatory**
success		selection
failure		selection
reason		optional

This service is confirmed and mandatory for LMT class 1 and class 2 slaves. The remote result parameter confirms the success or failure of the service. In case of a failure optionally the reason is confirmed.

6. INQUIRY SERVICES

The inquiry services are available only in configuration mode.

6.1 Inquire LMT Address

This service allows to determine the LMT-address parameters of a LMT Slave in configuration mode.

Parameter	*Request/Indication*	*Response/Confirmation*
Argument	**Mandatory**	
Remote Result		**Mandatory**
LMT_address		selection
manufacturer name		mandatory
product name		mandatory
serial number		mandatory
failure		selection
reason		optional

Exactly one LMT slave may be in configuration mode when this service is executed. This service is confirmed and mandatory for LMT class 2 slaves. The remote result parameter confirms the LMT address of the LMT Slave in configuration mode or the failure of the service. In case of a failure optionally the reason is confirmed.

7. Identification Services

7.1 LMT Identify Remote Slaves

Through this service, the LMT Master requests all LMT slaves, whose LMT address meets the LMT_Address_selection to identify themselves through the 'LMT Identify Slave' service. LMT_Address_sel consists of a fixed manufacturer and product name and a span of serial numbers. This service is unconfirmed and mandatory for LMT Nodes with Class 2.

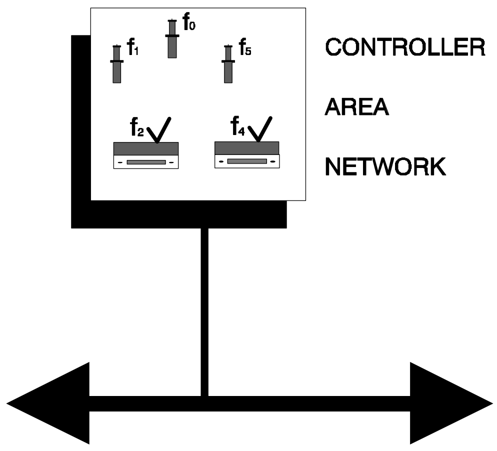
Parameter	*Request/Indication*
Argument	Mandatory
LMT_Address_sel	mandatory

7.2 LMT Identify Slave

Through this service, an LMT Slave indicates, that it is a Slave with an LMT address within the LMT_Address_sel of an 'LMT Identify Remote Slave' service executed prior to this service. The service is unconfirmed and mandatory for LMT Nodes with Class 2.

Parameter	*Request/Indication*
Argument	**Mandatory**

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS205-2
February 1996

LMT Protocol Specification

1. SCOPE

This document contains the protocol specification of the Layer Management (LMT). LMT is part of the CAN Application Layer. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN in the OSI Reference Model

/2/: CiA/DS205-1, LMT Service Specification

/3/: CiA/DS207, Application Layer Naming Conventions

/4/: CiA/DS102, Version 2.0, CAN Physical Layer for Industrial Applications

3. GENERAL DESCRIPTION

3.1 LMT Protocol Perspective

The Layer Management (LMT) service element in the CAN Reference Model (see /1/) provides the LMT services. The LMT Protocol is executed between the LMT Master and each of the LMT Slaves (see /2/) to implement these services.

3.2 LMT Slave Synchronisation

Since in the LMT Protocol all LMT Slaves use the same COB to send information to the LMT Master, there must be only one LMT Slave at a time that communicates with the LMT Master. For all protocols the LMT Master takes the initiative, a LMT Slave is only allowed to transmit within a confirmed service after it has been uniquely switched into configuration mode. Since there can be atmost one confirmed LMT service outstanding at a time (see /2/), the synchronisation is established.

3.3 LMT Protocol Descriptions

A protocol description specifies the sequence of COB's and their format that are exchanged between the LMT Master and LMT Slave(s) for a particular LMT service.

LMT uses command specifiers to identify the commands. Command specifiers from 0 - 07fh are reserved for use by standard LMT services. Command specifiers from 080h - 0ffh are free for application specific purposes and may only be used with at most one LMT Slave in configuration mode.

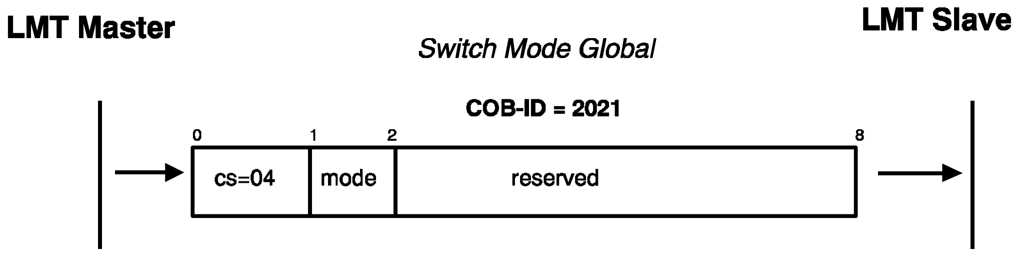
In the description of the COB data format, bytes are numbered from zero to and including seven. Bits within a byte are numbered from zero to and including seven. Byte zero is transmitted first, byte seven is transmitted last. Within a byte, bit zero is the least significant bit, bit seven is the most significant bit.

The terms 'lsb' and 'msb' stand for 'least significant byte' and 'most significant byte' respectively and are used to define how an integer number is represented in more than one byte for the LMT Protocol. The order of significance is increasing from lsb to msb.

4. SWITCH MODE PROTOCOLS

4.1 Switch Mode Global

This protocol is used to implement the Switch Mode Global service.



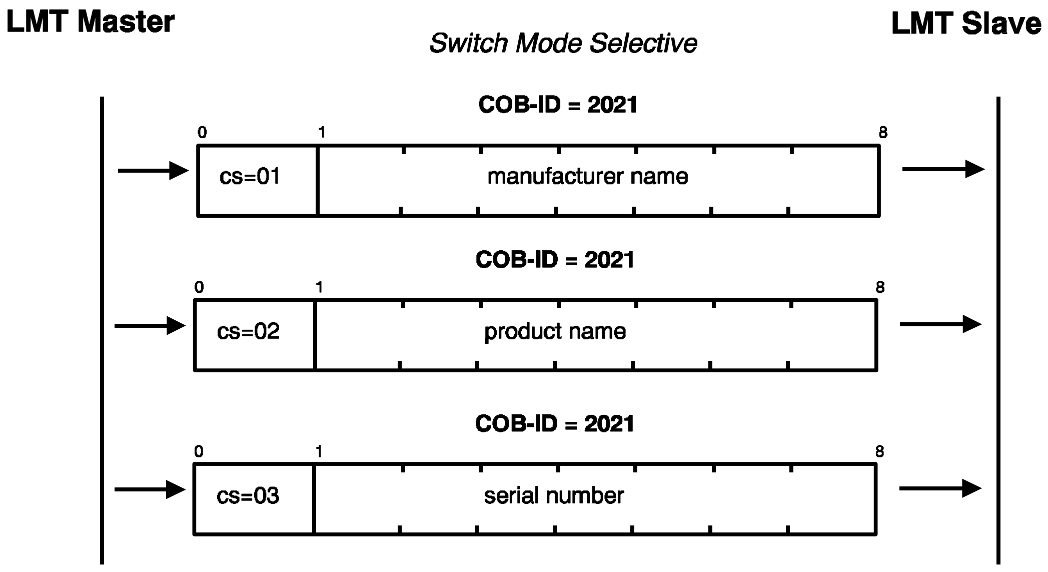
- **cs**: LMT command specifier
  04 for Switch Mode Global

- **mode**: The LMT mode to switch to:
  0: switches to operation mode
  1: switches to configuration mode

- **reserved**: reserved for further use by CiA.

4.2 Switch Mode Selective

This protocol is used to implement the Switch Mode Selective service.



- **cs**: LMT command specifiers
  01 to 03 for Switch Mode Selective
- **manufacturer_name**: The manufacturer name part of the LMT address, see /3/
- **product_name**: The product name part of the LMT address, see /3/

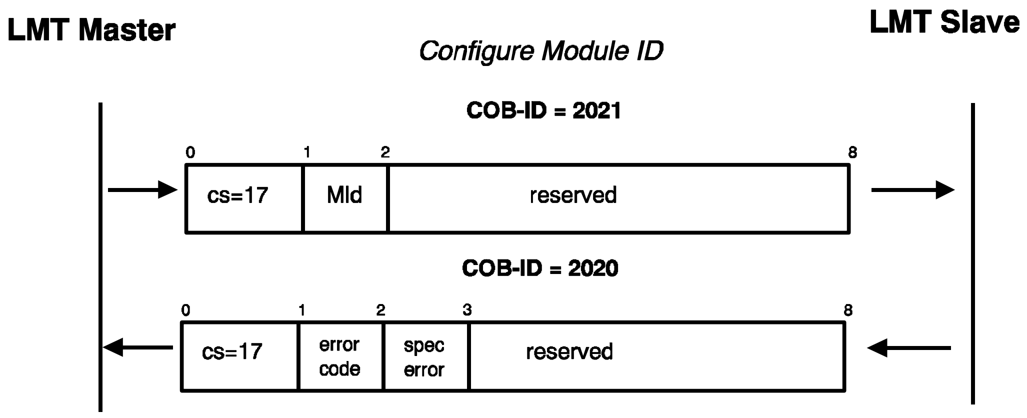
- **serial_number**: The serial number part of the LMT address, see /3/

5. CONFIGURATION PROTOCOLS

5.1 Configure NMT Address Protocols

Configure Module ID Protocol

This protocol is used to implement the Configure NMT Address service for the module-ID part of the NMT address.



- **cs**: LMT command specifier
  17 for Configure Module ID

- **MId**: The new module_id to configure, see /3/

- **error_code**:

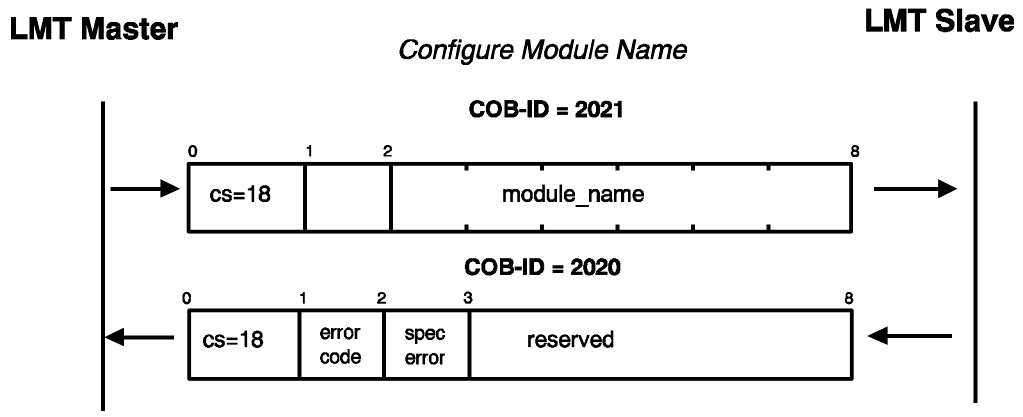
0:	protocol successfully completed
1 ... 254:	reserved for further use by CiA
255:	implementation specific error occured.

- **specific_error_code**: If error_code equals 255, specific_error_code gives an implementation specific error code, otherwise it is reserved for further use by CiA.

- **reserved**: reserved for further use by CiA

Configure Module Name Protocol

This protocol is used to implement the Configure NMT Address service for the module-name part of the NMT address.



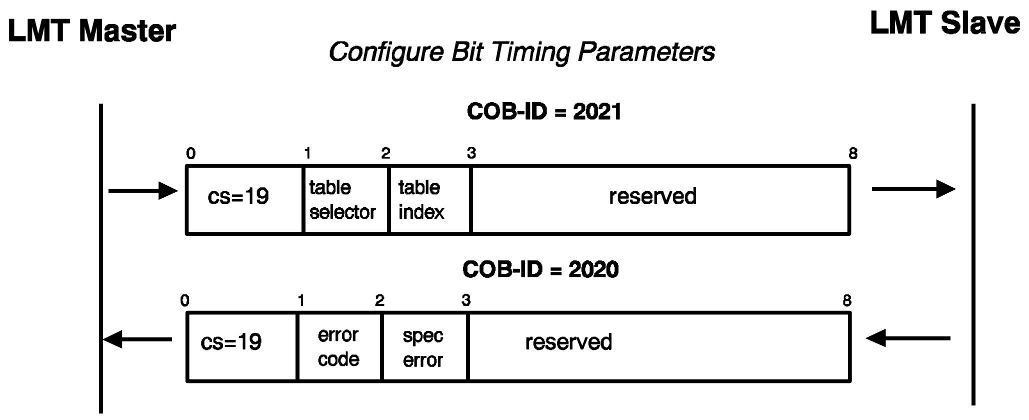
- **cs**: LMT command specifier
  18 for Configure Module Name
- **module_name**: the new module_name to configure, see /3/
- **table_index**: index for the bit timing to use.
- **error_code**:

0:	protocol successfully completed
1 . . 254:	reserved for further use by CiA
255:	implementation specific error occured

- **specific_error_code**: If error_code equals 255, specific_error_code gives an implementation specific error code, otherwise it is reserved for further use by CiA.
- **reserved**: reserved for further use by CiA.

5.3 Configure Bit Timing Parameters Protocol

This protocol is used to implement the 'Configure Bit Timing Parameters' service



- **cs**: LMT command specifier

19 for Configure Bit Timing Parameters

- **table_selector**: selects which bit timing parameters table has to be used

0:	standard CiA bit timing table (see /4/)
1..127:	reserved for further use by CiA
128..255:	may be used for manufacturer specific bit timings

- **table_index**: selects the entry (bit timeing parameters) in the selected table; see /4/ for valid indices when using the standard CiA bit timings (table_selector = '0')

- **error_code**:

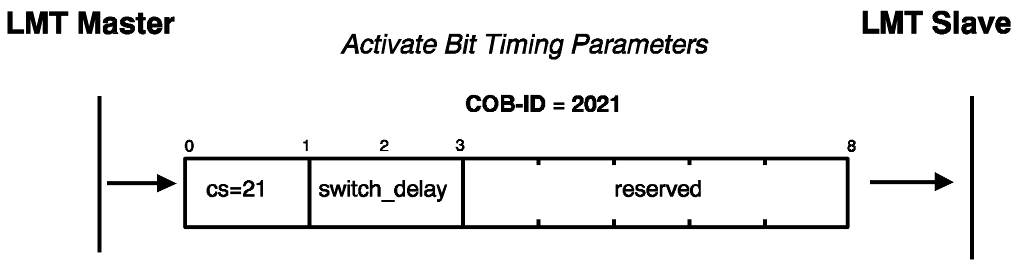
0:	protocol successfully completed
1:	bit timing not supported
2..254:	resrerved for further use by CiA
255:	implementation specific error occured

- **specific_error_code**: if error_code equals 255, specific_error_code gives an implementation specific error code, otherwise it is reserved for further use by CiA.

- **reserved**: reserved for further use by CiA.

5.4 Activate Bit Timing Parameters Protocol

This protocol is used to implement the Activate Bit Timing Parameters service.



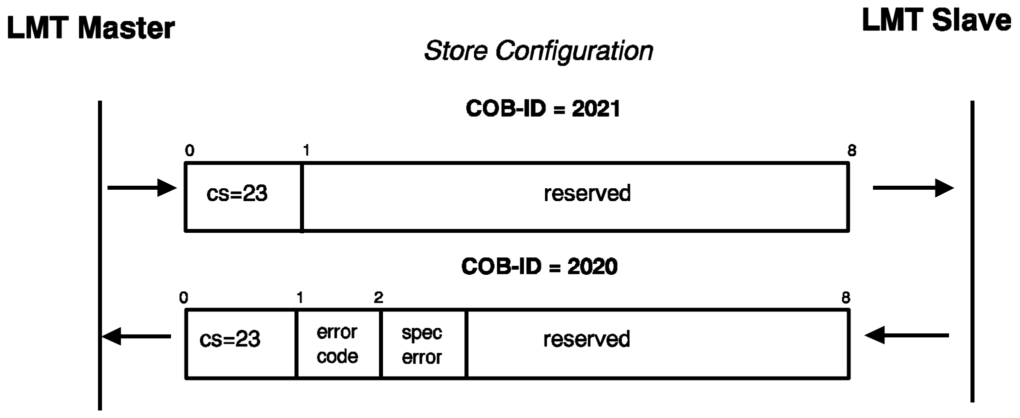
- **cs**: LMT command specifier
  21 for Activate Bit Timing Parameters

- **switch_delay**: The duration of the two periods of time to wait until the bit timing parameters switch is done (first period) and before transmitting any CAN message with the new bit timing parameters after performing the switch (second period) The time unit of switch delay is 1 ms.

- **reserved**: reserved for further use by CiA.

5.5 Store Configuration Protocol

This protocol is used to implement the Store Configured Parameters service.



- **cs**: LMT command specifier
  23 for Store Configuration

- **error_code**:

0:	protocol successfully completed,
1:	store configuration is not supported,
2 . . 254:	reserved for further use by CiA,
255:	implementation specific error occured.

- **specific_error_code**: If error_code equals 255, specific_error_code gives an implementation specific error code, otherwise it is reserved for further use by CiA.

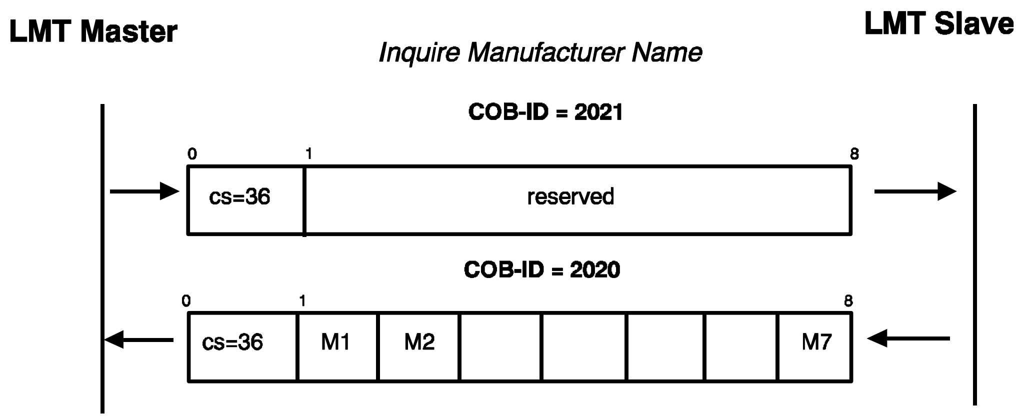
- **reserved**: reserved for further use by CiA.

6. INQUIRY PROTOCOLS

6.1 Inquire LMT Address Protocols

These protocols are used to implement the Inquire LMT Address service. To implement the service, each of the following three protocols has to be executed.

Inquire Manufacturer Name Protocol

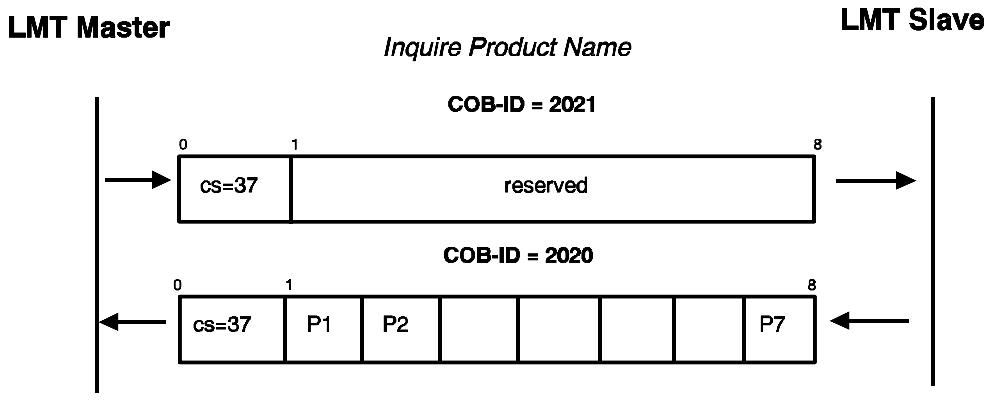


- **cs**: LMT command specifier
  36 for Inquire Manufacturer Name

- **M1 - M7**: The manufacturer_name (see /3/) of the selected module or error code. If M1 is a valid <alpha-num>, the response contains the name. If M1 is 0ffh, M2 contains the error code, M3 contains the reason if valid for the error code.

- **reserved**: reserved for further use by CiA.

Inquire Product Name Protocol

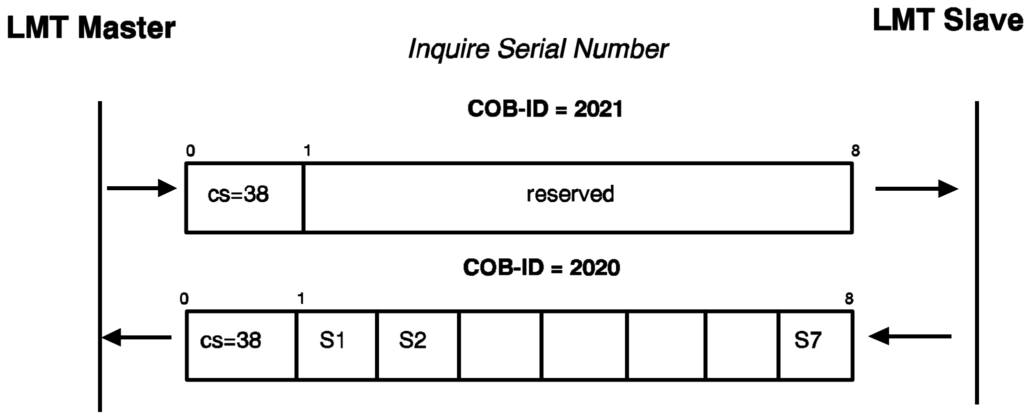


- **cs**: LMT command specifier
  37 for Inquire Product Name

- **P1 - P7**: The product_name (see /3/) of the selected module or error code. If P1 is a valid <alpha-num>, the response contains the name. If P1 is 0ffh, P2 contains the error code, P3 contains the reason if valid for the error code.

- **reserved**: reserved for further use by CiA.

Inquire Serial-Number Protocol



- **cs**: LMT command specifier
  38 for Inquire Serial Number

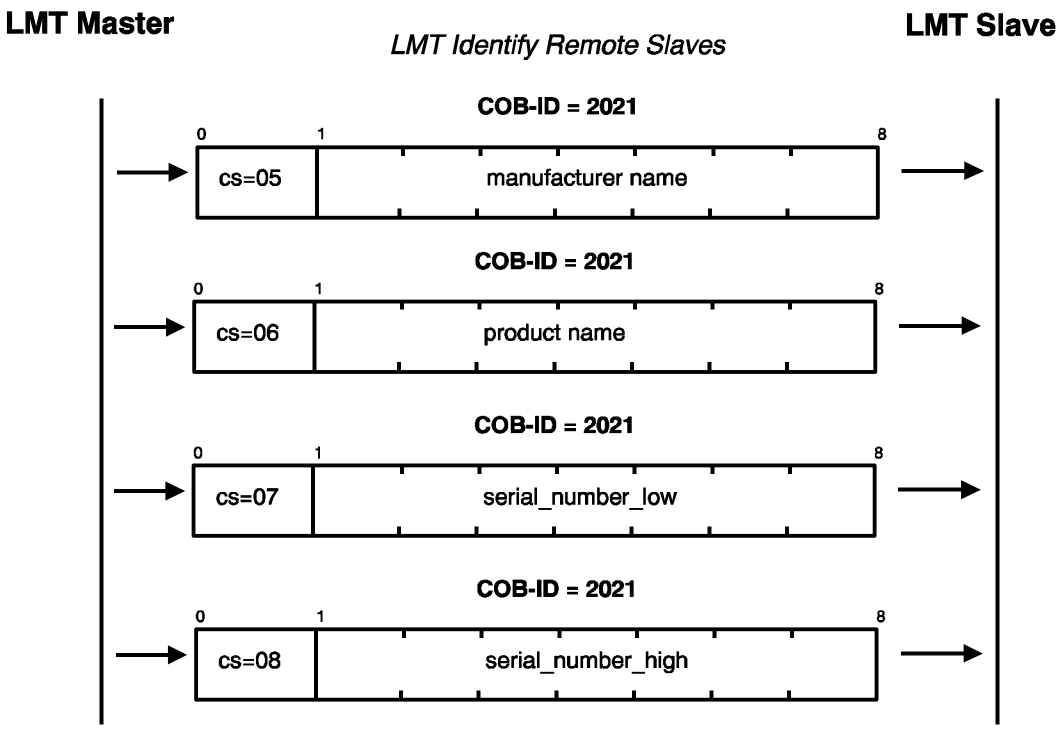
- **S1 - S7**: The serial_number (see /3/) of the selected module or error code. If S1 contains a valid BCD-pair (see /3/), the response contains the serial number. If S1 is 0ffh, S2 contains the error code, S3 contains the reason if valid for the error code.

- **reserved**: reserved for further use by CiA.

7. IDENTIFICATION PROTOCOLS

7.1 LMT Identify Remote Slaves

This protocol is used to implement the 'LMT Identify Remote Slaves' service.



- **cs**: LMT command specifier
  05 to 08 for LMT Identify Remote Slaves

- **manufacturer_name**: The manufacturer name part of the LMT Address

- **product_name:** The product name part of the LMT Address

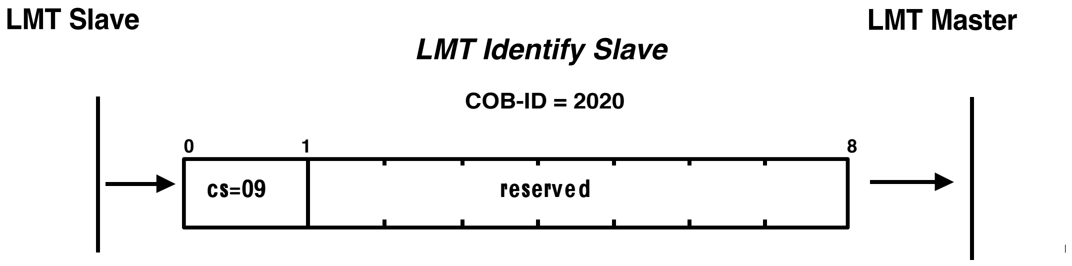
- **serial_number_low:** The lower boundary of the requested serial numbers range

- **serial_number_high:** The higher boundary of the requested serial numbers range

The boundaries are included in the interval. All LMT Slaves with matching manufacturer name and product name whose serial numbers lie within this range, are requested to identify themselves with the LMT Identify Slave service.

7.2 LMT Identify Slave Protocol

This protocol is used to implement the 'LMT Identify Slave' service.



- **cs:** LMT command specifiers
  09 for Identify Slave

- **reserved**: all bytes set to '0'

ANNEX I

IMPLEMENTATION RULES

When implementing the LMT protocols, the following rules have to be followed to guarantee inter-operability. The rules deal with the following implementation aspects:

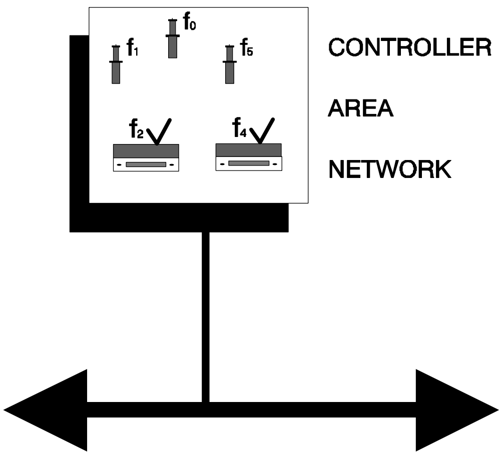
Invalid COBs

A COB is invalid if it has a COB-ID that is used by the LMT Protocol, but contains invalid parameter values according to the LMT Protocol. This can be caused by errors in the lower layer (see /1/) or implementaton errors. Invalid COBs must be handled locally in an implementation specific way that does not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. As far as the LMT Protocol is concerned, an invalid COB must be ignored.

Time-Outs

Since COBs may be ignored, the response of a confirmed LMT service may never arrive. To resolve this situation, an implementation may, after a certain amount of time, indicate this to the service user (time-out). A time-out is not a confirm of the LMT service. A time-out indicates that the service has not completed yet. The application must deal with this situation. Time-out values are considered to be implementation specific and do not fall within the scope of the CiA Standard on the CAN Application Layer for Industrial Applications. However, it is recommended that an implementation provides facilities to adjust these time-out values to the requirements of the application.

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS206
February 1996

Recommended Standard CAL Module Data Sheet

1. SCOPE

This document contains a description of a recommended standard for a CAL module data sheet. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

/2/: CiA/DS202-1, CMS Service Specification

/3/: CiA/DS203-1, NMT Service Specification

/4/: CiA/DS204-1, DBT Service Specification

/5/: CiA/DS205-1, LMT Service Specification

/6 /: CiA/DS207, Application Layer Naming Conventions

3. GENERAL DESCRIPTION

3.1 Perspective

The purpose of the recommended standard module data sheet is the provision of a standard description format for the complete specification of CAL-based modules in non-standardized-profile (proprietary) applications.

The recommended Module Data Sheet consists of three parts and shall specify the functionality of a module as accessible from the bus. This means that not only the communication interface has to be specified but also the application specific functionality.

3.2 General Description of a Module (Part A)

This part specifies the module type, function, identification and capability by means of the following information:

- **Module Type**
  Free format specification of the module type

- **Module Function**
  Textual description of the module function

- **Specification of Module Capabilities**
  in terms of LMT-, NMT- and DBT node class according to CAL service specification (see /3/, /4/, /5/). Specifies the supported LMT-, NMT- and DBT services.

- **LMT- Identification**
  manufacturer name, product name, serial number according to LMT naming conventions (see /6/). Only valid if LMT class > 1

- **NMT-Identification**
  module name, module-ID according to NMT naming conventions (see /6/).

3.3 Specification CMS Objects (Part B)

This part of the data sheet describes the supported CMS objects in terms of CMS object attributes.

- **Module relative CMS Object Number**
  beginning with 0

- **CMS Object Name**
  according to CMS naming conventions;
  for multiplexed Variables: Variable-Set-Name;
  if the object names are given in fixed format notation then the last three digits of the name representing the module ID must be written as ´xxx´ except when they are set to ´000´.

- **CMS Object Type**
  Variable or Domain or Event

- **Class**
  Variable, Domain: basic, multiplexed;
  Event: controlled, uncontrolled, stored

- **Access Type**
  only valid for Variables (Write-only, Read-only, Read-Write)

- **User Type**
  Client or Server

- **Default Priority Group**
  according to CMS specification;
  if the module does not provide a DBT slave but works with preset identifiers then this column can be split into two colums to specify the assigned COBs: one column for the transmit COB of the Client (C-Tx) and the other column for the transmit COB of the Server (S-Tx) of the object;

- **Default Inhibit Time**
  according to CMS specification (in units of 0.1 ms)

- **MUX Value**
  according to CMS specification; only valid for Variables and Domains of class multiplexed

- **Message Component Number**
  specifies the component for constructed messages beginning with '0' for the first component of the message

- **Message Component Name**
  specifies the component name in free format

- **Message Component Data Type**
  data type according to CMS specification of corresponding message component

- **Error Data Type**
  data type of corresponding error message; only relevant for Variables with confirmed data transfer (optional)

3.4 Specification of Module Functionality (Part C)

This part of the module data sheet specifies the module functionality.

- **Object Number, MUX-Value, Message Component Number**
  for reference to sheet 2

- **Meaning/Function**
  free format description of message components meaning or function

- **Engineering Units**
  if relevant according to function of component

- **Value Range**
  if relevant; e.g. valuable codes

- **Default Value**
  initialization value

- **Message Triggering Condition**
  specification of the component triggering condition if relevant, e.g. on-change, on-threshold-exceeding, on error, cyclically (cycle time), etc.

- **Error Coding**
  Specificaton of error codes, only valid for confirmed Variables

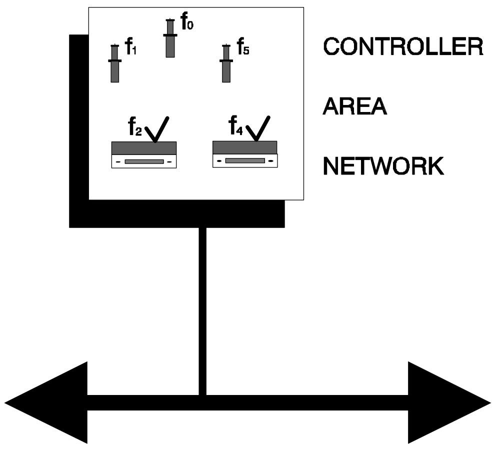
- **Remarks**
  free format

Module Data Sheet - Part A	
Module Type:	
Module Function	
Module Capabilities	
LMT class [ ] **NMT network class [ ]** **NMT node class [ ]** **DBT class [ ]**	
Module Identification	
Manufacturer Name **Product Name** **Serial Number**	
Module Name **Module ID**	

Module Data Sheet - Part B												
Module Type												
Obj. Nr.	**Object Name**	**Object Type**	**Class**	**Access Type**	**User Type**	**Default Priority Group**	**Default Inhibit Time**	**Mux Value**	**Mess. Comp. Nr.**	**Message Comp. Name**	**Message Comp. data Type**	**Error Data Type**

Module Data Sheet - Part C									
Module Type									
Obj. Nr.	**Mux Value**	**Mess. Comp. Nr.**	**Meaning, Function**	**Eng. Unit**	**Value, Range**	**Default Value**	**Message Triggering Condition**	**Error Coding**	**Remarks**

CAN in Automation (CiA)
International Users and Manufacturers Group e.V.



CAN Application Layer for Industrial Applications
CiA/DS207
February 1996

Application Layer Naming Conventions

1. SCOPE

This document contains the naming conventions that apply to instances of the objects that are defined by the service elements of the CAN Application Layer. This document is part of a set of documents that standardize the CAN Application Layer for Industrial Applications.

2. REFERENCES

/1/: CiA/DS201, CAN Reference Model

3. GENERAL DESCRIPTION

3.1 Naming Conventions Perspective

The CAN Application Layer (see /1/) offers an open CAN network where modules from different suppliers cooperate in a distributed application. To this purpose, the service elements of the CAN Application Layer (see /1/) model their functionality through the use of objects. Applications can create instances of those objects identified by a *name*. These names have a *network-wide scope*. Applications that want to execute remote services via the network on such an object must know its name and this name must identify the object.

3.2 Symbolic Name

A symbolic name is built according to the following syntax rules:

```
<symb-name>         ::= { <alpha-num> | <special> }
<alpha-num>         ::= 'A' | ... | 'Z' | 'a' | ... | 'z' |  '_' |  <num>
<num>               ::= '0' | ... | '9'
<special>           ::= '#'
```

If a symbolic name is transferred via the CAN network, each character is transferred in one data byte of the COB as an 8-bit unsigned integer, whose value is the ASCII code of that character. The bits in this byte are transferred most significant bit first. The character sequence is transmitted from left to right. Symbolic names are case sensitive.

3.3 Byte Selector

A byte selector is a non-negative integer that can assume the following values:

```
<byte-selector>        ::= 1 | ... | 255
```

If a selector is transferred via the CAN network, it is transferred in one data byte of the COB as an 8-bit unsigned integer, whose value equals the value of the selector. The bits in this byte are transferred most significant bit first.

3.4 BCD Number

A BCD number is built according to the following syntax rules:

```
<bcd-num>              ::= { <bcd-pair> }
<bcd-pair>             ::= <bcd-digit-1><bcd-digit-0>
<bcd-digit-1>          ::= <bcd-digit>
<bcd-digit-0>          ::= <bcd-digit>
<bcd-digit>            ::= 0 | ... |  9
```

If a BCD number is transferred via the CAN network, each BCD pair is transferred in one data byte of the COB as an 8-bit unsigned integer, whose value is defined as 16*<bcd-digit-1> + <bcd-digit-0>. The bits in this byte are transferred most significant bit first. The sequence is transmitted from left to right. A <bcd-pair> represents the value 10*<bcd-digit-1> + <bcd-digit-0>.

4. CMS NAMING CONVENTIONS

4.1 Purpose

CMS is one of the service elements of the CAN Application Layer, see /1/. CMS defines a number of objects and remote services on these objects. In order to implement the remote services two (or more) peer CMS entities have to exchange information using a protocol. Such a protocol uses COB's each with a unique COB-ID, to transfer the protocol data. Such A COB-ID uniquely identifies the CMS object. All clients and servers of a CMS object must use the same COB-ID for the COB's used by the protocol of that CMS object.

In the CAN Application Layer the COB-ID's of CMS objects may be distributed by the Distributor service element (DBT, see /1/). The DBT distributes COB-ID's based on the *names of the COB's* used by the protocol of that CMS object. The CMS naming conventions serve to control the distribution of COB-ID's by the DBT.

4.2 CMS Object Name Syntax

A CMS object name is a symbolic name. The Client(s) and Server(s) of a CMS object do not necessarily use the same name for an object. The DBT service element has the possibility to 'link' these names to the same CMS object.

The name of a CMS object must adhere to the following syntax:

<CMS_obj_name>	::= <fix-format-name> | <free-format-name>
<fix-format-name>	::= <CMS-prof-ID> <appl-spec-name> <CMS-node-ID>
<CMS-prof-ID>	::= { <alpha-num> }^{3}
<appl-spec-name>	::= { <alpha-num> }^{7}
<CMS-node-ID>	::= { <num> }^{3}
<free-format-name>	::= '#' { <alpha-num> }^{12}

All CMS object names consist of thirteen characters. All names that start with a '#' are free format names. Names that do not start with a '#' are names whose format is fixed by CiA.

The <CMS-prof-ID> is a sequence of three alpha-numeric characters. It indicates the profile-ID or application type of the CMS object. Profile-ID's are assigned and registered by the CiA. The profile-ID "000" is to be used by CMS objects that do not belong to a registered profile.

The <appl-spec-name> is a sequence of seven alpha-numeric characters and can be chosen freely by the application unless further conventions are defined for a certain profile-ID.

The <CMS-node-ID> is a sequence of three alpha-numeric characters that represents synbolically, from left to right, the value of the Node-ID of the NMT Slave where the CMS object is used. The Node-ID of a module is defined by the NMT Service Element, see /1/. A <CMS-node-ID> equal to "000" may be used to generate CMS object names that are independent from the Node-ID of the NMT Slave where they are used.

4.3 COB Name Syntax

A COB name is a symbolic name consisting of fourteen characters. The names of the COB's that are used by a protocol of a CMS object are defined by the following rules:

- If the protocol of a CMS object requires one COB, the name of that COB is equal to the name of that CMS object concatenated with the character 'X'.
- If the protocol of a CMS object requires two COB's, the COB transmitted from Client to Server is equal to the name of that CMS object concatenated with the character 'C'.
- If the protocol of a CMS object requires two COB's, the COB transmitted from Server to Client is equal to the name of that CMS object concatenated with the character 'S'.

```
<COB-name>          ::= <X-COB-name> | <C-COB-name> | <S-COB-name>
<X-COB-name>        ::= <CMS-name> 'X'
<C-COB-name>        ::= <CMS-name> 'C'
<S-COB-name>        ::= <CMS-name> 'S'
```

4.4 CMS Data Type Names

CMS Data Type Names are symbolic names for CMS Data Types. CMS Data Type Names are not transferred via the CAN Network.

5. NMT NAMING CONVENTIONS

5.1 Purpose

The NMT naming conventions serve to identify the NMT objects that can be managed via remote services on these objects.

5.2 NMT Object Name Syntax

Since there can be only one network object in a CAN network, it does not require a name.

An NMT Slave is identified by an *NMT-Address*. An NMT-Address consists either of a *module-ID* or of a *module-name* and a *module-ID*. A module name is a symbolic name. A module-ID is a selector. They adhere to the following syntax:

<NMT-Address>	::= <module-name> <module-ID>|<module-ID>
<module-name>	::= { <alpha-num> }^{7}
<module-ID>	::= <byte-selector>

An NMT-Address can be configured on an NMT Slave by the LMT Service Element (see /1/) or by module specific means that do not fall within the scope of the CiA Standard on the CAN Application Layer. For an NMT-Address at least one of the following conditions must be met:

- there is no other NMT Slave in the network with the same <module-name>
- there is no other NMT Slave in the network with the same <module-ID>

6. LMT NAMING CONVENTIONS

6.1 Purpose

The LMT naming conventions serve to identify the LMT objects that can be managed via remote services on these objects.

6.2 LMT Object Name Syntax

A class 2 LMT Slave is identified by an LMT-Address. An LMT Address consists of a *manufacturer-name*, a *product-name* and a *serial-number*. The manufacturer-name and product name are symbolic names. The serial-number is a BCD number. They adhere to the following syntax:

<LMT-Address>	::= <manufacturer-name> <product-name> <serial-number>
<manufacturer-name>	::= { <alpha-num> $\}^7$
<product-name>	::= { <alpha-num> $\}^7$
<serial-number>	::= { <bcd-pair> $\}^7$

A <manufacturer-name> is assigned to module suppliers by CiA. A <product-name> and a <serial-number> are assigned by the module supplier. For LMT-Addresses the following condition must be met:

- there exists no other class 2 LMT Slave in the world with the same <LMT-Address>