CANopen

Framework for Programmable CANopen Devices

CiA Draft Standard Proposal 302

Not recommended for implementation

May be changed without notification

Version 3.1.1

Date: 10.05.2002

History

Date	Version	Changes
05.03.1998	1.0	Initial revision
27.11.1998	2.0	• New chapter for NMT Master related objects • Extension of Configuration Manager • Groups / Multiplexor PDO: Clarification and new objects for configuration
29.06.2000	3.0	• Introduced definition of the boot-up procedure • Renamed chapter Slave Assignment to Network List • New objects for the network list • Clarification of existing objects according to DS-301 V4 • Moved chapter Configuration Master behind chapter NMT Master • New objects for the Configuration Manager • Adaptation of Client/Server relationships to Producer/Consumer model according to DS-301 V4. Removed references to CAL. • Network variables may have access type rww • Removed duplicated example in section 8.3 • Moved data type declaration 23H to 25H due to overlap with DS-301 V4 • Moved objects 1020H Verify Configuration, 1021H/1022H EDS Storage to appendix of DS-301 • Moved chapter OS Command and Prompt to appendix of DS-301 • Moved chapter Groups to appendix of DS-301 • Change of SDO Manager Mechanisms
07.03.2002	3.1	• Definition of CANopen Manager includes now the case NMT Master + Configuration Manager • Self-starting devices • Flying Master taken over from SIG Maritime with some changes • Updated references • Object 1F81H, Bit 1 has become obsolete by the defined processes; removed • Clarification of object attribute M/O • Editorial changes • Added Error Codes
08.05.2002	3.1.1	• Bug fixes and clarifications

Table of Contents

Figures

1 Scope

The CANopen Communication Profile (DS-301) defines the basic communication mechanisms for exchanging data via a CANopen-based networks. This includes the structure of the object dictionary, the network management and boot-up as well as communication objects like PDO, SDO, SYNC and time stamp. The object dictionary provides a standard interface for accessing of communication parameters as well as process data. The part of the object dictionary which describes the general device and communication parameters is common for all devices types.

Application specific functionalities which are provided by certain device types are detailed in specific device profiles (DS-4xx). A device profile is always based upon the definitions in the communication profile.

In general the mechanisms which are specified in the communication profile are sufficient for the definition of profiles for devices which, on the application level, provide some kind of I/O functionality. Example devices include I/O modules, drives and regulators. These devices whilst they may be complex are not termed 'intelligent' as they do not run an application level program.

For the description and operation of intelligent devices further mechanisms are necessary which are specified in DS-302. DS-302 has to be regarded as a framework for the definition of device profiles for intelligent or programmable devices in form of an extension to the communication profile DS-301. The additional mechanisms specified in DS-302 are useful especially for intelligent devices like PLCs, HMIs or CANopen tools.

DS-302 comprises the following mechanisms and definitions:

- The term CANopen Manager is introduced to specify more clearly the *network functionality* of a network controlling device.
- Definition of the Boot-Up process and the related objects.
- A possibility for configuration of unconfigured nodes during system boot-up by means of a Configuration Manager.
- The dynamic establishment of SDO connections between devices. Dynamic SDO connections are handled by the SDO Manager.
- The definition of dynamically allocated entries in an object dictionary which can be used for the representation of I/O data e.g. on programmable nodes like PLCs.
- A general mechanism for downloading program data and functions for the control of programs on a device.

Some of these new mechanisms are also useful not only for intelligent or programmable devices.

2 References

/1/ CiA DS 301, CANopen - Communication Profile for Industrial Systems, v 4.01, June 2000

/2/ CiA DSP-305, CANopen Layer Setting Services and Protocols (LSS), v1.0, May 2000

/3/ CiA DSP-306, Electronic Data Sheet Specification, v1.1, June 2001

/4/ CiA DSP-405, Device Profile for IEC1131 Programmable Devices, v2.0, December 2000

3 CANopen Manager, Terms and Definitions

Besides the application process several different additional functionalities can exist in a CANopen system. These functionalities are referred to by different terms. This chapter is intended to clarify these terms.

Within a distributed system the application process is divided into several parts running on different nodes. From the applications point of view usually one node is responsible for the control of the system. This node is called *application master* (e.g. a PLC).

From the network's point of view there are several additional functionalities which not directly deal with the application but provide application supporting functions. These additional functionalities are based on a master / slave, client / server or producer / consumer relationship.

- *NMT Master*
  The network management (NMT) provides services for controlling the network behaviour of nodes as defined in /1/ DS-301. All nodes of a network referred to as NMT Slaves are controlled by services provided by an NMT master and which have to be executed by an NMT master application. Usually the NMT master application is also part of the application master.

- *SDO Manager*
  The SDO Manager is an optional functionality responsible for handling of the dynamic establishment of SDO connections as defined in chapter 7. If an SDO Manager is present in a system it must reside together with the NMT Master on the same node.

- *Configuration Manager*
  The Configuration Manager is an optional functionality which provides mechanisms for configuration of nodes in a system during boot-up as defined in chapter 6. The mechanisms are called Configuration Management CMT. The Configuration Manager must reside on the same node together with the NMT Master and SDO Manager.

- *SYNC Producer*
  The SYNC Producer is an optional functionality which is responsible for transmitting the SYNC object. It may reside on any one node in a CANopen system.

- *TIME Producer*
  The TIME Producer is an optional functionality which is responsible for transmitting the TIME STAMP object. It may reside on any one node in a CANopen system.

- *LSS Master*
  The layer specification services (LSS) provides services for configuring layer 2 (bit timing) and NMT (Node-ID) via CAN as defined in /2/ DS-305. All nodes in a network which support LSS services are LSS Slaves. The services are provided by the LSS Master and used by a LSS Configuration Application.

Because it is usual to combine several of the additional functionalities on one node an additional term is introduced: the *CANopen Manager*.

A node is referred to as a *CANopen Manager* when the functionality of an NMT Master is provided by the node and at least one of the functionalities SDO Manager or Configuration Manager.

Basically all objects in this document are optional. If denoted as mandatory, this is valid if the concerned functionality is provided by the device (e.g. SDO Manager). Some objects consist of a set of bits, specifying several kinds of behaviour (as e.g. 1F80H). Only those bits have to be implemented that correspond to a supported behaviour. If not implemented, the bit has to be 0.

4 Boot-Up Procedure

When the CANopen Manager starts after Power-On, it will perform the state machine according to DS-301. Before switching the state from Pre-Operational to Operational it has the task of booting all assigned slaves. The overall process is shown in the following two flow charts. The flow charts show the process with the most complete set of implemented features. Refer also Figure 10.

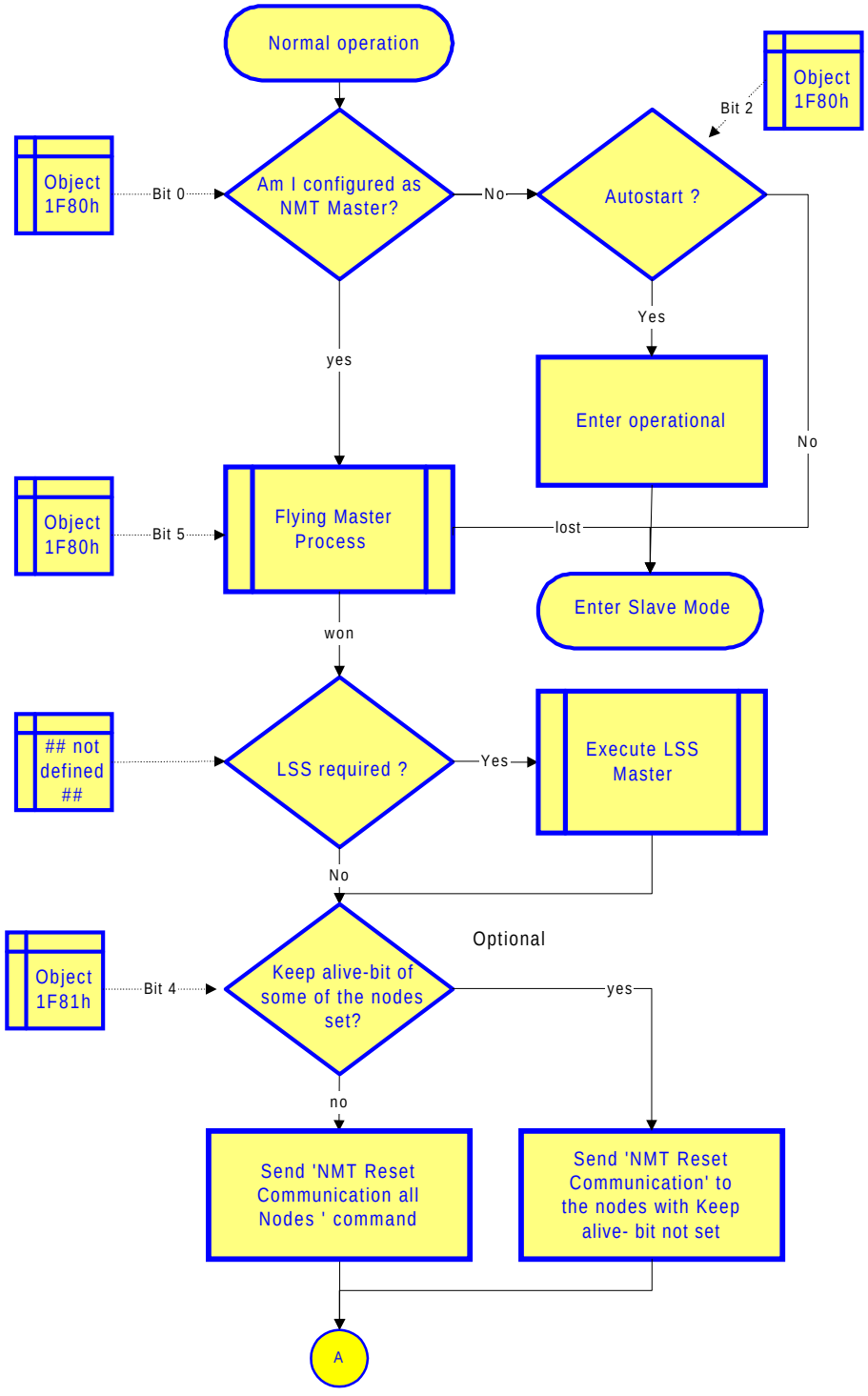


Figure 1: CANopen boot-up procedure main flow, part 1.
Predefined process 'Execute LSS' is not given in this document.

The Boot-Up main flow consists of the following basic steps:

1. If the Flying Master process shall be executed, this finds out the active NMT Master. The process itself is described in chapter 5.5. The process can directly lead to Slave Mode.

2. Bit 0 of the object 1F80h is checked to decide upon whether this node is assigned to be the NMT Master or not. If not, the node will enter a slave mode if supported.

3. For systems that require setting of Node-IDs or baudrate CANopen defines the LSS. If necessary, this has to be performed (perhaps even before step 1). The block in the flow chart is only a place holder. It is not the scope of this document to specify the concrete entry point for LSS actions.

4. The Master starts with the service NMT Reset Communication All Nodes, except if any of the nodes is forbidden to be reset without state check. In some applications some of the slave nodes may enter a special mode (like manual mode) in case of CANopen manager sudden drop-out. If the continuous state is the safe state, it may not be tolerable to send NMT Reset Communication -command for such a node if the state of the node is initially Operational. Bit 4 in the object 1F81h (see chapter 5.2) is provided to force a state check prior to sending NMT Reset Communication -command. In case the Bit 4 of the object 1F81h in any of the sub-indexes is non-zero, NMT Reset Communication All Nodes must not be sent, but the nodes are reset individually.

   The Reset Communication shall not apply for the Master itself.

   This step allows to put the slaves in a state, where the settings of all parameters are well-defined. In the case, that the Master detected a booting slave later on, it uses the service only for the individual node.

   The main flow resumes in Figure 2 .

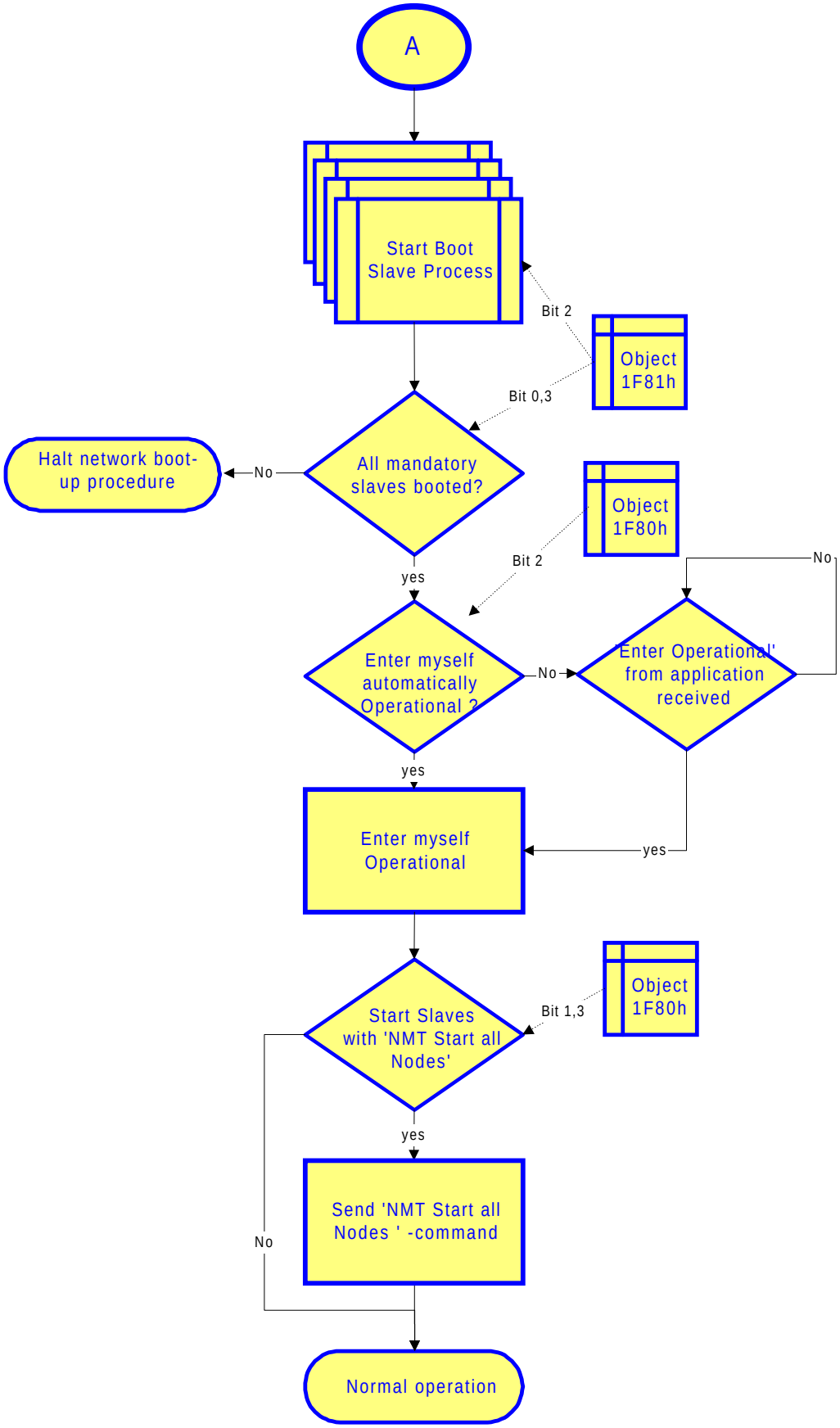


Figure 2: CANopen boot-up procedure main flow, part 2.

5. The main flow resumes with the Start Boot Slave Process as illustrated in Figure 3. This will try to boot all optional slaves at least once and will boot all mandatory slaves.
6. If an error occurs on booting a mandatory slave the main flow forces a halt of the boot-up procedure.
7. Start Remote Node

   After all mandatory nodes are started, the Master will enter state Operational according to NMT Start-up configuration bit 2 in object 1F80H (refer to chapter 5.1).

   If the bit 3 of object 1F80h (see chapter 5.1) allows, perform the service NMT Start Remote Node (= 'Enter Operational') – either individually after finishing the configuration or by broadcast command after all nodes have been configured. Whether the NMT Start Remote node should be sent individually or globally is configured in bit 1 of the object 1F81h (refer to chapter 5.2).

The Start Boot Slave Process is given by Figure 3.

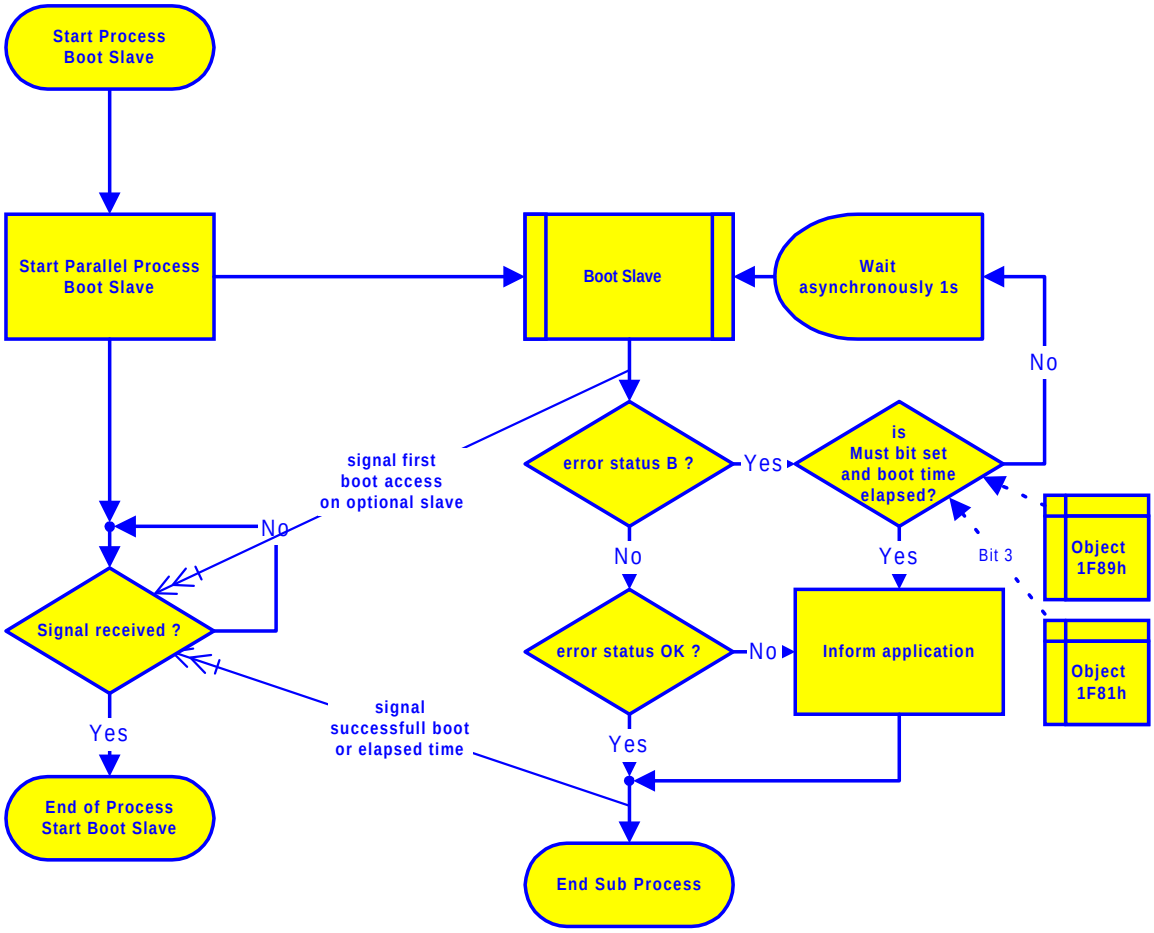


Figure 3 : Start Boot Slave Process

The steps of the Start Boot Slave Process are as follows:

1. Start parallel Process
2. Wait for a signal, that a boot was successful or at least was tried once

The parallel process will

1. Perform the process Boot Slave
2. Create Signal for a Boot Slave try

3. If Boot Slave returned with status OK finish the process. For optional slaves the process runs endlessly until the slave is found. The recommended cycle time for Baudrates above 125 kBit/sec is 1 second. This will stop only, if the slave is found or the application removes the slave from the network list. The loop will stop also if the slave is mandatory and the predefined timeout (object 1F89h) occurs. In that case the Boot slave -process ends with an error status. The application and error specific error handler may then give a warning to the operator and may resume with other nodes in a 'limb mode'. If none of the loop end conditions above is satisfied, the slave poll loop will run endlessly. The check is done asynchronously. That means, that other processes will run in parallel.

The "Boot Slave" -process is given by the following three flow charts:

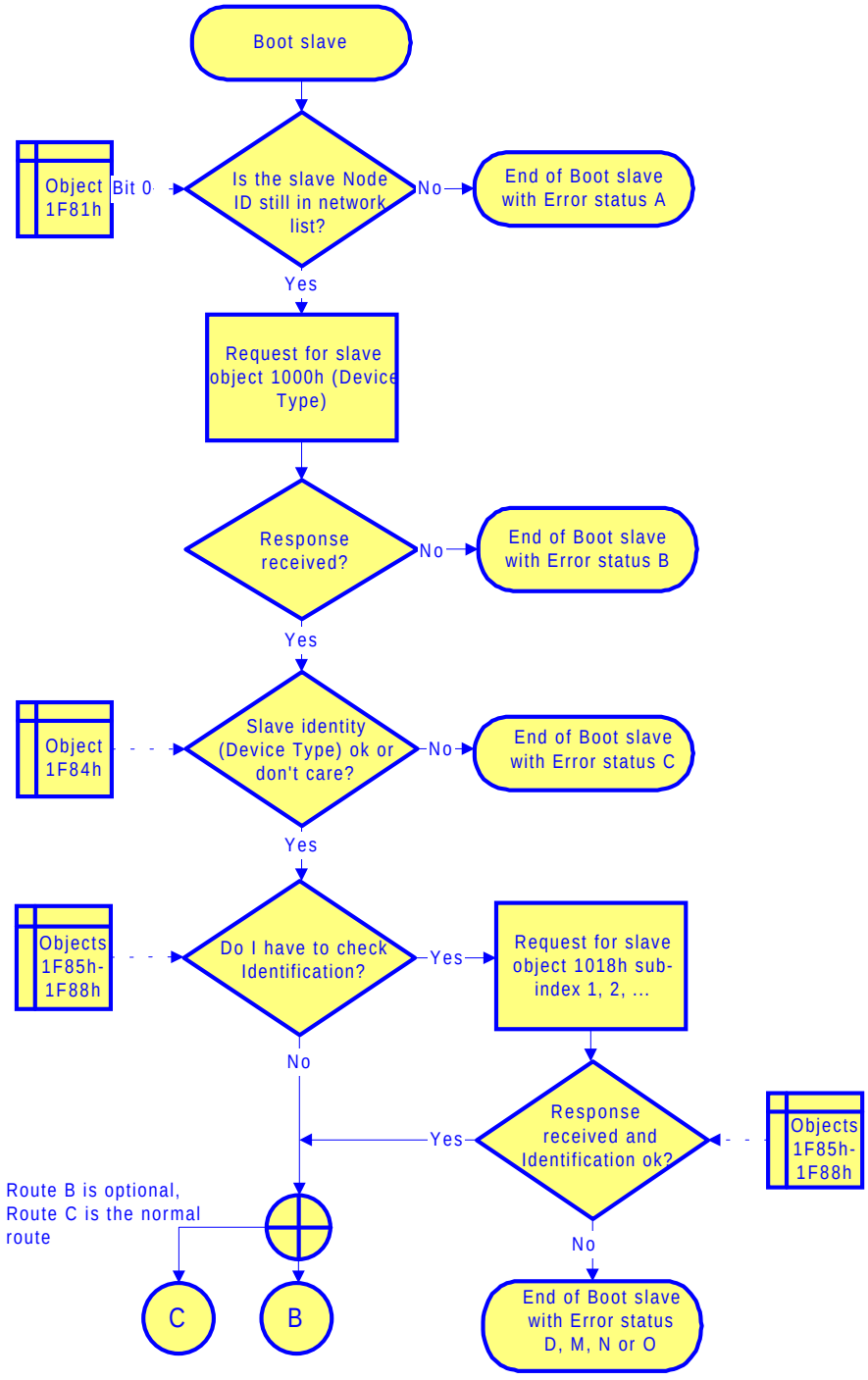


Figure 4: Boot slave predefined process, part 1.

The steps of the Boot slave -process are as follows:

1. Attempt to read slave's index 1000H to check if the node is present.

   If the slave does not answer the read request, the Boot slave -process ends with an error status.

2. Identification

   If the Device Type Identification value for the slave in Network List object 1F84H (refer to 5.2) is not 0x0000 ("*don't care*") compare it to the actual value.

   If the configured Vendor ID in Network List object 1F85H is not 0x0000 ("*don't care*"), read slave's Index 1018H, Sub-Index 1 and compare it to the actual value. The same happens with ProductCode, RevisionNumber and SerialNumber with the according objects 1F86H-1F88H.

   On identification failure, the process continues with error handler (whose reaction is application specific).

The Boot slave -process continues with an optional part (Part 2, see Figure 5), which introduces two optional boot-up features: keeping alive initially operational slave nodes and controlling application software versions including automatic update of the slave software.

The Keep alive -feature is used in situations where a slave node is already in Operational state when the CANopen manager starts up. This situation may happen for example in case of sudden drop-out and restart of the CANopen manager. In some cases it is not tolerable to reset the slave node, because the slave node may have entered a special mode (e.g. manual operation mode) in case it notices a drop-out of the CANopen manager. This type of mode is needed if the continuous state of the sub-process controlled by the slave is the safe state. The slave node notices the restart of the CANopen manager from the 'NMT Start Remote Node' -indication and may resume its normal operation mode.

The software version control is used in situations where it is important to check the application software version prior to starting of the remote node. The 'Check and update software version' -predefined process includes also possibility to download the application software automatically if the software version is not correct. In case of a checksum error during slave start-up self diagnostics, the slave may also respond deliberately with a wrong version information (like zero date and zero time) to force the CANopen manager to download the application software. Automatic download requires a file system to exist on the CANopen manager to store the application software of the slave nodes that are configured to join the automatic software download process.

Both or only one of the two features may be implemented optionally.

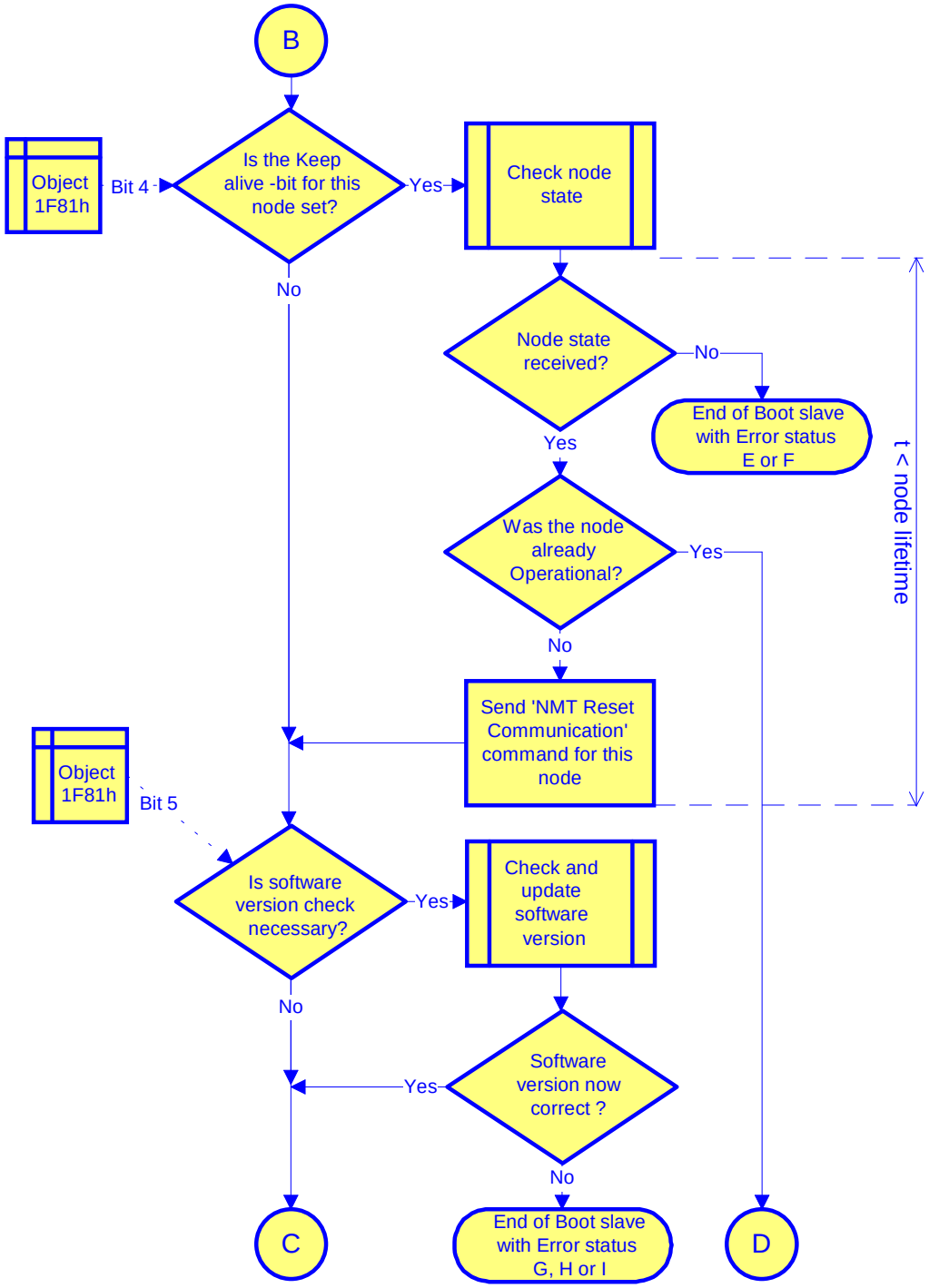


Figure 5: Boot slave predefined process, part 2 (optional).

3. Keeping alive initially operational slaves

If the Keep alive -bit (bit 4 of object 1F81h, see chapter 5.2) is set the state of the slave has to be checked to verify whether the slave is in Operational state or not. If the node state was not received the process is resumed by the application and error specific error handler. In case the state information is received and is noticed to be 'Operational', the process resumes from connection point D in Figure 8. In practice this means that software version checking and configuration version checking steps are skipped. If the state was not 'Operational', the NMT Master

sends 'NMT Reset Communication' -command to this node. Note that in case the slave supports Node Guarding protocol, the Check node -process sends a single RTR to request the Node Guard response. Hence, the slave is unintentionally fooled to start its local Error Control process. Therefore, it is recommended to keep the time between sending of the RTR and sending the 'NMT Reset Communication' -command less than Node Life Time in order to prevent the slave from generating a Life Guarding Event. However, in most cases it may cause no harm if the Life Guarding Event nevertheless occurs, as the 'NMT Reset Communication' -indication occurs subsequently anyway. In case the slave is already Operational and the next RTR is delayed more than Node Life Time, succeeding Life Guarding Event will not cause problems, because the slave had already got the Life Guarding Event due to CANopen manager drop-out. The slave must not resume its normal operation as a consequence of receiving RTR but only after receiving NMT Start Remote Node -command.

Check node state -predefined process is illustrated in Figure 6

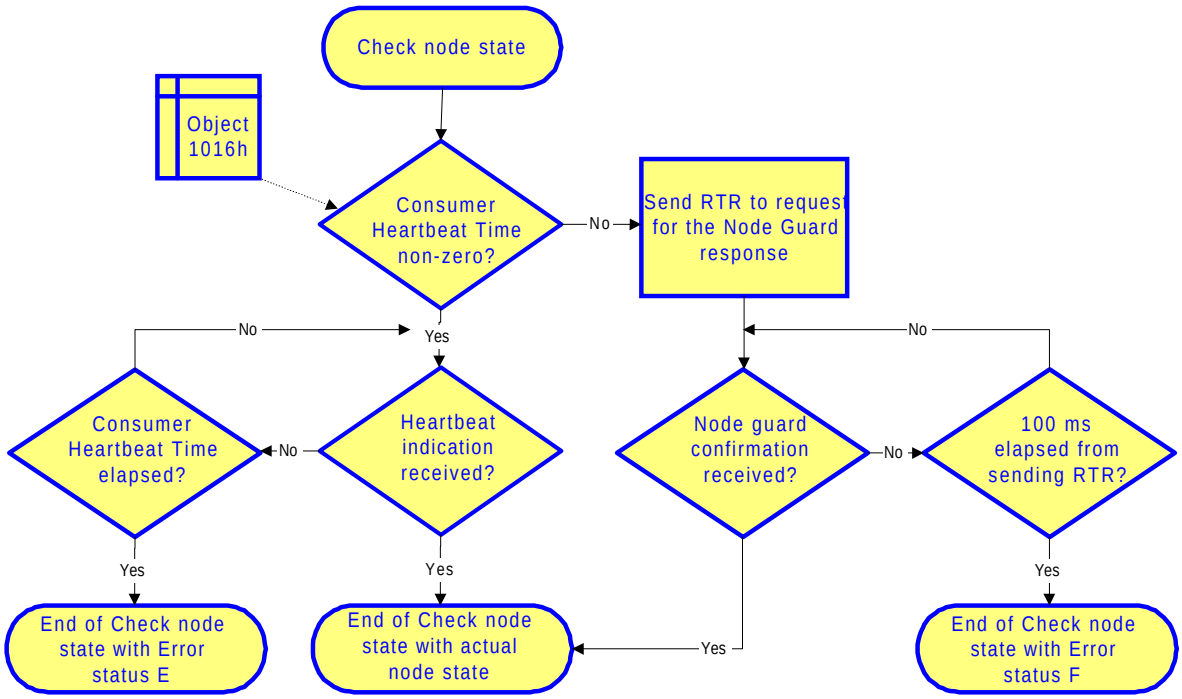


Figure 6: Check node state -predefined process.
(needed only if the optional part of the Boot slave -process is implemented).

4. Application software version control

Setting Bit 5 of the object 1F81h (see chapter 5.2) forces application software version verification. The software version is checked and if automatic software update is allowed for the node, the software is downloaded in case of version inconsistency. In case of errors during version checking or software download, the process is resumed by the error handler.

Check and update software version -predefined process is illustrated in Figure 7.

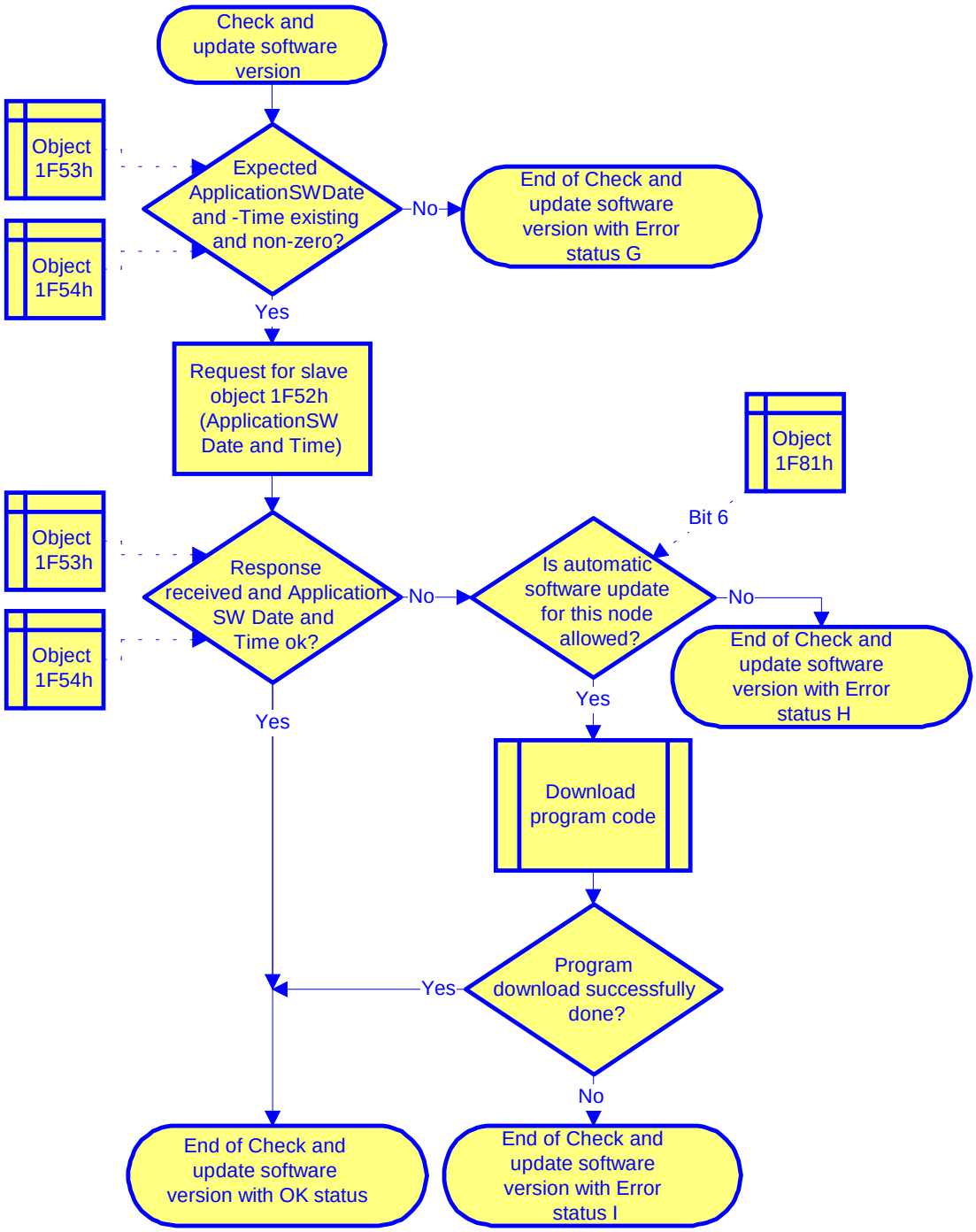


Figure 7: Check and update software version -predefined process
(needed only if the optional part of the Boot slave -process is implemented).
Download program code -predefined process is not illustrated in this context.
For objects 1F52h to 1F54h refer to chapter 9; for bit 6 of object 1F81h refer to chapter 5.2.

The Boot slave -process continues with part 3 (see Figure 8). This part is mandatory.

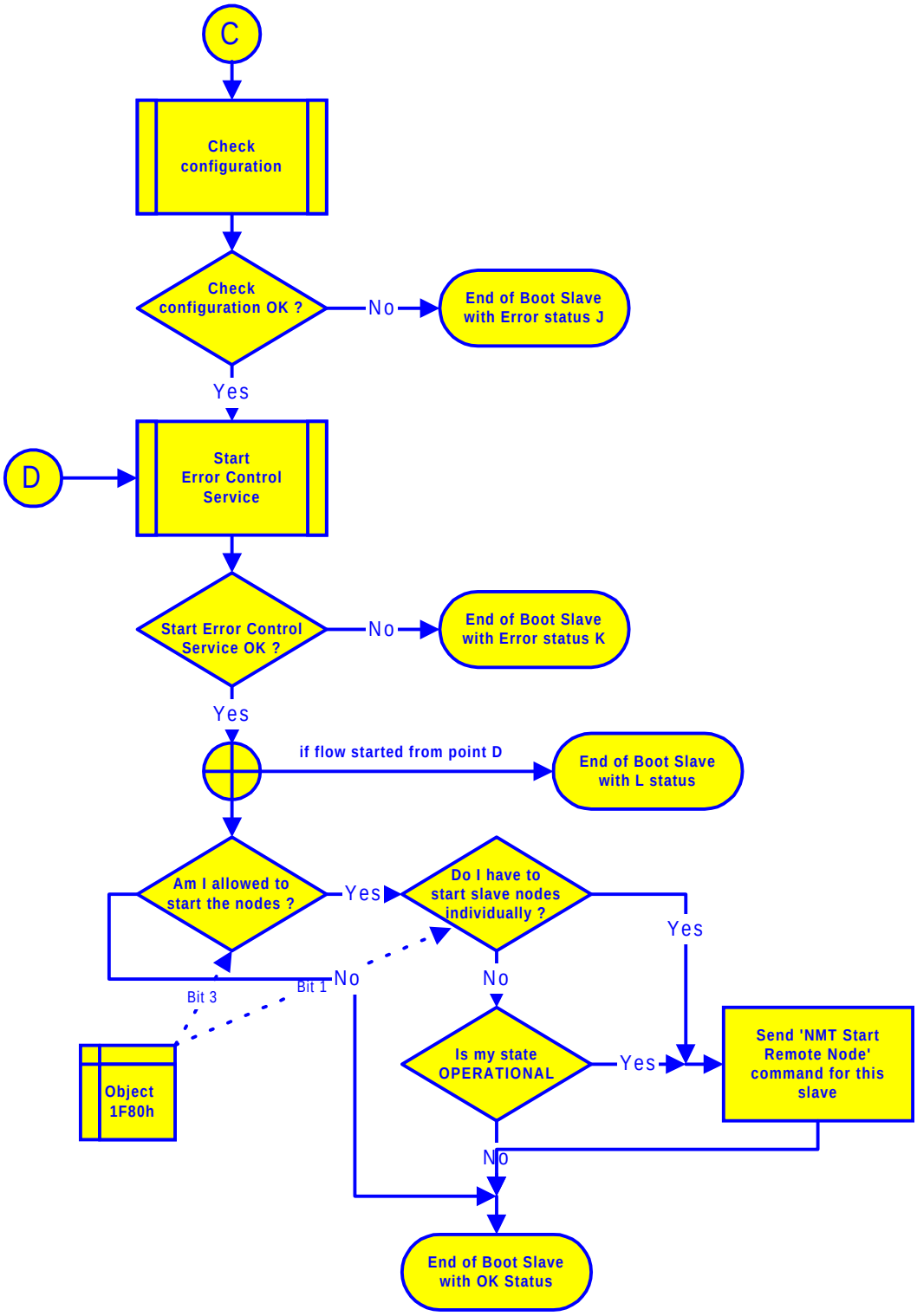


Figure 8 : Boot slave predefined process, part 3

5. Check configuration

The Configuration Management (CMT) is described in detail in chapter 6. It will check the configured value of the Network List object 1F26H and 1F27H. If both values are not 0, it will read the slave's object 1020H Verify Configuration. If the read access fails or the actual values are not equal to the configured values or the objects 1F26H, 1F27H are not implemented, the CMT will start the CMT Download process.

If the CMT Download process fails, the boot-up process continues with the error handler (whose reaction is application specific).

The process flow may skip the Check configuration -step in case Keep alive -feature is implemented and the slave is noticed to be initially Operational. In that case, the flow resumes from connection point D. If the flow resumes from connection point D the situation is informed to the application by returning from the Boot slave -process with an error code. The error code is used by the error handler to give a warning 'Slave X was initially Operational' to the operator.

Check configuration -predefined process is illustrated in Figure 9.

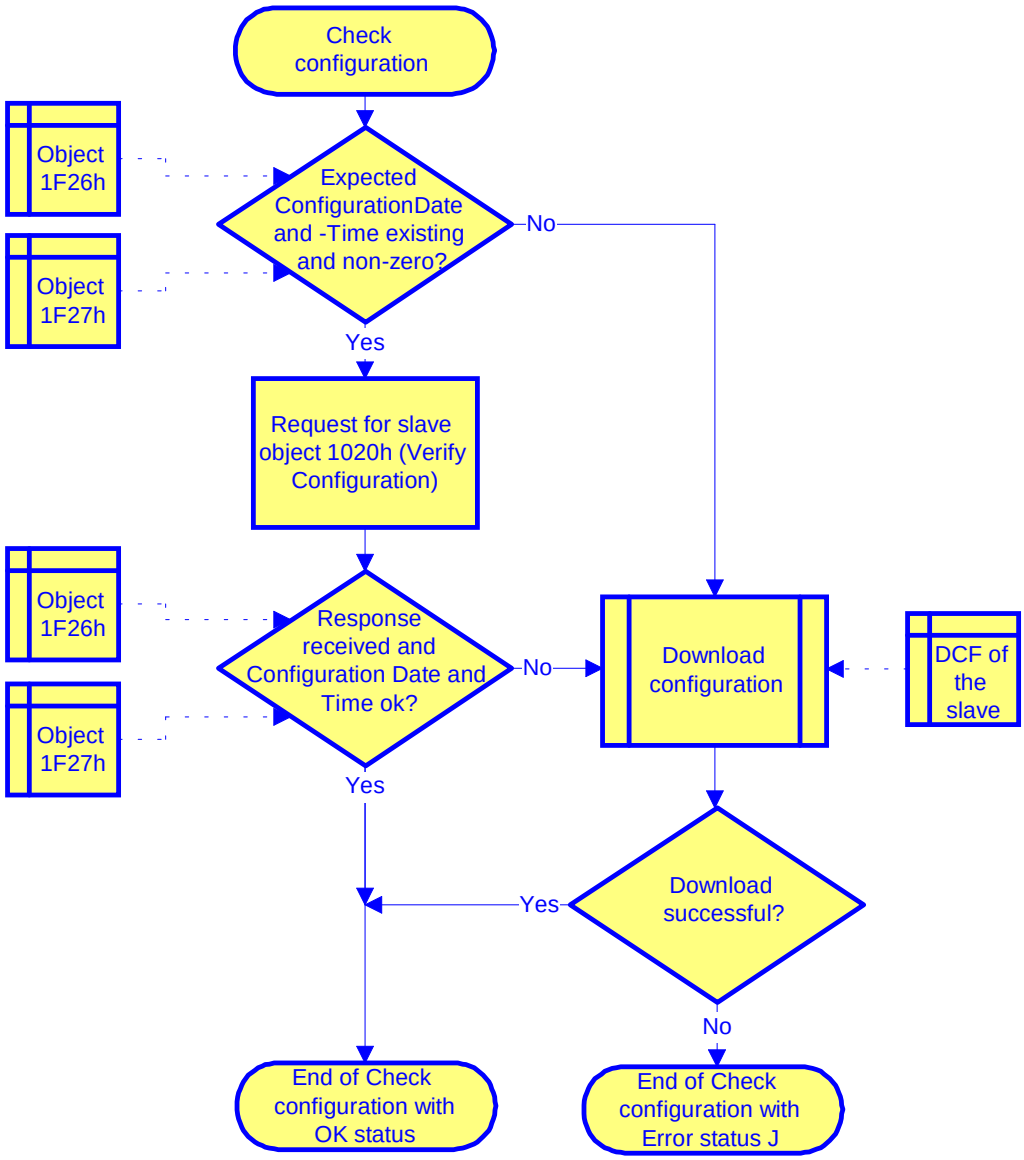


Figure 9: Check configuration -predefined process.
Download configuration -predefined process is not illustrated in this context.

6. Start Error Control Service

The procedure shall support DS-301 V3 devices as well as V4 devices. For this the Master inspects it's configured values in object Consumer Heartbeat Time 1016H and Slave Assignment 1F81H and proceeds as described in chapter 5.2. If the Start Error Control Service -process fails, the boot-up continues with error handler (whose reaction is to restart the Boot Slave Process and further application specific behaviour).

Start Error Control Service -predefined process is illustrated in Figure 11 in chapter 5.3.

7. Starting individual slaves

   If the bit 3 of object 1F80h (see chapter 5.1) allows and if the bit 1 of object 1F80h is configured zero (0), the slave has to be started individually.

   In Normal Operation, after the end of the initial boot-up procedure, the NMT master will not perform anymore the "NMT Start all node" command that may be required according object 1F80h Bit 1. In that case the Boot Slave process has to start individually the node if the NMT master is allowed to start a node. The test whether then NMT master's node is OPERATIONAL allows to know if the initial boot-up procedure has reached its end or not. This case happens when the process is started by the Error Handler (see chapter 5.3) or by the Boot-Up handler (see chapter 5.4) for an optional node with a late boot-up or a new device replacing a defunct one.

Descriptions of the Error status codes showing up in the boot-up procedure flow charts:

Error status	Description
A	The slave no longer exists in the Network list
B	No response on access to Actual Device Type (object 1000h) received
C	Actual Device Type (object 1000h) of the slave node did not match with the expected DeviceTypeIdentification in object 1F84h
D	Actual Vendor ID (object 1018h) of the slave node did not match with the expected Vendor ID in object 1F85h
E	Slave node did not respond with its state during Check node state -process. Slave is a heartbeat producer
F	Slave node did not respond with its state during Check node state -process. Slave is a Node Guard slave (NMT slave)
G	It was requested to verify the application software version, but the expected version date and time values were not configured in objects 1F53h and 1F54h respectively
H	Actual application software version Date or Time (object 1F52h) did not match with the expected date and time values in objects 1F53h and 1F54h respectively. Automatic software update was not allowed
I	Actual application software version Date or Time (object 1027h) did not match with the expected date and time values in objects 1F53h and 1F54h respectively and automatic software update failed
J	Automatic configuration download failed
K	The slave node did not send its heartbeat message during Start Error Control Service although it was reported to be a heartbeat producer (Note! This error situation is illustrated in Figure 11 in chapter 5.3)
L	Slave was initially operational. (CANopen manager may resume operation with other nodes)
M	Actual ProductCode (object 1018h) of the slave node did not match with the expected Product Code in object 1F86H
N	Actual RevisionNumber (object 1018h) of the slave node did not match with the expected RevisionNumber in object 1F87H
O	Actual SerialNumber (object 1018h) of the slave node did not match with the expected SerialNumber in object 1F88H

Application notes:

In principle parameters are just written and not read afterwards; the correct implementation of the SDO protocol must be trusted. This may be changed for safety critical parameters or applications.

It is allowed to boot one device after another or all in parallel.

The defined process gives an overview on the Boot-Up, it shall not define specific API for NMT or other concrete instances. The main purpose is to have some common rules for Master Code and to give Slave Devices the knowledge, what they have to expect on Boot-Up.

The same process will be used, if a slave has to be booted while the rest of the system is already running, e.g. after a Reset or Error Control Event. In that case the NMT Start Node may always be sent individually.

Since nearly all objects and features in this document are optional it is possible to implement very basic NMT Master, which could make sense for some kinds of applications. The most simple boot-up process possible besides self-starting devices is shown in Figure 10.

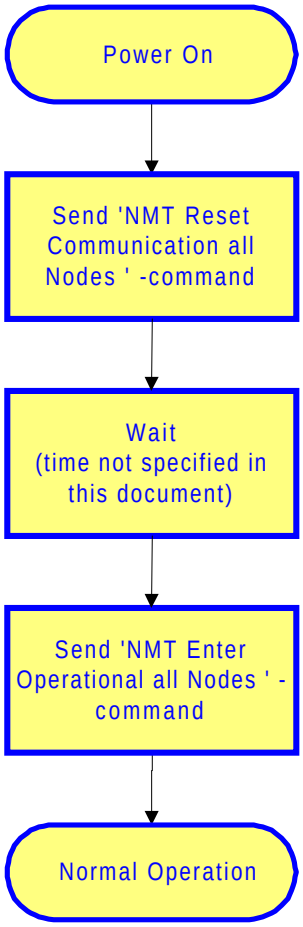


Figure 10: Simplest possible NMT Boot Process

5 NMT Master

The NMT Master provides services for controlling the network behaviour of nodes as defined in DS-301. Only one NMT Master can exist in a CANopen Network. Since there may be several devices that are able to perform the task of an NMT Master, it is necessary to configure this functionality.

5.1 NMT Start-up

Index	Object	Name	Type	Attr.	M/O
1F80H	VAR	NMTStartup	Unsigned32	RW	O

This object configures the start-up behaviour of a device that is able to perform the NMT. The value has the following interpretation:

Bit 0

= 0 Device is NOT the NMT Master. The objects of the Network List have to be ignored. All other bits have to be ignored. Exception for Autostart 0001000b, 0000010b see below.

= 1 Device is the NMT Master.

Bit 1

= 0 Start only explicitly assigned slaves (if Bit 3=0).

= 1 After boot-up perform the service NMT Start Remote Node All Nodes (if Bit 3=0)

Bit 2

= 0 Enter myself automatically Operational

= 1 Do not enter myself Operational automatically. Application will decide, when to enter the state Operational.

Bit 3

= 0 Allow to start up the slaves (i.e. to send NMT Start Remote Node -command)

= 1 Do not allow to send NMT Start Remote Node -command; the application may start the Slaves

Bit 4

= 0 On Error Control Event of a mandatory slave treat the slave individually.

= 1 On Error Control Event of a mandatory slave perform NMT Reset all Nodes (including self). Refer to Bit 6 and 1F81H, Bit 3

Bit 5

= 0 Do not participate Flying Manager Process

= 1 Participate the Flying Manager Process

Bit 6

= 0	On Error Control Event of a mandatory slave treat the slave according to Bit 4
= 1	On Error Control Event of a mandatory slave send NMT Stop all Nodes (including self). Ignore Bit 4.

Bit 7-31 Reserved by CiA, always 0

Bits that control a feature that is not supported by the implementation are RO.

Object 1F80H is a configuration object. Internal state transitions must not change this object.

The system integrator has the responsibility to combine the bits appropriately. The following combinations for configuration have a standardised behaviour:

- NMT Master, Flying Master not supported

Bit	0	1	2	3	4	5	6
Value	1	X	X	X	X	0	X

This is the standard case for CANopen Managers without performing a Flying Master Process. The device performs the Boot-Up as a described with evaluating all the other bits.

- NMT Master supporting Flying Master Process

Bit	0	1	2	3	4	5	6
Value	1	X	X	X	X	1	X

The Bit 0 stays 1 even if the device looses the Flying Master Process. In that case Bit 2 Autostart feature is not evaluated. The device has to store the information that it currently is not the NMT Master internally. If the device gets the Mastership it continues the Boot-Up as a described with evaluating all the other bits.

- Start-up capable Device, Autostart feature only

Bit	0	1	2	3	4	5	6
Value	0	0	0	1	0	0	0

Devices that are **not** implementing an NMT Master, but shall enter Operational mode automatically, shall implement object 1F80H. This feature can be configured with setting Bit 2 to 0 whilst Bit 0,1 are set to 0. Bit 3 is 1 due to the not implemented NMT. Bit 4 is ignored.

- Start-up capable Device, NMT Start command implemented

Bit	0	1	2	3	4	5	6
Value	0	1	0	0	0	0	0

Devices that are **not** implementing a complete NMT Master, but shall enter Operational mode automatically **and** shall transmit the 'NMT Start all Nodes' command, shall implement object 1F80H. This feature can be configured with setting Bit 2 to 0 whilst Bit 0,3 are set to 0 and Bit 1 to 1. Bit 4 is ignored.

Index	Object	Name	Type	Attr.	M/O
1F89H	VAR	BootTime	Unsigned32	RW	O

This object describes the maximum time in ms, the master will wait for all mandatory slaves before signalling an error. If the time is zero (0), it will wait endlessly. The Default Value is 0.

5.2 Network List

The objects defined in this chapter are only valid and allowed to perform NMT actions, if object 1F80H configures this device as NMT Master.

The Network List consists of some objects, that give information which slaves have to be managed, how they have to be booted and about requested actions on Error Control events.

Index	Object	Name	Type	Attr.	M/O
1F81H	ARRAY	SlaveAssignment	Unsigned32	RW	O

This object assigns slaves to the NMT Master. It gives information about Error Control Parameters and about actions to be performed on Error Control Events. All other parameters according to a slave in the Network List are only valid, if the slave is assigned to the SlaveAssignment 1F81H.

Sub-Index 0 has the value 127.

Each sub-index in the array corresponds to the slave with the Node ID equal to the sub-index. The sub-index equal the Master's Node-ID is ignored.

Byte 0

Bit 0	= 0	Node with this ID is not a slave
	= 1	Node with this ID is a slave. After configuration (with Configuration Manager) the Node will be set to state Operational.
Bit 1		reserved for compatibility
Bit 2	= 0	On Error Control Event or other detection of a booting slave inform the application but do NOT automatically configure and start the slave.
	= 1	On Error Control Event or other detection of a booting slave inform the application and do start the process "Start Boot Slave".
Bit 3	= 0	Optional Slave: Network may be started even if this node could not be contacted.
	= 1	Mandatory slave: Network must not be started if this slave node could not be contacted during the boot slave procedure.

	Bit 4	= 0	The slave node may be reset with NMT Reset Communication -command independent of its state. Hence, no checking of its state has to be executed prior to NMT Reset Communication -command.
		= 1	NMT Master must not send NMT Reset Communication for this node if it notices the slave to be in Operational state. This is noticed by waiting for the heartbeat message or sending RTR for the node guard message.
	Bit 5	= 0	Application software version verification for this node is not required
		= 1	Application software version verification for this node is required
	Bit 6	= 0	Automatic application software update (download) is not allowed
		= 1	Automatic application software update (download) is allowed
	Bit 7		reserved by CiA (always 0)
Byte 1			8 Bit Value for the RetryFactor
Byte 2,3			16 Bit Value for the GuardTime

Bits that control a feature that is not supported by the implementation are RO.

If the answer on a Guard RTR is missing the NMT Master will retry the request (RetryFactor-1) times each with an interval of GuardTime. Guarding will be performed only if non-zero values are entered for RetryFactor and GuardTime. For more details concerning Error Control see chapter 5.3.

Index	Object	Name	Type	Attr.	M/O
1F84H	ARRAY	DeviceTypeIdentification	Unsigned32	RW	O

This object allows to enter values for expected device types. Sub-Index 0 has the value 127. Each Sub-Index in the array corresponds to the slave with the Node ID equal to the Sub-Index. The Sub-Index equal to the Master's Node-ID is ignored. On Boot-Up the Master reads object 1000H of each assigned slave. If the value in DeviceTypeIdentification is 0, this read access only gives information about the principle existence of a device with this Node-ID. If the value is not 0, it is compared against the value read from the device and the boot-up for that device is only continued on exact equality. For multi-device-modules the application may perform additional checks.

Index	Object	Name	Type	Attr.	M/O
1F85H	ARRAY	VendorIdentification	Unsigned32	RW	O

This object allows to enter values for expected Vendor IDs. Sub-Index 0 has the value 127. Each Sub-Index in the array corresponds to the slave with the Node ID equal to the Sub-Index. The Sub-Index equal to the Master's Node-ID is ignored. If a value in this array is not 0, on boot-up procedure the Master will read object 1018H, Sub-Index 1 of the device and compare it to the actual value. The Boot-Up for that device is only continued on exact equality. The application may perform additional checks – e.g. for serial numbers.

Index	Object	Name	Type	Attr.	M/O
1F86H	ARRAY	ProductCode	Unsigned32	RW	O

1F86h ProductCode : List of expected ProductCode (Object 1018h Sub-Index 2h) for each expected slave device. Sub-Index 0 has the value 127. Each sub-index in the array correspond to the slave with the Node-ID equal to the sub-index. If a value in this array is not 0, on Boot-Up procedure the Master will read object 1018H, Sub-Index 2 of the device and compare it to the actual value, and the Boot-Up for that device is only continued on exact equality.

Index	Object	Name	Type	Attr.	M/O
1F87H	ARRAY	RevisionNumber	Unsigned32	RW	O

1F87h RevisionNumber : List of expected RevisionNumber (Object 1018h Sub-Index 3h) for each expected slave device. Sub-Index 0 has the value 127. Each sub-index in the array correspond to the slave with the Node-ID equal to the sub-index. If a value in this array is not 0, on Boot-Up procedure the Master will read object 1018H, Sub-Index 3 of the device and compare it to the actual value. As revision number include major and minor revision, the Boot-Up for that device is only continued on "exact mach" on the major revision and "greater than" on minor revision number.

Index	Object	Name	Type	Attr.	M/O
1F88H	ARRAY	SerialNumber	Unsigned32	RW	O

1F88h SerialNumber : List of expected SerialNumber (Object 1018h Sub-Index 4h) for each expected slave device. Sub-Index 0 has the value 127. Each sub-index in the array correspond to the slave with the Node-ID equal to the sub-index. If a value in this array is not 0, on Boot-Up procedure the Master will read object 1018H, Sub-Index 4 of the device and compare it to the actual value, and the Boot-Up for that device is only continued on exact equality

5.3 Error Control

The NMT Master has to be able to manage DS-301 V3 devices as well as V4 devices. In particular this requires a mechanism to decide, if Guarding or Heartbeat has to be used for Error Control. A further requirement is the compatibility of the objects 1F80H-1F83H from DS-302 V2.0.

The mechanism cannot rely only on DS-301 V4 specified Heartbeat Consumer behaviour, since this defines Heartbeat Consuming in a way, that it is only monitored, when the Heartbeat Producer really started the Heartbeat. For an NMT Master ’s point of view it is highly desirable to detect, if a Heartbeat Producer does not start within a certain amount of time. For this reason for the NMT Master it is specified that

> Heartbeat Consuming Time-Out checking immediately starts with the process ‘Start Error Control Service’. The first time-out equals the value of Heartbeat Consumer Time in object 1016h.

Start Error Control Service -predefined process is illustrated in Figure 11.

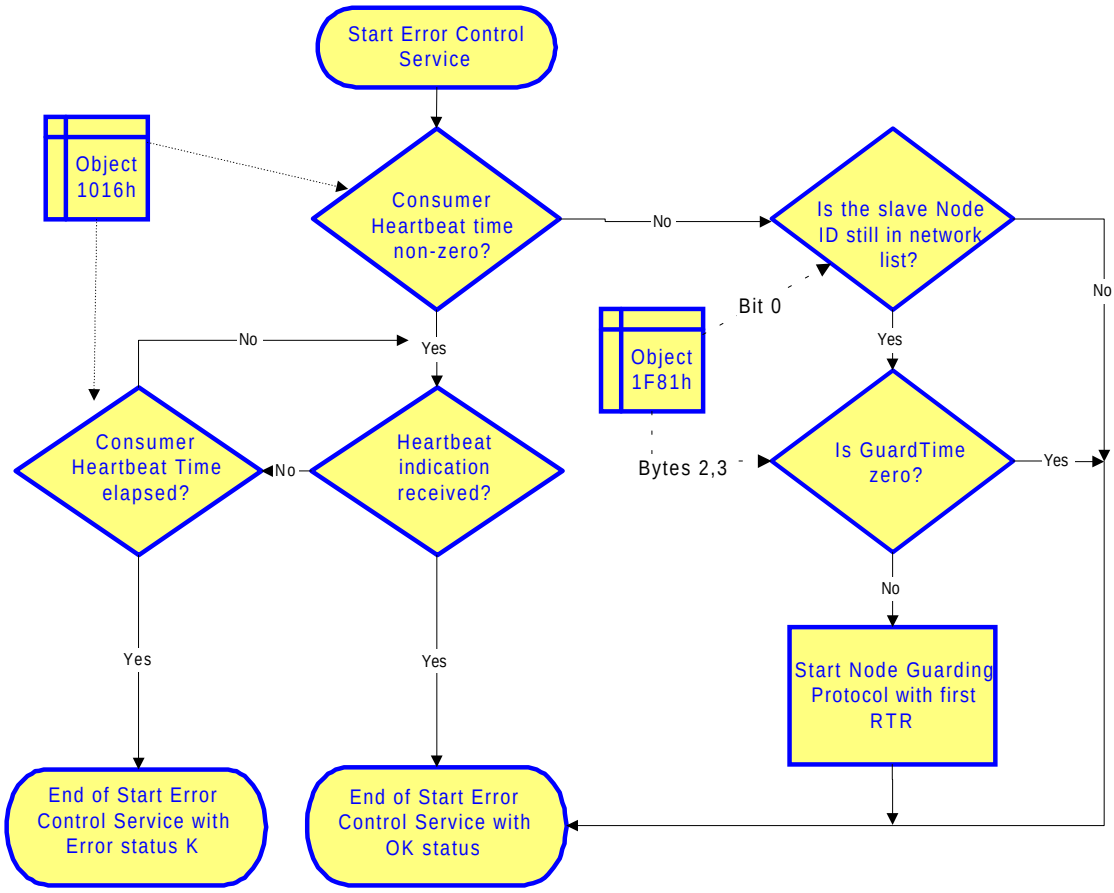


Figure 11: Start Error Control Service -predefined process.

If the Error Control creates an Error Event (refer to /1/), the Start Boot Slave Process will be re-started and further application specific behaviour may be activated. This is illustrated in Figure 12.

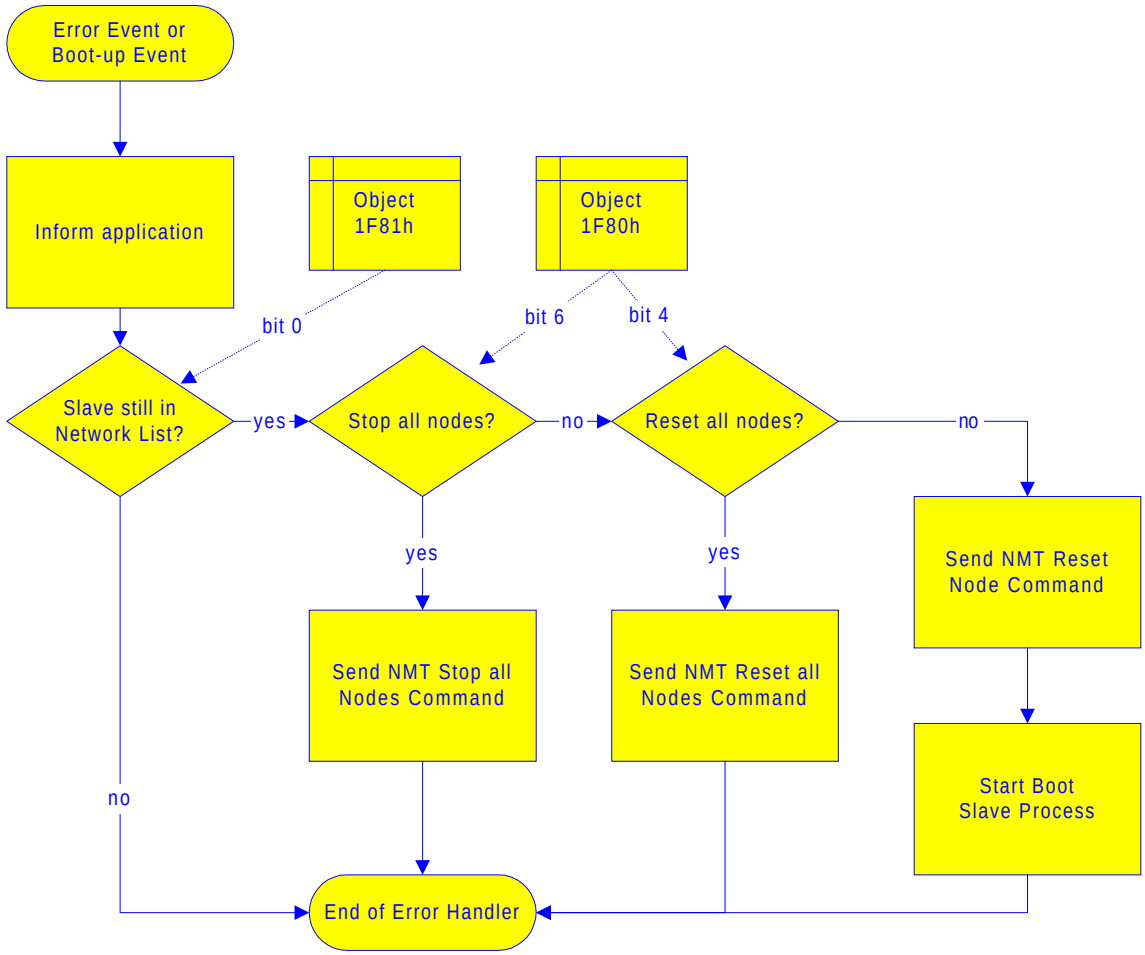


Figure 12: Error Handler

5.4 Request NMT

Only the NMT Master is allowed to perform NMT services on the Network. Other devices (such as a configuration tool) are not allowed to perform NMT service for two reasons:

- CAN forbids the transmission of the same CAN identifier (except RTR) for more than one device.
- If another device than the NMT Master changes the state of an NMT Slave and the NMT Master guards that Slave, the NMT Master will recognize a Guard Error.

The problem is solved, if the device that wants to change the state of another device requests this action at the NMT Master. A further advantage of this mechanism is, that e.g. an IEC61131 application can request NMT actions by writing to its local object dictionary.

Index	Object	Name	Type	Attr.	M/O
1F82H	ARRAY	RequestNMT	Unsigned8	Sub 0: RO Sub 1-127: RW Sub 128: WO	O

Sub-Index 0 NrOfSupportedObjects has the RO value 128.

Sub-Index i (with i=1..127): Request NMT Service for the Slave with Node ID i.

Sub-Index 128: Request NMT Service for all Nodes.

On write access to this object the value is the requested node state. On read access this object reports the actual node state.

State	Value on write access	Value on read access
Stopped	4	4
Operational	5	5
ResetNode	6	-
ResetCommunication	7	-
PreOperational	127	127
not known	-	0
node missing	-	1

These commands can also be applied to the Master itself. A RequestNMT to all nodes will also apply for the Master itself.

Application hint: Network Tools need the possibility of performing NMT services at least for networks where no CANopen Manager exists. It is even useful for development purposes to have this feature available in networks with a Configuration Manager. To allow this, the user of such a Tool should have the possibility to configure, if the Tool performs direct NMT or via Request NMT.

Index	Object	Name	Type	Attr.	M/O
1F83H	ARRAY	RequestGuarding	Unsigned8	Sub 0: RO Sub 1-127: RW Sub 128: WO	O

Sub-Index 0 NrOfSupportedObjects has the RO value 128.

Sub-Index i (with i=1..127): Request Guarding for the Slave with Node ID i.

Value	Write Access	Read Access
1	Start Guarding	Slave actually is guarded
0	Stop Guarding	Slave actually is not guarded

Sub-Index 128: Request Start/Stop Guarding for all Nodes.

For activation of Heartbeat Protocol, there is no special object required. One side has to be configured as Heartbeat Producer, the other as Heartbeat Consumer. To obtain the information, if the Master actually performs the Heartbeat Monitoring, the object 1F82H may be read. Any other value than 0 shows a running Heartbeat.

These objects are only allowed to perform NMT actions, if object 1F80H configures this device as NMT Master.

Application hints:

Object 1F81H defines reaction of the NMT Master on Guard Errors. If this is configured not to perform automatically recovery, the application uses objects 1F83H, 1F25H (see Configuration Manager) and 1F82H to “manually” re-boot the slave.

The objects defined in this chapter are volatile. They are not stored by Save Command 1010H.

When the NMT State of a Slave is changed via 1F82H, no automatic re-boot shall appear. Otherwise a system administrator never had the chance to get the slave into a stable Pre-Operational state.

5.5 Flying Master

There are several functions within a CANopen network that have to be performed by only one supervising instance. Functions of this type are:

- NMT Master (System-Startup)
- SDO Manager (optional)
- Configuration Manager (optional) (System Configuration)

When the function of the NMT Master is of vital necessity for a system, it shall be guaranteed that after a failure of the actual NMT Master another NMT Master automatically takes over the function of the failed NMT Master.

In the following the terms NMT Master and CANopen Manager can be exchanged, since the described process itself does not depend on the SDO Manager and Configuration Manager. Anyhow, once an NMT Master is determined, the possibly existing SDO Manager on this device is getting the active SDO Manager and the possibly existing Configuration Manager is getting the active Configuration Manager. Changing SDO/Configuration Managers imply the question of their concrete behaviour. This is well-defined without further specifications, since in any case on a NMT Master change all devices are performing a reset and the SDO/Configuration Managers get active only if the NMT Master residing on the same device is getting active.

The process of NMT Master Determination is called *Flying Master Process*. In Figure 13 the basic principle of the Flying Master Process is shown. This process is to be performed by any NMT Master -capable device, having Bit 1 and Bit 5 of 1F80H set. This process is embedded in the general boot-up process as described in chapter 4.

First the device checks, if there is already an NMT Master active by means of the ”Detection of active NMT Master protocol” (Section 5.5.1.1). If no NMT Master is already active in the system an NMT Master negotiation process is performed, which normally ends with sending the master identification of the newly initialised device. If in the system already an NMT Master is active and the priority of the active NMT Master is less than the new device, then a new NMT Master negotiation is forced (Section 5.5.2). If the priority of the new device is less than the already active NMT Master, only the slave application is started.

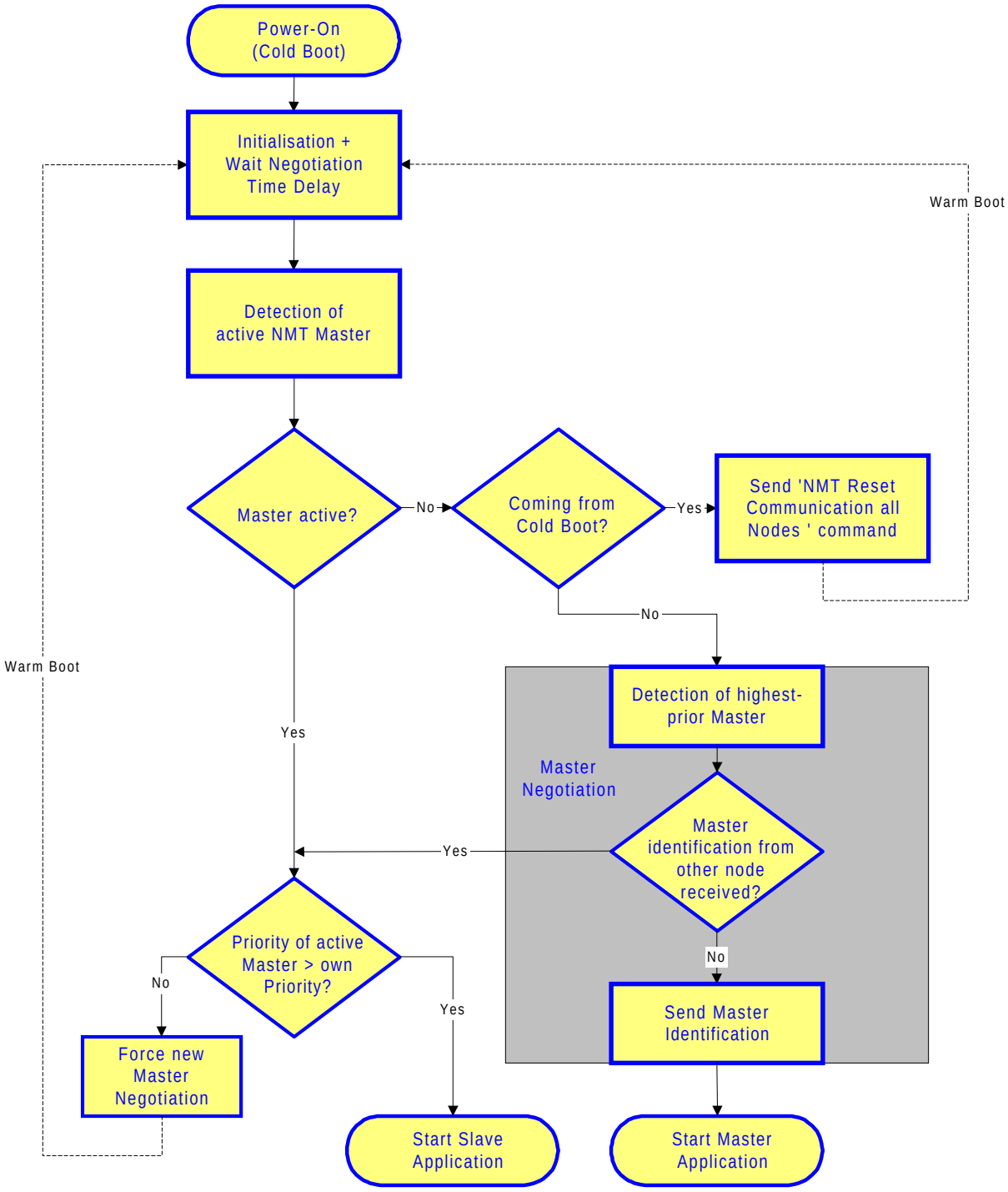


Figure 13: Flying Master Process Overview

5.5.1 Protocols for NMT Master-Capable Devices

5.5.1.1 Detection of Active NMT Master Protocol

With this protocol (Figure 14) an NMT Master -capable device can request the active NMT Master for its ” NMT Master Priority Level”. There are three priority levels defined, where 0 is the highest priority level and 2 is the lowest.

The reply from the active NMT Master must be provided within a configurable period of time. This time value is specified in Object 1F90h/01 of the Flying Master Timing Parameters.

Reserved Identifiers

Request of NMT Master priority level: ID = 73h (no data)

Reply from the active NMT Master: ID = 71h

Data Byte [0] ... NMT Master priority level

Data Byte [1] ... Node ID

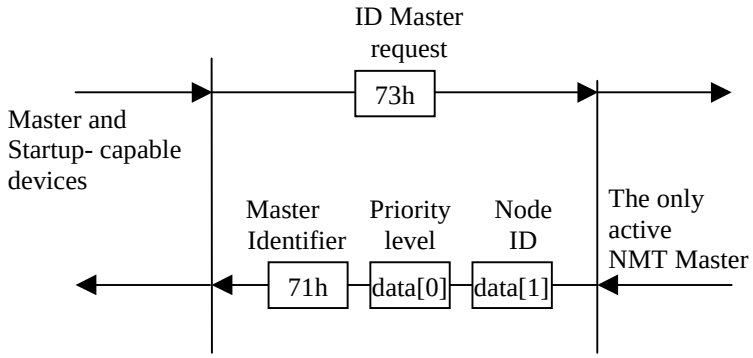


Figure 14: Detection of Active NMT Master Protocol

5.5.1.2 Preparation of Master Negotiation

The protocol "NMT Master Negotiation" (Chapter 5.5.1.3, Figure 15) is used for starting an NMT Master determination process when no active NMT Master is in the network. This also takes place, when a lower-prior Master has been kicked by a "Force new Master Negotiation".

If the boot-up came from Power-On (Cold Boot) and no Master is active, it has to be ensured that PDO traffic does not disturb the timing, which is described below. Furthermore it has to be ensured, that instances of SDO Manager and other network services have to be reset properly. Therefore all NMT Master -capable devices have to perform an "NMT Reset Communication" Command. This will force them all again to fall back to Initialisation, but with "Warm Boot".

The "NMT Reset Communication" has to reset also the transmitters internal state machine and object dictionary.

For compensation of different initialisation times the devices wait the NMT Master Negotiation Time Delay (Object 1F90h/02).

5.5.1.3 NMT Master Negotiation Protocol

NMT Master -capable devices transmit the ”ID trigger timeslot” command for starting and synchronising their timers. When a device receives the ”ID trigger timeslot” command there is no need to transmit also the ”ID trigger timeslot” to prevent high busload. If the device receives again an "ID trigger timeslot" the timers have to be restarted.

Each device has a unique waiting period based on its node ID and priority group. The device with the lowest waiting period will transmit the Master identification first (Figure 16).

The waiting period is determined according to the following rule:

Waiting_time_after_trigger = (Priority_Level * Priority_Time_Slot) + (Node ID * Node_Time_Slot)

with ”Priority_Time_Slot” and ”Node_Time_Slot” being configurable basic time intervals. For the priority time slot the following rules must be considered:

Priority_Time_Slot > 127 * Node_Time_Slot

The priority level values are according to Table 5-1: Priority level values

Priority level	Description
0	High
1	Medium
2	Lowest

Table 5-1: Priority level values

The timing parameter of the NMT Master negotiation protocol are specified in the object dictionary under the following indices:

Priority_Level Object 1F90h/03

Priority_Time_Slot Object 1F90h/04

Node_Time_Slot Object 1F90h/05

Reserved Identifiers

Time Slot Trigger command: ID = 72h (no data)

Reply from a Master-capable device: ID = 71h:

Data Byte [0] NMT Master priority level

Data Byte [1] Node ID

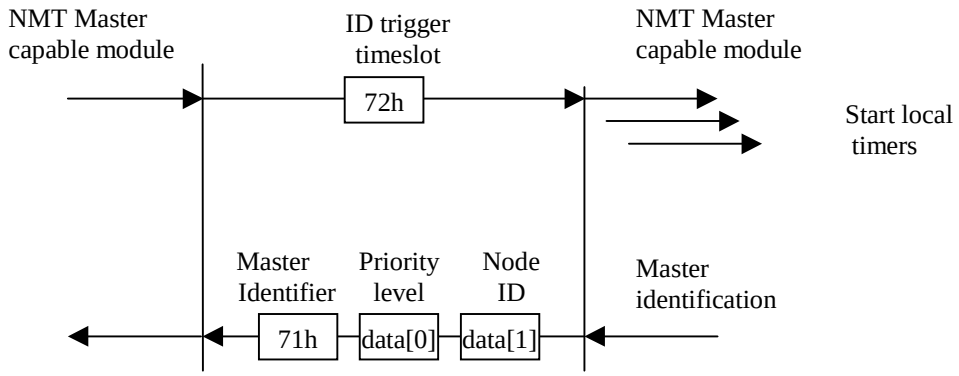


Figure 15: NMT Master Negotiation Protocol

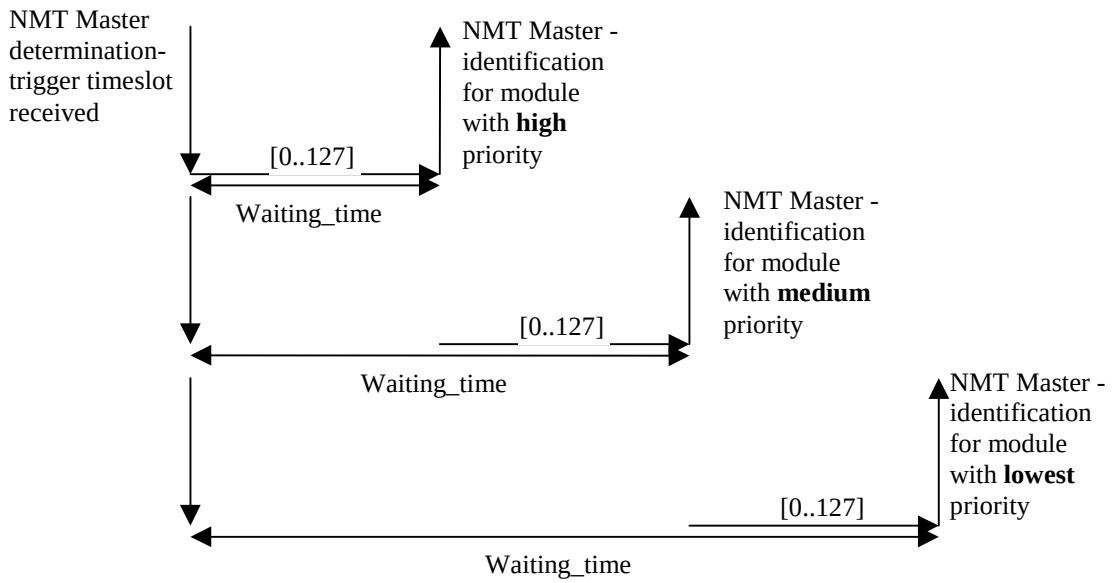


Figure 16: Waiting Period after Reception of the Trigger Time Slot Command

5.5.2 Forcing a New NMT Master Negotiation Protocol

This protocol is used for starting a new NMT Master determination process when

- During the master determination process (Section 5.5.1.1) an active NMT Master of lower priority is detected or
- multiple active NMT Masters are detected (Section 5.5.4.2)

To force a new NMT Master negotiation the ”Force NMT Reset Communication Command” is transmitted by the device which requests the new NMT Master negotiation.

Reserved Identifiers

Request of Force NMT Reset Communication Command

: ID = 76h (no data)

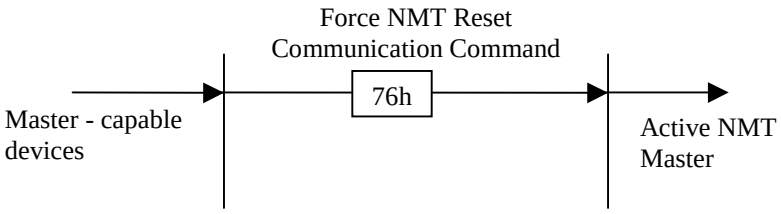


Figure 17: Forcing a New NMT Master Negotiation Protocol

This forces the active Master to transmit the NMT Reset Communication All Nodes Command. It has to apply this reset also to his own state machine and own object dictionary. Afterwards it will follow again the standard entry into the procedure as warm boot-up.

Implementation hint: To avoid a deadlock of the system, an NMT Master should not force a further NMT Master negotiation when it lost the first negotiation and a node of lower priority again got the Mastership (this could be caused by wrong configured timing parameters.

5.5.3 Protocol for Startup-Capable Devices

Startup-Capable Devices are devices that are able to start without an NMT Master. They only can enter Operational Mode automatically and optionally set a group of nodes to operational by sending an NMT Start all nodes command. If this feature is configurable the device has to implement the appropriate bits of object 1F80H.

5.5.3.1 Detection of NMT Master-Capable Devices Protocol

With this protocol a startup-capable device can check after ”Power On” if there is at least one NMT Master -capable device in the network.

Before starting the protocol the device must wait a configurable time delay (Object 1F91h/02 of Startup Timing Parameters) after initialization. With a reserved identifier data frame with no data the device asks for NMT Master -capable devices. Any device with NMT Master capability will reply with a reserved identifier frame and zero data field. (Figure 18).

If one or more devices answer within a configurable period of time (Object 1F91h/01 of timing parameters), the requesting device knows that there exist at least one device with the capability to become NMT Master.

If there is no response from any device, the requesting device will send the NMT command ”Start Network”. To avoid several start commands when a group of startup-capable nodes of the same type is switched on, the devices have a node-ID-based waiting period which is calculated by:

Waiting_Time = Node_ID * Node_Time_Slot

(with Node_Time_Slot value at 1F91h/03) If this feature is supported, it must be guaranteed that multiple start network commands are ignored by already started nodes.

Reserved Identifiers

Request of NMT Master -capable devices: ID = 75h (no data)

Reply from an NMT Master -capable device: ID = 74h (no data)

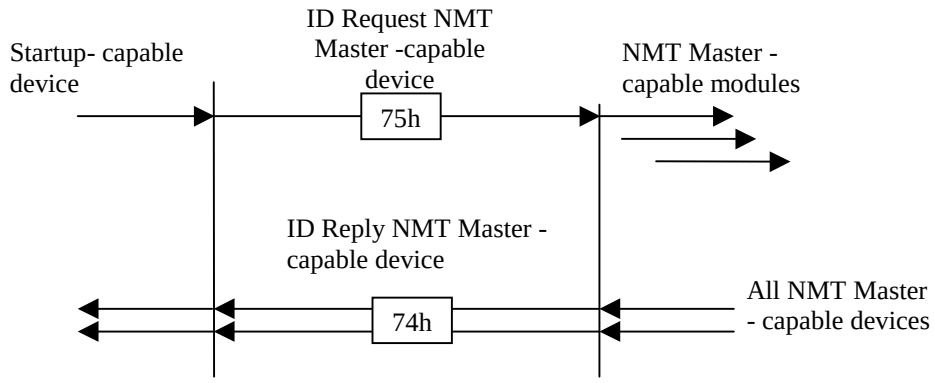


Figure 18: Detection of NMT Master Capable Devices

5.5.4 Detection of Failures

In a CANopen network only one active NMT Master is allowed. Due to failure conditions problems can occur which shall be resolved automatically by the following measures:

- When the active NMT Master fails, a backup NMT Master must detect the loss of the active NMT Master and correct the situation.
- When two or more NMT Master become active at the same time this must be detected by the NMT Masters and corrected. This situation can for example happen when the CAN-line was temporarily split, causing two NMT Masters becoming active.

5.5.4.1 Detection of an NMT Master Failure

To detect the loss of the active NMT Master, every NMT Master -capable device needs to monitor the heartbeat of the active NMT Master. Therefore the node Id of the active NMT Master is captured during the ”Detection of Active NMT Master Protocol”.

To solve the problem, the Master negotiation must be initiated. This shall be done by sending an "NMT Reset Communication" command to all devices.

Remark: It is not possible to use the the ”Force New NMT Master Negotiation Protocol”, since there is no Master that could perform the Reset command.

Implementation remark: When an NMT Master supports the ”Heartbeat Consumer” entry (1016h), all NMT Master capable devices must be configured by a configuration tool in this entry. When an NMT Master doesn’t support this entry, the NMT Master must automatically configure the internal receive object for reception of the heartbeat message of the active NMT Master.

5.5.4.2 Detection of Multiple NMT Masters

Active NMT Masters transmit cyclically the ”Forcing New NMT Master Negotiation Protocol” - without the sender forcing itself to send "NMT Reset Communication". The cycle time period can be configured in object 1F90h/06 of the flying master timing parameters.

If multiple NMT Masters exist, they will be forced into a reset by this mechanism. The very rarely case that both (or more) send the message at the same time will have the effect that they do not see it as received. In order to resolve this situation the cycle time periods of all NMT Master shall be configured with different values.

5.5.4.3 System Design

The mechanism to use messages of same ID without data works only if there are devices that do not send these messages. Therefore it is not allowed to build a system that consists only of Flying Master capable devices.

A network performing the Flying Master Process must not use state-critical nodes and vice-versa.

5.5.4.4 Physical Line disturbances

If there are error frames on the bus, they may shift the time slots. In the worst case there can be overlaps of the messages, which will lead to a total bus shutdown. The implementation is responsible to detect this situation and implement appropriate fallbacks.

5.5.5 Disabling of the Flying Master

When an NMT Master -capable device shall be used as normal node in a network with an NMT Master which do not support the flying master procedure, the flying master-mechanism must be disabled. This is possible by clearing object 1F80H.

5.5.6 Object Dictionary Entries

Object 1F90H Flying Master Timing Parameters

Object Description

Index	**1F90H**
Name	Flying Master Timing Parameters
Object code	ARRAY
Data Type	UNSIGNED16
Category	Optional; Mandatory for devices supporting the flying master mechanism

Entry Description

Sub index	**0h**
Description	Number of entries
Entry category	Mandatory
Access	ro
PDO mapping	No
Value range	UNSIGNED8: 6
Default Value	6

Sub index	**1h**
Description	Timeout for Detection of an active NMT Master
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	100 (ms)

Sub index	**2h**
Description	NMT Master negotiation time delay
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	500 (ms)

Sub index	**3h**
Description	Master Priority Level
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	0h-2h
Default value	No

Sub index	**4h**
Description	Priority Time Slot
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	1500 (ms)

Sub index	**5h**
Description	Node Time Slot
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	10 (ms)

Sub index	**6h**
Description	Multiple Master Detect Cycle Time
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	4000 + Node_ID * 10 (ms)

Object 1F91H Startup-Capable Device Timing Parameters

Object Description

Index	**1F91H**
Name	Startup-Capable Device Timing Parameters
Object code	ARRAY
Data Type	UNSIGNED16
Category	Optional; Mandatory for startup-capable devices

Entry description

Sub index	**0h**
Description	Number of entries
Entry category	Mandatory
Access	ro
PDO mapping	No
Value range	UNSIGNED8: 3
Default Value	3

Sub index	**1h**
Description	Timeout for Detection of an NMT Master - capable device
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	100 (ms)

Sub index	**2h**
Description	Delay Time for an NMT Master-Capable Device Request
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	500 (ms)

Sub index	**3h**
Description	Node Time Slot
Entry category	Mandatory
Access	rw
PDO mapping	No
Value range	UNSIGNED16
Default value	15 (ms)

Application note: The timing values are calculated for a system running at 125 kBit/s. For other baudrates the devices have to be configured with appropriate time values.

6 Configuration Manager

The Configuration Manager has the task of configuring all network devices at the network boot-up. For this it has to know all the (application dependent) parameter values. This information is set-up in the Device Configuration File DCF of each device.

Most often the set-up of the DCF will be done on a computer station (PC or Workstation). Afterwards the DCF has to be transferred to the Configuration Manager. If this is located on the same Computer, the file(s) may be transferred locally by passing the file name(s). If the Configuration Manager is on another device in the network, the transfer has to be done via CANopen.

In practice we find two different systems. On the one side we have systems with some kind of disk access. They are able to download/upload a DCF file as is. On the other side we have systems without disks and only low memory resources. They require the DCF information in a very compressed form. For meeting those requirements both methods are defined. If the Configuration Manager does not have disk based storage but does have sufficient non-volatile memory, it is recommended to perform the file transfer option at least on a logical level.

The Configuration Manager is active only if the NMT Master residing on the same device is active. Refer to chapters 5.1 and 5.5. If it is not active, the object dictionary entries can be configured, but after a reset it shall not use them for configuring the slaves.

6.1 DCF storage

Index	Object	Name	Type	Attr.	M/O
1F20H	ARRAY	Store DCF	domain	RW	O
1F21H	ARRAY	Storage format	Unsigned 8	RW	O

Index 1F20H, sub-index 0 describes the number of entries. This is equal to the maximum possible Node-ID (127). Each sub-index points to the Node-ID of the device, for which the DCF belongs.

Uploading the DCF of a device from a Configuration Tool to the Configuration Manager is done by writing the DCF file as a domain to the object 1F20H with the sub-index equal to the devices Node-ID.

Downloading the DCF of a device from the Configuration Manager to a Tool is done by reading the object 1F20H with the sub-index equal to the devices Node-ID.

The filename does not need to be stored since every DCF contains its own filename.

Object 1F21H describes the format of the storage. This allows the usage of compressed formats.

Value	Format
0	ASCII, not compressed
1-255	reserved

The device may always store the file compressed internally. The object describes the external behaviour.

If no data had been stored, an SDO Read Request to 1F20H or 1F21H is aborted with the error code 08000024H "Data set empty"

6.2 Concise configuration storage

The concise device configuration does not contain every information of the DCF. It is recommended to use this if the complete DCF storage is not possible.

The information to be stored consists of the parameter values of the object dictionary entries.

Index	Object	Name	Type	Attr.	M/O
1F22H	ARRAY	Concise DCF	Domain	RW	O

Sub-index 0 describes the number of entries. This is equal to the maximum possible Node-ID (127). Each sub-index points to the Node-ID of the device, to which the configuration belongs.

The content is a stream with the following structure:

Number of supported entries	Unsigned32
Index 1	Unsigned16
Sub-index 1	Unsigned8
Data size of parameter 1	Unsigned32
Data of parameter 1	Domain
Index 2	Unsigned16
Sub-index 2	Unsigned8
Data size of parameter 2	Unsigned32
Data of parameter 2	Domain
....	
Index n	Unsigned16
Sub-index n	Unsigned8
Data size of parameter n	Unsigned32
Data of parameter n	Domain

The Data Size is counting bytes (i.e. Unsigned16 has size 2; size of Boolean is given as 1).

Uploading the configuration of a device from a Configuration Tool to the Configuration Manager is done by writing the stream to the object 1F22H with the sub-index equal to the devices Node-ID.

Downloading the configuration of a device from the Configuration Manager to a Tool is done by reading the object 1F22H with the sub-index equal to the devices Node-ID.

Application hint:

The generation of the concise data stream has to consider the rules according to DS-301. Especially the generation of PDO parameters requires an appropriate ordering of the objects:

1. Delete the PDO by setting Bit 31 to 1 (if it is not readonly)
2. Configure Sub-Indexes 2 upwards of the Communication Parameters (may be step 4)
3. Deactivate mapping by setting NrOfMappedObjects to 0
4. Configure the mapping entries
5. Create the PDO by writing a valid COB-ID with the Bit 31 reset to 0.

Application hint:

1. An empty data set can be written by the following concise stream:

Number of supported entries	0 (Unsigned32)

2. If no data has been stored, an SDO Read Request will return a valid concise stream with the following content:

Number of supported entries	0 (Unsigned32)

6.3 Check configuration process

DS-301 defines the object 1020H Verify Configuration. If a device supports the saving of parameters in non volatile memory, a network configuration tool or a CANopen manager can use this object to verify the configuration after a devices reset and to check if a reconfiguration is necessary. The configuration tool has to store the date and time in that object and has to store the same values in the DCF. Now the configuration tool lets the device save its configuration by writing to index 1010H Sub-Index 1 the signature "save". After a reset the device restores the last configuration and the signature automatically or by request. If any other command changes boot-up configuration values, the device has to reset the object Verify Configuration to 0.

The Configuration Manager compares signature and configuration with the value from the DCF and decides if a reconfiguration is necessary or not. The comparison values are stored on the Configuration Manager in the objects

Index	Object	Name	Type	Attr.	M/O
1F26H	ARRAY	ExpectedConfigurationDate	Unsigned32	RW	O
1F27H	ARRAY	ExpectedConfigurationTime	Unsigned32	RW	O

Sub-Index 0 NrOfSupportedObjects has the RO value 127.

Sub-Index i (with i=1..127, i != Node-ID of Configuration Manager): Stores the Configuration date/time of the slave that has Node-ID i.

The usage of Check Configuration is described in chapter 4, specially in Figure 9.

Application hint: The usage of this object allows a significant speed-up of the boot-up process. If it is used, the system integrator has to consider the that a user may change a configuration value and afterwards activate the command store configuration 1010H without changing the value of 1020H. So the system integrator has to ensure a 100% consequent usage of this feature. It should be a feature of configuration tools to force or at least encourage a correct usage of object 1020H.

6.4 Request Configuration

In applications there might be situations, where it is necessary to configure the slaves while runtime. An example is, that a slave has fallen down and re-boots. The NMT-Master will recognize this and will inform the application (see chapter 4, 5). With the object Configure Slave the application is able to tell the Configuration Manager, that it shall configure that slave.

Another example is the connection of a new machine part with several devices. The application needs a possibility to start the Configuration Manager at least for the new nodes.

Index	Object	Name	Type	Attr.	M/O
1F25H	ARRAY	ConfigureSlave	Unsigned32	Sub 0: RO Sub 1-128: WO	O

Sub-Index 0 NrOfSupportedObjects has the RO value 128.

Sub-Index i (with i=1..127): Request re-configuration for the Slave with Node ID i.

Sub-Index 128: Request re-configuration for all Nodes.

To avoid accidental access, the signature ‘conf’ (this equals the Unsigned32 number 0x666E6F63) has to be written to initiate the process.

If no data had been stored, an SDO Write Request to 1-127 is aborted with the error code 08000024H "Data set empty". An SDO Write Request to sub-index 128 returns without error even if there is no data stored.

Application hint: The latter allows to implement simple applications to request the CMT without knowing the actual project configuration. If the application wants to configure the network with more control it can use a loop over all known slaves.

6.5 EDS storage

For some devices it may be possible to store the EDS. This has some advantages:

- The manufacturer does not have the problem of distributing the EDS via disks
- Management of different EDS versions for different software versions is less error prone, if they are stored together
- The complete network settings may be stored in the network. This makes the task of analysing or reconfiguring a network easier for Tools and more transparent for the users.

For those devices which are not able to store their EDS, the Configuration Manager may take over this task. For this the following objects are defined in the Configuration Manager:

Index	Object	Name	Type	Attr.	M/O
1F23H	ARRAY	Store Slave EDS	domain	RW	O
1F24H	ARRAY	Slave EDS Storage Format	Unsigned8	RW	O

Index 1F23H, sub-index 0 describes the number of entries. This is equal to the maximum possible Node-ID (127). Each sub-index points to the Node-ID of the device, for which the EDS belongs.

Uploading the EDS of a device from a Configuration Tool to the Configuration Manager is done by writing the EDS file as a domain to the object 1F23H with the sub-index equal to the devices Node-ID.

Downloading the EDS of a device from the Configuration Manager to a Tool is done by reading the object 1F23H with the sub-index equal to the devices Node-ID. If no data had been stored, this is aborted with the error code 08000024H "Data set empty"

The filename does not need to be stored since every EDS contains its own filename.

Object 1F24H have the same description and behaviour as object Storage Format in the DCF storage (object 1F21H).

7 Dynamic Establishment of SDO Connections

CANopen offers a communication mechanism between devices via Service Data Objects. These communication channels are always established between two nodes. For accessing a device the first time at least one SDO per device is required. This is the default SDO. Only the SDO-Manager has the right to access that SDO.

Each CANopen device may support additional SDOs. By default they are disabled. Refer to DS-301 sections "SDO parameter" and "Detailed Specification of Communication Profile specific Objects" 1200H-12FFH.

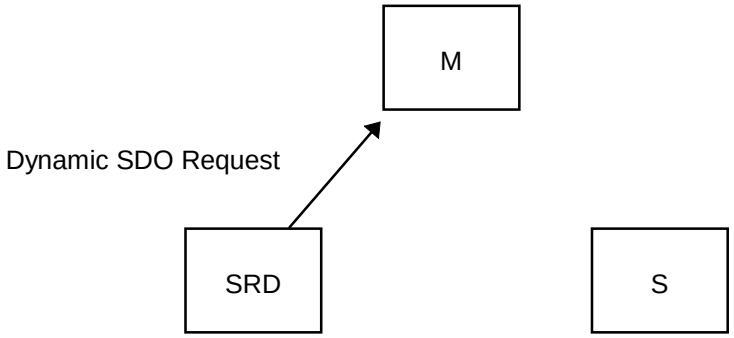
For most of the application and most of the device types the mechanism of using SDOs is as simple as with PDOs; any pair of devices may be pre-configured to have an SDO connection.

This chapter describes the method whereby nodes that are plugged in and out while the system is running can establish dynamic SDO connections. This concerns for example configuration tools, analysis tools or even HMIs with very intelligent configuration set-up. In general every CANopen device can use this mechanism to establish dynamic SDO connections to other devices. In the following every such device is referred to as "SDO Requesting Device" *SRD*. This is done to make a clear distinction of this dynamic mechanism from the more static method of pre-configured SDO connections.

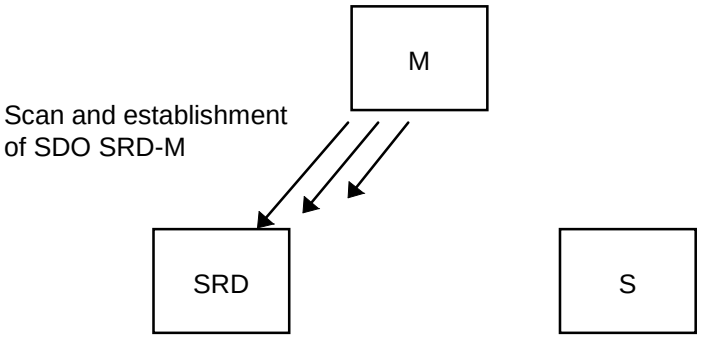
The SDO Manager is active only if the NMT Master residing on the same device is active.

7.1 Basic mechanism

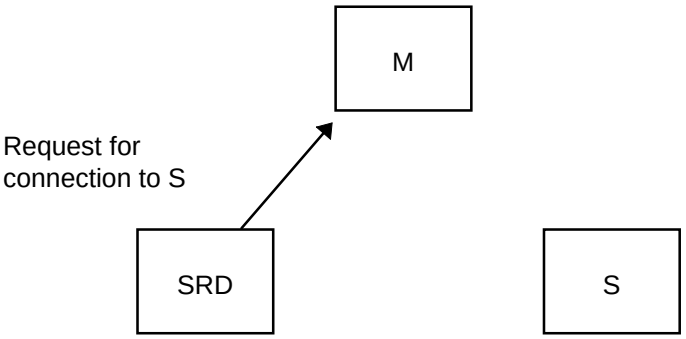
The SDO Manager M manages all SDO connections in the network. It can dynamically establish new connection between a SRD and a slave S. For this the device SRD first has to become registered. This is done with the service "Dynamic SDO Request".



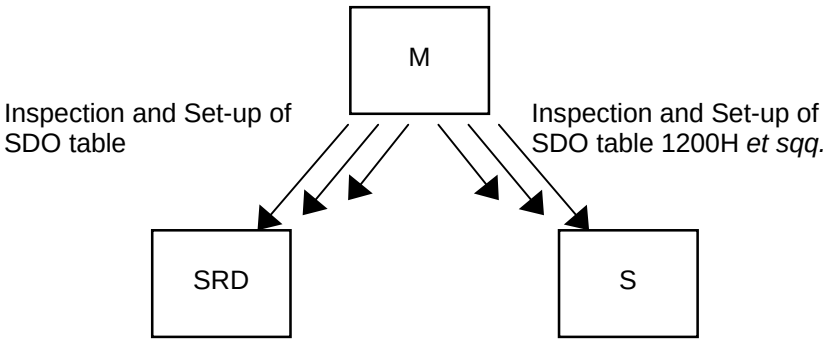
In the next step the SDO Manager scans for the requesting device and establishes a connection with SRD as client and M as server. For establishing the connection on the SRD, the SDO Manager uses the object index of the Client SDO object dictionary entry which the SRD transmits within the object entry Dynamic SDO Request State:



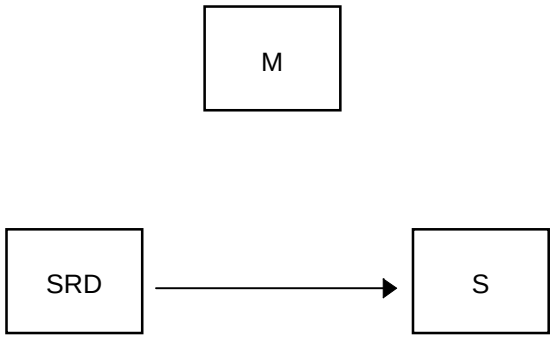
Hereafter the device SRD can perform requests to the Manager via the new SDO. It will use this to request a connection to device S (or some more devices). Together with the request, the SRD transmits the object index of an own free Client SDO object dictionary entry which the SRD wants to use for the SDO communication with device S.



The SDO Manager checks its internal table to determine whether the default SDO of S is free. If this is occupied it will check the object dictionary of S for free additional SDOs (objects 1200H *et sqq.*). It then establishes the connection by writing into the object dictionary entries of the free Client SDO table entry on the SRD and 1200H *et sqq.* of S. If it decides to use the default SDO of S, it does not need to write to the object dictionary of S, in that case it will update only its internal table.



The result is a SDO connection with SRD as client and S as server. With this SRD can access S as requested.



Established SDO connection

7.2 Specification

7.2.1 SRD registration

Registration process

The SDO Manager manages a table with all SRDs, that have access to SDOs. Those SRDs have the state *registered*.

To become registered, an SRD sends the "Dynamic SDO Request". This is a data frame without data bytes.

The COB-ID is 1760 dec.

After receiving such a request, the SDO Manager scans the network for SRDs. It stops the scan at the first unregistered SRD it finds.

Scanning is carried out using SDO access to the object dictionary of the SRD. As all nodes are attached to the Manager by default, the Manager uses the default SDO. The Manager knows all default SDOs which are already allocated to other nodes.

For the scanning the Manager reads the object 1F10H "Dynamic SDO Connection State" of all possible Node-IDs which are not yet registered as SRD. If this object states an open request, the Manager uses the value OD Index from the object 1F10H and enables this entry in the “Client SDO parameter” table. Afterwards the object 1F10H is reset by the Manager.

The described registration process results in a SDO connection from the SRD (Client) to the SDO-Manager (Server).

Error Control of SRD

Any registered SRD has to participate in Error Control. After entering the list of registered SRDs the SDO Manager will start the Node Guarding Protocol or Heartbeat Protocol. This implies that the SDO Manager is placed in the same device instance as the NMT Master.

On a Error Control Error Event caused by the SRD, the Manager forces all SDOs requested by that SRD to be released by writing into the corresponding SDO tables of the SRD and all by the SRD accessed devices.

De-registration

If a SRD wants to de-register it has to indicate that by writing to the SDO Manager's object dictionary (Object 1F01H, Slave-Node-ID = 0).

If all the SDO connections of an SRD are released, it may stop performing the Error Control Services.

7.2.2 Requesting/Releasing SDO Connections by SRD

Requesting Dynamic SDO Connections

In order to establish a new dynamic SDO connection to a device S, a registered SRD requests the new connection by writing on object index 1F00H at the SDO Manager. The request data consists of the node-ID of the SRD, the node-ID of the device S and an object dictionary index of a free Client SDO object entry on the SRD which the SRD wants to use for the communication with device S. After setting up the dynamic SDO connection between the SRD and device S the SDO Manager informs the SRD by writing on object index 1F10H on the SRD. The SDO Manager also informs the SRD by writing to object 1F10H if it was not possible to establish the connection. An error code informs the SRD about the reason for the failure.

Requesting Access to all default SDOs

A SRD can also request access to all default SDOs in a system. This is done during the registration process. The SRD indicates the request in object index 1F10H. If the SDO Manager grants the access to all default SDOs to the SRD, the SDO Manager confirms this by writing to object index 1F10H on the SRD. There is no explicit SDO connection established between SRD and SDO Manager as in the normal registration process.

Releasing Access to all default SDOs

A SRD can release an granted access to all default SDOs only by stopping performing the Error Control Services. As a consequence the SRD will also be de-registered.

Releasing Dynamic SDO Connections

In order to release a dynamic SDO connection to a device S the SRD transmits a release request by writing to object index 1F01H on the SDO Manager. The request data consists of the node-ID of the SRD, the node-ID of the device S and the object dictionary of the Client SDO of the SRD which the SRD wants to be released. If the SRD specifies the index number 0 then all established dynamic SDO connections to the specified device are requested to be released.

Error Control of Slave Devices

An optional feature of the SDO Manager is that in the registration process a SRD can request the SDO Manager to start also the Node Guarding Protocol or Heartbeat Protocol for the slave devices to which the SRD has established connections if there are no error control services active for the devices. The SRD requests this feature by setting a bit in the object Dynamic SDO Connection State (1F10h). If the SDO Manager supports this feature this bit remains set otherwise the SDO Manager resets this bit when confirming the establishment of the SDO connection from SRD to SDO Manager.

On a Error Control Error Event caused by the slave S, first the Manager reports the fail of slave S by writing to object 1F11H. Then the Client SDO entry which was assigned to the slave S is cleared on the SRD.

7.2.3 Object Dictionary Extensions

Mandatory Manager facilities

The following entries are used by SRDs to request and release SDOs:

Index	Object	Name	Type	Attr.	M/O
1F00H	VAR	Request SDO (c->s)	Unsigned32	WO	M
1F01H	VAR	Release SDO (c->s)	Unsigned32	WO	M

Any registered SRD may write to one of those objects to request or release an SDO.

At object 1F00H the written object value indicates the node to which the SRD wants to have access to the SRD's Node-ID and the index to a free Client SDO entry, where the channel should be established.

16-31	8-15	0-7
OD Index	SRD Node-ID	Slave Node-ID

If the Manager refuses the request this will be done by the Service "Abort domain transfer" with abort code 060A 0023h ("Resource not available: SDO connection"). The requested Node-ID must not be 0.

On object 1F01H the value is coded as 32 Bit value:

16-31	8-15	0-7
OD Index	SRD Node-ID	Slave Node-ID

Bits 0-7 indicate the Node-ID from which the SRD wants to be released. If the SRD wants to release the SDO connections to all nodes and also to de-register, the value must be set to 0. This field is mandatory.

Bits 8-15 indicate the Node-ID of the SRD.

Bits 16-31 indicate the index into the object dictionary of the SRD, where the SDO connection is stored. If this value is 0, all SDO connections to the slave are released. If the value points to a valid SDO connection, this connection is released.

The act of writing to the objects above only initiates the action on the Manager. A successful response means, that the Manager starts the Establishment/Release. It does not mean that the complete action has been successfully completed. The SRD has to observe the result at its own object dictionary entries (establishment in 1F10H respectively release in the range 1280H-12FFH). The time-outs used have to take into account that the Manager has to do several SDO accesses to the indicated node and the SRD.

SRD objects

Any Node which wants to establish SDO connections dynamically, has to support the following objects:

Index	Object	Name	Type	Attr.	M/O
1F10H	VAR	Dynamic SDO Connection State	Unsigned32	RW	M

This object is used for the SRD registration process as well as for the confirmation of an established SDO connection to a device S which the SRD has requested by the SDO Manager before.

When this object is read and the SRD did not send a “Dynamic SDO Request” or is already registered by the Manager, the value has to be 0.

If the SRD sends a "Dynamic SDO Request", it has to set the value of this object to xxxx0001H, whereas xxxx is the value of OD index (if the SRD requests active error control services for the devices S it has to set the value to xxxx 0009H). If the SRD wants to have access to all default SDOs, it sets the value 0000 0003H.

After the SRD is registered and a connection from the SRD to the Manger is established, the Manager confirms the registration by writing to this object. Also errors during the registration process are reported from the Manager to the SRD using this object.

When the SRD has requested a dynamic SDO connection to a device S, the Manager confirms the successful or non-successful establishment of the requested SDO connection to the SRD using this object.

16-31	15-8	7-4	3	2-1	0
OD Index	Error code	X	Req EC	Cnxn State	Rq Indication

Rq Indication:

When the SRD wants to be registered, this bit has to be set.

This bit has to be reset to 0 by the Manager. This indicates the successful recognition and registration of the SRD by the Manager.

Cnxn State:

If the Manager is not able to establish an SDO connection from SRD to Manager or from SRD to device S, it will set this field to 0. The reason is written into the field error code.

If the Manager supports the dynamic establishment of SDOs and has established an SDO connection from the SRD to the Manager by entering the new SDO in the SRDs SDO table, it will set this field to 1.

If the SRD requested the ownership of all default SDOs (1F10H = 0000 0003H) and all default SDOs in the system are unused (which will be true in many systems after completion of the boot-up), the Manager may set this field to 2. This assigns the ownership of all default SDOs to the SRD. In this case the Manager does not have to set-up any SDOs on the slaves and on the SRD. This has the advantage of establishing the SDOs relatively quickly.

If the Manager has established a requested SDO connection to device S, it sets this field to 3.

Req EC

This bit must be set if the SRD requests the SDO Manager to start also the Node Guarding Protocol or Heartbeat Protocol for the devices S to which the SRD has established connections if there are no error control services active for the devices S.

If the SDO Manager supports this optional feature it will set this bit if it was set before in the SRD registration request. If the SDO Manager does not support this feature it will reset this bit.

X Always 0. This field is reserved for further use by CiA.

Error code

If the field Cnxn State has been set to 0, the field Error code gives the reason for the error:

Error Code	Meaning
0	No precise details for the reason of the error
1	No SDO channel free for connection from SRD to SDO Manager
2	No more free SDO channels available in the network
3	No more free Server SDO entries on slave S
4	Slave S not available

The codes not listed here are reserved.

OD Index:

SRD reports a free Client SDO entry to the Manager, where the registration channel should be established on the SRD. When the Manager writes to this object, it contains the index of the Client SDO object entry, where the channel was established. This value is not used if access to all default SDOs is requested.

Index	Object	Name	Type	Attr.	M/O
1F11H	VAR	Slave Failed	Unsigned16	RO	O

8-15	0-7
Reason	Slave Node-ID

This object dictionary entry is used to inform the SRD that the slave S caused a Error Control Error Event. The dynamic SDO connection is not longer valid. The Manager starts automatically to release the dynamic SDO connection. This object is optional on the SRD and is only supported if the Manager supports bit 3 in object 1F10H (Dynamic SDO Connection State).

Reason	Meaning
0	Slave S failed to respond on Node Guard/Heartbeat Protocol

Configuration of SDO Manager

For establishing dynamic SDO connection the SDO Manager requires information about the COB-IDs that are free in the system. This can be configured in the following table:

Index	Object	Name	Type	Attr.	M/O
1F02H	ARRAY	SDOMgr COB-IDs	Unsigned32	RW	M

The fields have the following structure:

	Unsigned32 MSB				LSB
bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0/1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0/1	1	29-bit Identifier	

bit number	value	meaning
31 (MSB)	0 1	COB-ID valid COB-ID not valid
30	0 1	COB-ID free for usage COB-ID is actually in use
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)
28 - 11	0 X	if bit 29=0 if bit 29=1: bits 28-11 of 29-bit-COB-ID
10-0 (LSB)	X	bits 10-0 of COB-ID

The COB-ID valid/not-valid bit signs if this entry is configured. The SDO Manager will set the bit "COB-ID free for usage/is actually in use" to 1, if it allocates that COB-ID from the list and reset it to 0, if the corresponding SDO connection is released.

If a Configuration Tool writes to these entries, it has to ensure, that there are no open dynamic SDO connections.

All actually running SDO connections are stored in the following table:

Index	Object	Name	Type	Attr.	M/O
1F03H	ARRAY	SDO Connections Part 1	Unsigned32	RO	M
1F04H	ARRAY	SDO Connections Part 2	Unsigned32	RO	O
1F05H	ARRAY	SDO Connections Part 3	Unsigned32	RO	O
1F06H	ARRAY	SDO Connections Part 4	Unsigned32	RO	O

This table allows the description of up to 254 connections with every SDO Manager. Optionally it can be extended for up to 1016 connections.

The fields have the following structure:

	Unsigned32 MSB			LSB
bits	*24-31*	*16-23*	*8-15*	*0-7*
field	Client offset	Client ID	Server offset	Server ID

Server ID is the Node ID of the SDO connection server. Server offset is the offset into the servers SDO table that begins with 1200H. Client ID is the Node ID of the SDO connection client. Client offset is the offset into the clients SDO table that begins with 1280H. If one of the fields Server ID or Client ID is 0, the connection is not valid, the entry is actually free.

The SDO Connections table is ReadOnly. If a tool wants to force connections to be released, it may do this with object 1F01H "Release SDO".

Device Configuration File

Configuration tools have to know, which device in the network is the CANopen Manager. For the purpose of designating this manufacturer independent, this information is written into the DCF files of the project. The section `DeviceComissioning` of each device contains the boolean entry `CANopenManager`. If the entry is missing, the described device is not the CANopen Manager.

7.2.4 Implementation guideline

1. If the SDO Manager does not answer the "Dynamic SDO Request", the SRD should inform the user. In this case it's the user's responsibility to decide whether he wants to switch the Manager off and continue the action. Then the SRD may use the default SDO to access other nodes.

2. The SRDs should get the highest possible Node-ID in the system. This enables the Manager to start the scan at the highest address 127. Normally then it will find the SRD - without unnecessary accesses to slave nodes - in the shortest possible time.

3. The SDO Manager is allowed to serve the SRD's requests in the manner of a relay station. Amongst others this allows the advantage of connecting a device which has only the default SDO to several other devices.

4. If the SDO Manager assigned all default SDOs to the SRD (Object 1F10H field Cnxn state set to 2), it has to assume that the SDOs remain in the SRDs ownership until the SRD stops Guarding

5. For configuring the SDO Manager, the Configuration Tool has to request an SDO connection to it. At least at the very first configuration the COB-ID list of the the SDO Manager does not contain any valid entries. For establishing the SDO connection to the Configuration Tool, the SDO Manager uses the pre-defined connection set (default SDO).

7.2.5 Dynamic SDO establishment algorithm summary

1 The SDO Manager manages all SDOs in the system.
2 The SDO Manager and all SRDs have a Node-ID and an object dictionary.
3 If the SRD is not yet registered:
3.1 SRD requests SDO via "Dynamic SDO Request".
3.2 SDO Manager scans with SDO access.
3.3 Manager establishes SDO from SRD (client) to Manager (server), if not all default SDOs requested
3.4 SDO Manager confirms registration of SRD to SRD
3.5 SDO Manager enters the SRD in its guarding/heartbeat list.
4 If the Master did not already assign all default SDOs to the SRD (in 3.3):
4.1 SRD asks the SDO Manager for an SDO connection to device S.
4.2 SDO Manager enables SDO on the device S and attaches it to the SRD.
4.3 SDO Manager confirms SRD the established connection to device S
4.4 Optional: SDO Manager requests NMT Master to start error control services for device S (if not already started) and enters device S in its guarding/heartbeat list.
5 Releasing the SDO connection:
5.1 On Node-Guard/Heartbeat -Error the SDO connection is forced to be released.
5.2 SRD releases SDO if no longer needed.

8 Input/Output of a Programmable Device

8.1 Basics

In a network programmable nodes can be characterised as a process having input variables and output variables. The set of variables will be arguments of the program and hence will be only known in a final state when the program has been written. The arguments must be handled as variables located in the object dictionary.

The marking of such parameters depends on the programming system (e.g. IEC61131) and can not be standardised here. But it can be assumed that there is a set of network variables with the logic attribute EXTERN.

Compiling/Linking (or interpreting) a program including EXTERN variables requires relocation information. Within CANopen devices this information is the index (and sub-index) of the variable. Most of the programming systems know the mechanism of a resource definition. This can be used to assign the CANopen attributes (index, sub-index, R/W, Assignment of CANopen data type to local data type etc.) to the corresponding symbolic names (variable name in the program). The resource definition may be created with a simple editor by the user or with much more comfort by a configuration tool. On systems with a disk-based file system a direct exchange via the DCF format is possible.

The names of variables have to meet the rules of the underlying programming system. CANopen makes no restrictions for this. So this is the responsibility of the programmer/manufacturer.

Defining EXTERN variables requires a rule for distributing the indices. It is called "dynamic index assignment".

8.2 Dynamic Index Assignment

The index area used for dynamic index assignment is dependent on the device. Each data type and direction (Input/Output) has its own area, called segment. These segments must not overlap. Variables of same type are gathered in one array. If all elements of an array are defined (sub-index 1-254), the next free object of the area is allocated.

In order to allow programmable devices the use of a process picture, they may implement a conversion formula which calculates the offset of a variable in the process picture in direct dependence from the index and sub-index.

Definition of the abstract object segment:

A segment is a range of indexes in the object dictionary with the following attributes:

Data type	This is the data type of the objects which can be defined in this segment.
Direction	This flag distinguishes between inputs and outputs. The values are ‘wo’ for outputs and ‘ro’ for inputs. The distinction is important to know whether the variable can be mapped into a receive PDO (wo) or transmit PDO (ro). This does not concern the access possibilities via SDO.
Index range	Range of indices with start index and end index.
PPOffset	Offset in the process picture, where the first object of this segment is allocated. For byte and multi-byte variables this is a 32 bit unsigned offset value. For Boolean variables it is the offset and additionally the address difference between two Boolean variables counted in bits. If Boolean variables are packed in bytes one bit after the other, the value is 1, if Booleans are each stored in a byte cell, the value is 8 (see the EDS example below).
Maximum count	The maximum number of variables in this segment.

Many devices distinguish strictly between different segments in the process picture for different data types. For those devices the PPOffset of the first segment will be 0, the PPOffset of the second segment will be the maximum count of the first segment multiplied by the data type size of the first segment and so forth. If this does not exactly meet the physical configuration, the device software is free to implement this on a logical point of view by using internal segment descriptors/offsets.

Other devices mix different data types in the same segment. For those devices all PPOffset attributes will have the value 0. Configuration Tools which allocate space in that process picture by assigning indexes have to take into account, that in this case indexes have to be left out to avoid overlapping. (For special applications it may be a feature to explicitly overlap variables. This helps interpreting memory cells as different types in debuggers.)

Any mixed form of those two device types is possible.

8.3 EDS

The Electronic Data Sheet is a general description of a device type. More details are defined in /3/ DS-306 "Electronic Data Sheet Specification".

The capability of a device to manage dynamic variables is declared in the EDS with an entry in the section `DeviceInfo`:

```
DynamicChannelsSupported=1
```

If the entry does not exist, it is assumed to be 0. The type of this entry is Unsigned32. The lowest bit states the general facility of allocating dynamic variables. Bit 2 is set by /4/ Profile DS-405 for a predefined segment description. Bit 3 is set, if the objects are already described in the EDS. Any other bit may define a specialised strategy. Actually bits 4-31 are reserved by CiA.

If the entry `DynamicChannelsSupported` is non-zero, the section `DynamicChannels` describes the usable segments. The following keywords are used:

`NrOfSeg` Number of segments. It is a decimal number or hexadecimal with leading 0x.

The following entries are counted with an "entry index". This is a decimal unsigned32 value starting with 1.

`Type<entry index>=data type`

The data type is the data type of the objects which can be stored in this segment. It is coded as in the entry `DataType` of object descriptions.

`Dir<entry index>={ ro | wo | rww}`

This entry distinguishes between inputs (ro) and outputs (wo/rww).

`Range<entry index>=index range`

The index range is pair of numbers, the first giving the start index, the second giving the end index. The numbers are unsigned16 hexadecimal with leading 0x.

`PPOffset<entry index>=segment offset in process picture [, bit address difference]`

This is the offset of this segment inside the process picture. The number is unsigned32 decimal or hexadecimal with leading 0x. If the data type of this segment is Boolean (0x0001), the bit address difference has to be defined as a decimal number or hexadecimal number with leading 0x. Most often this will have the values 1 or 8.

`MaxCnt<entry index>=maximum count`

Maximum number of objects which can be allocated in this segment. The unsigned32 number is decimal or hexadecimal with leading 0x.

`MSpecific<entry index>=generic`

This entry is optional. The contents are manufacturer specific. This entry allows to extent the description for application specific purposes. The contents will be interpreted by specialised configuration tools only. They have the responsibility to detect, if the content is correct. For example this could be done by using a unique marker at the beginning of the content.

Example:

```
[DeviceInfo]
....
DynamicChannelsSupported=1

[DynamicChannels]
NrOfSeg=2
Type1=5
Dir1=ro
Range1=0x7001-0x7017
PPOffset1=0
MaxCnt1=4096
Type2=1
Dir2=ro
Range2=0x7018-0x7018
PPOffset2=4096,1
MaxCnt2=64
```

This example defines two segments. The first describes the storage of 4 kByte of byte input variables, the second additional 64 Boolean input variables.

The EDS describes the state of a device as it is delivered by the device manufacturer. In this state, the application of the device is not defined yet. For this reason, the application variables cannot be defined, so any dynamic variables must not be defined in the EDS. Exception see above.

8.4 DCF

The Device Configuration File is the description of a concretely configured and programmed device. More details are defined in /3/ DSP-306 "Electronic Data Sheet Specification".

Besides the information of the EDS, the DCF stores the dynamic allocated objects. With this information it is possible for a configuration tool to retrieve the objects of a programmed device.

In the other direction it is possible to define variables with a project planning tool and afterwards use that information by the linker/relocator/resource definition editor of a programming system. This makes it possible to implement programming systems with nearly no knowledge of CANopen.

9 Program Download

In this chapter, a common way for program downloading to a device via its object dictionary is specified. Here only the mechanism for performing the program download is specified but not the structure of program data and not the data structure. Alternative mechanisms can be used with the help of the OS Command and OS Prompt objects 1023H-1026H as described in DS-301.

The specified mechanism can be used for downloading complete programs to devices (e.g. if a device only provides a kind of CANopen bootstrap-loader) or only parts of a program (e.g. specific tasks of real-time systems). The data structure of the transferred program data has to be specified by the manufacturer (e.g. INTEL-HEX format or binary format).

Further specifications for the program download have to be made in specific device profiles (e.g. in /4/ DS-405 for the download of PLC programs).

For the download of the program data a new object is introduced:

Index	Object	Name	Type	Attr.	M/O
1F50H	ARRAY	Download Program Data	Domain	RW	O

The sub-objects for the Download Program Data Object are:

Index	Sub-Index	Field in Download Program Data	Data Type
1F50H	0H	Number of different programs supported on the node	Unsigned8
	1H	program number 1	Domain
	2H	program number 2	Domain
	:		
	FEH	program number 254	Domain

If the download fails, the Device responds with an Abort SDO Transfer (error code 0606 0000h).

A second object is specified for controlling the execution of stored programs:

Index	Object	Name	Type	Attr.	M/O
1F51H	ARRAY	Program Control	Unsigned8	RW	O

The sub-objects for the Program Control Object are:

Index	Sub-Index	Field in Program Control	Data Type
1F51H	0H	Number of different programs on the node	Unsigned8
	1H	program number 1	Unsigned8
	2H	program number 2	Unsigned8
	:		
	FEH	program number 254	

The range for the sub-objects of the Program Control Object is 0 to 2.

The values have the following meaning:

0 - stop program (W) / program stopped (R)
1 - start program (W) / program running (R)
2 - reset program (W) / program stopped (R)

If the action is not possible, the Device responds with an Abort SDO Transfer (error code 0800 0024h).

A third object is defined to support verification of the version of the stored program number 1 (application software)[1]:

Index	Object	Name	Type	Attr.	M/O
1F52H	ARRAY	Verify Application Software	Unsigned32	RW	O

The sub-object of the Verify Application Software -object are:

Index	Sub-Index	Field in Program Control	Data Type
1F52H	0H	Number of supported entries	Unsigned8
	1H	Application software date	Unsigned32
	2H	Application software time	Unsigned32

Application software date contains the number of days since January 1, 1984. Application software time contains the number of milliseconds after midnight (00:00).

Note that only the date and time of a single application program is supported. Dates and times of programs 2 to 254 are not supported. Hence, if the application software is a single entity, it should be the program number 1. In case of two or more programs, one possibility could be to store the latest update time and date of any of the programs 2 to 254 into the object 1F54.

CANopen manager owned objects

Two objects are specified for verification of the version of the application software at the slaves:

The sub-objects for the Program Control Object are:

Index	Object	Name	Type	Attr.	M/O
1F53H	ARRAY	ExpectedApplicationSWDate	Unsigned32	RW	O
1F54H	ARRAY	ExpectedApplicationSWTime	Unsigned32	RW	O

ExpectedApplicationSWDate contains the number of days since January 1, 1984. ExpectedApplicationSWTime contains the number of milliseconds after midnight (00:00).

Sub-Index 0 NrOfSupportedObjects has the RO value 127. Sub-Index i (with i=1..127, i != Node-ID of Configuration Manager): Stores the expected application software date/time of the slave that has Node-ID i.

[1] Note that the object 100Ah (Manufacturer Software Version) of DS-301 can be regarded as a version number of a fixed program of a non-programmable node or as a firmware (like boot block and operating system) version number of a programmable node. Hence, a separate object for re-programmable application software is defined.

10 Summary of object dictionary extensions

SDO Manager and SDO requesting related objects

Index	Object	Name	Type	Attr.	M/O
1F00H	VAR	Request SDO (c->s)	Unsigned32	WO	M
1F01H	VAR	Release SDO (c->s)	Unsigned32	WO	M
1F02H	ARRAY	SDOMgr COB-IDs	Unsigned32	RW	M
1F03H	ARRAY	SDO Connections Part 1	Unsigned32	RO	M
1F04H	ARRAY	SDO Connections Part 2	Unsigned32	RO	O
1F05H	ARRAY	SDO Connections Part 3	Unsigned32	RO	O
1F06H	ARRAY	SDO Connections Part 4	Unsigned32	RO	O
1F10H	VAR	Dynamic SDO connection state	Unsigned32	RW	M
1F11H	VAR	Slave Failed	Unsigned16	RO	O

Configuration Manager related objects

Index	Object	Name	Type	Attr.	M/O
1F20H	ARRAY	Store DCF	Domain	RW	O
1F21H	ARRAY	Storage format	Unsigned8	RW	O
1F22H	ARRAY	Concise DCF	Domain	RW	O
1F23H	ARRAY	Store Slave EDS	Domain	RW	O
1F24H	ARRAY	Slave EDS Storage Format	Unsigned8	RW	O
1F25H	ARRAY	ConfigureSlave	Unsigned32	RW	O
1F26H	ARRAY	ExpectedConfigurationDate	Unsigned32	RW	O
1F27H	Array	ExpectedConfigurationTime	Unsigned32	RW	O

Program Control related objects

Index	Object	Name	Type	Attr.	M/O
1F50H	ARRAY	Download Program Data	Domain	RW	O
1F51H	ARRAY	Program Control	Unsigned8	RW	O
1F52H	ARRAY	Verify Application Software	Unsigned32	RW	O
1F53H	ARRAY	ExpectedApplicationSWDate	Unsigned32	RW	O
1F54H	ARRAY	ExpectedApplicationSWTime	Unsigned32	RW	O

NMT Master related objects

Index	Object	Name	Type	Attr.	M/O
1F80H	VAR	NMTStartup	Unsigned32	RW	O
1F81H	ARRAY	SlaveAssignment	Unsigned32	RW	O
1F82H	ARRAY	RequestNMT	Unsigned8	RW	O
1F83H	ARRAY	RequestGuarding	Unsigned8	RW	O
1F84H	ARRAY	DeviceTypeIdentification	Unsigned32	RW	O
1F85H	ARRAY	VendorIdentification	Unsigned32	RW	O
1F86H	ARRAY	ProductCode	Unsigned32	RW	O
1F87H	ARRAY	RevisionNumber	Unsigned32	RW	O
1F88H	ARRAY	SerialNumber	Unsigned32	RW	O
1F89H	VAR	BootTime	Unsigned32	RW	O
1F90H	ARRAY	Flying Master Timing Par.	Unsigned16	RW	O
1F91H	ARRAY	Startup-Capable Device Timing	Unsigned16	RW	O

Defined SDO Abort Codes

Code	Meaning
060A0023H	Resource not available: SDO connection
08000024H	Requested action cannot be performed by the application
08000024H	Data set empty

Identifier definitions

ID range	defined by
71h-76h	Flying Master
6e0h	SDO Manager