

yaca

yet another CAN automation

technical manual

Table of Contents

1 Protocol definition.....	3
1.1 General thoughts.....	3
1.2 Master communication.....	3
1.2.1 Temp-ID association.....	3
1.2.2 Network-ID.....	4
1.2.3 New nodes.....	4
1.2.4 Bus power-off message.....	5
1.2.5 Error messages.....	5
1.2.6 Reading/writing the CANid data block.....	5
1.2.7 Changing the Master-ID.....	6
1.3 Bootloader.....	6
1.3.1 Entering the bootloader.....	7
1.3.2 Hard-entering the bootloader.....	7
1.3.3 Entering bootloader update.....	7
1.3.4 Programming.....	7
2 Node datasheet.....	9
3 Message index.....	11

1 Protocol definition

1.1 General thoughts

The message-based structure of the CAN bus is used – messages are not addressed to a node, but have a number (the CAN message ID, in the following: „*CANid*“) identifying their usage. yaca uses extended *CANids*. These are assigned dynamically by the *master node*. The *master node* is used for setup, configuration and monitoring/logging, but the system must continue working even if the master fails.

In code, messages are simply implemented as C functions with a special prefix. A pre-/postprocessor then creates header files for including in code of other nodes, a function pointer table and *CANid* table in EEPROM for calling the functions when a message with a matching *CANid* is received. On the other hand, it also creates glue code in assembler to send a message when a remote message is „called“. This implements a simple RMI (remote method invocation) system which is supposed to make the code simple and logical.

As avr-gcc is little-endian and the message parameters are not significantly changed (just packed / avr-gcc rounds param addresses to 2, but the 8 bytes param of CAN messages are just too expensive for that) before transmission, the transmitted params are also little-endian. Any other **numbers** wider than 1 byte are **represented as little-endian** too.

Despite being message-based, yaca still uses node identifiers (6-byte *Boot-IDs*) in configuration messages. The *Boot-ID* is hardcoded into the bootloader, effectively becoming a hardware address unique for each node.

1.2 Master communication

Configuration mode is entered using messages with a special *CANid*, the *Master-ID*. The target is identified by the *Boot-ID*. Configuration mode simply refers to the *Temp-ID* being set and enabled, see below. After being in configuration mode, the node can be referred to with the *Temp-ID*, making it possible to transfer more bytes per CAN message.

A possibility exists to change the *Master-ID* (e.g. for using yaca on a CAN bus that has foreign nodes operating too), but it should not be done if not necessary (if the *Master-ID* is lost, one can't enter the bootloader anymore). See ■ [Changing the Master-ID](#) for details.

1.2.1 Temp-ID association

This message is used to enter configuration mode – this is done by sending a temporary *CANid* (*Temp-ID*) to the target node which is then used for configuration messages later on. It sets a single byte of the *Temp-ID* at a time. If *byten* is 3 (4th byte), it also enables the *Temp-ID*. The frame is only valid if *byten* is between 0 and 3. The *Temp-ID* may only be stored in volatile memory (it must not be held after a power cycle).

CANid	0	1	2	3	4	5	6	7
<i>Master-ID</i>	<i>byten</i>	<i>data</i>	<i>Boot-ID</i>					

[0; 3]

Message 1: Temp-ID association

If the *master node* is done with the configuration, it revokes the temporary *CANid*.

CANid (RTR) 0 1 2 3 4 5 6 7

<i>Temp-ID</i>	0x01	<i>d.c.</i>					
----------------	------	-------------	--	--	--	--	--

Message 2: Temp-ID revocation

The node then answers with the same message, without the RTR bit set. The master may then free up the temporary *CANid* internally. If the node doesn't reply, the master shouldn't free up the *Temp-ID*. Cleaning up is performed on ■ Bus power-off.

1.2.2 Network-ID

Each node transmits a *Network-ID* request on bootup, except when it receives such a request from another node (in which case it must cancel the transmission to stop flooding the bus).

CANid 0 1 2 3 4 5 6 7

<i>Master-ID</i>	0xFF	0x00	<i>d.c.</i>				
------------------	------	------	-------------	--	--	--	--

Message 3: Network-ID request

On receipt of a *Network-ID* request, the *master node* broadcasts it, except when it wants to ■ Hard-enter a bootloader.

CANid 0 1 2 3 4 5 6 7

<i>Master-ID</i>	0xFF	0x01	<i>Network-ID</i>				
------------------	------	------	-------------------	--	--	--	--

Message 4: Network-ID broadcast

1.2.3 New nodes

If a node receives a *Network-ID* different from the ID stored in EEPROM, it has changed bus (it is now connected to a new bus). The node doesn't react on any stored *CANids* but informs the *master node*:

CANid 0 1 2 3 4 5 6 7

<i>Master-ID</i>	0xFF	0x02	<i>Boot-ID</i>				
------------------	------	------	----------------	--	--	--	--

Message 5: New node

The node also tells the *master node* its device type:

CANid 0 1 2 3 4 5 6 7

<i>Master-ID</i>	0xFF	0x03	<i>device type</i>				
------------------	------	------	--------------------	--	--	--	--

Message 6: New node device type

When the user wishes to use the node on the new bus, the *master node* clears the old *CANids* and overwrites the *Network-ID* in the new node:

CANid 0 1 2 3 4 5 6 7

<i>Master-ID</i>	0xFF	0x04	<i>Boot-ID</i>				
------------------	------	------	----------------	--	--	--	--

Message 7: Format node

1.2.4 Bus power-off message

Some nodes may hold status information in volatile RAM (e.g. to conserve EEPROM write cycles). When powering off the bus on purpose, this message advises all nodes to save volatile information in non-volatile memory. After powering off the bus, the master may clear all *Temp-IDs* in memory.

CANid	0	1	2	3	4	5	6	7
Master-ID	0xFF	0xFF	d.c.					

Message 8: Bus power-off broadcast

1.2.5 Error messages

If the bootloader detects a wrong CRC after startup, it transmits a CRC error message:

CANid	0	1	2	3	4	5	6	7
Master-ID	0xFE	0x01	Boot-ID					

Message 9: application CRC error

When the EEPROM overflows while writing the *CANid* data block, the node being programmed transmits a *CANid* data block address overflow:

CANid	0	1	2	3	4	5	6	7
Master-ID	0xFE	0x02	Boot-ID					

Message 10: CANid data block address overflow

1.2.6 Reading/writing the CANid data block

CANid data block write message:

CANid	0	1	2	3	4	5	6	7
Temp-ID	0x02	offset	count	data				

Message 11: CANid data block write

CANids are stored as zero-terminated strings. The flash memory contains the necessary function pointers. When a message is received, a loop goes through the *CANid* list byte-wise and matches the received *CANid*. On a zero, the index of the function pointer array is incremented. An entry may contain several *CANids*, i.e. an additional *Group-ID*.

CANid data block example (2 message functions): 01 23 45 67 12 34 56 78 00 23 45 67 89 00

The first message function is called when either *CANid* 01 23 45 67 or 12 34 56 78 is received.

It is important that the *master node* waits for a ■ CANid data block write done message after writing the *CANid* data block.

Single *CANid* read command – *mi* is the message index, *ci* the *CANid* sub-index (when *ci* is 0xFF, the command reads the count of *CANids* associated with a message):

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x03	<i>mi</i>	<i>ci</i>	<i>d.c.</i>				

Message 12: Single CANid read

The node sends a *CANid* reply:

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x04	<i>CANid/count (byte 1 only)</i>						<i>d.c.</i>

Message 13: CANid reply

It is also possible to make a *CANid* data block read:

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x05	<i>offset</i>	<i>d.c.</i>					

Message 14: CANid data block read

The node sends a *CANid* data block reply (usually returns 7 bytes, except if the read offset was larger than EEPROM-size - *CANid* data block size - 7):

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x06	<i>data</i>						

Message 15: CANid data block reply


The *CANid* data block is cached in RAM to be able to execute fast consecutive writes. While the cache content differs from the data stored in EEPROM, a dirty flag is set in EEPROM. When the cache is completely flushed to EEPROM, the node clears the dirty flag and transmits a *CANid* data block write done message.

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x07	<i>d.c.</i>						

Message 16: CANid data block write done

1.2.7 Changing the Master-ID

WARNING: If you lose the new *Master-ID*, you can't enter the bootloader anymore.

Out of security considerations, this message only works after  Hard-entering the bootloader.

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x08	<i>new Master-ID</i>					<i>d.c.</i>	

Message 17: Changing the Master-ID

1.3 Bootloader

The bootloader is used to update the firmware of a node without having to open the wall, box etc. where the node is located. It is the first program started on the microcontroller, for being able to download a new firmware even when the current firmware is bad. For this purpose, the

bootloader waits a few seconds for a ■ Temp-ID association after startup (bootloader time-out). After having received that, the bootloader will not start the application except when instructed to do so (by a ■ Temp-ID revocation).

Before starting the firmware, the bootloader also checks its CRC-checksum to ensure firmware integrity (e.g. when the firmware loading was interrupted).

1.3.1 Entering the bootloader

This shall work in both application and bootloader when in configuration mode (■ Temp-ID association sent). When executing the application, it jumps over to the bootloader section and tells the bootloader the current *Temp-ID*.

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x09	d.c.						

Message 18: Enter bootloader

1.3.2 Hard-entering the bootloader

If the current application firmware is bad (wrong CRC or application error which prevents entering the bootloader), the master needs to hard-enter the bootloader. This is done by powering off the bus (see ■ Bus power-off), waiting a few seconds until the buffer caps are discharged, re-applying power to the bus, waiting for the nodes to safely power up (e.g. 1 second less than bootloader time-out) and sending a ■ Temp-ID association to enter configuration mode. As the bootloader section runs first on powerup, the bootloader can be entered even if the application is bad.

1.3.3 Entering bootloader update

WARNING: If the newly programmed bootloader is not able to download a new bootloader, you have effectively locked yourself out. **Check this before updating the bootloader.**

Out of security considerations, this message only works after ■ Hard-entering the bootloader.

CANid	0	1	2	3	4	5	6	7
<i>Temp-ID</i>	0x0A	d.c.						

Message 19: Enter bootloader update

1.3.4 Programming

The application firmware memory is divided into pages. For the size of the pages or of the application flash, refer to the ■ NDS. The memory is updated page-wise; first, select a page, then send data until the page is full. The page will automatically be programmed once all bytes are received. **While programming a page, further data is ignored.** After programming, the page counter will be incremented automatically; selecting further pages is only needed when leaving out pages while programming partially. When page programming is done, the node sends a [programming done] message.

The following message selects a page in the flash of the microcontroller:

CANid	0	1	2	3	4	5	6	7
-------	---	---	---	---	---	---	---	---

Temp-ID	0x0B	page	d.c.
---------	------	------	------

Message 20: Bootloader page select

After a page is selected (or the page counter was incremented automatically), the *master node* sends the page data (the data pointer is advanced by the CAN message size):

CANid	0	1	2	3	4	5	6	7
-------	---	---	---	---	---	---	---	---

Temp-ID	0x0C	data
---------	------	------

Message 21: Bootloader page data

When the node has finished programming the page, it sends the following message:

CANid	0	1	2	3	4	5	6	7
-------	---	---	---	---	---	---	---	---

Temp-ID	0x0D	d.c.
---------	------	------

Message 22: Bootloader page done

2 Node datasheet

The *node datasheet (NDS)* is an XML file containing information about a specific node firmware, e.g. message functions, EEPROM size, page size etc. These are needed by the *master node*.

3 Message index

Message 1: Temp-ID association.....	3
Message 2: Temp-ID revocation.....	4
Message 3: Network-ID request.....	4
Message 4: Network-ID broadcast.....	4
Message 5: New node.....	4
Message 6: New node device type.....	4
Message 7: Format node.....	4
Message 8: Bus power-off broadcast.....	5
Message 9: application CRC error.....	5
Message 10: CANid data block address overflow.....	5
Message 11: CANid data block write.....	5
Message 12: Single CANid read.....	6
Message 13: CANid reply.....	6
Message 14: CANid data block read.....	6
Message 15: CANid data block reply.....	6
Message 16: CANid data block write done.....	6
Message 17: Changing the Master-ID.....	6
Message 18: Enter bootloader.....	7
Message 19: Enter bootloader update.....	7
Message 20: Bootloader page select.....	8
Message 21: Bootloader page data.....	8
Message 22: Bootloader page done.....	8