

**Draft
Specification
CAN Framing Protocol**

Alexander Bernauer

May 3, 2004

1 Abstract

CAN Messages can carry up to 8 bytes of payload. This suffices for most real time applications when sending sensor values or control commands for instance. But if you need to transfer more than 8 bytes over the network the sender has to split the information in proper portions. And the receiver has to know how to handle the fragments in order to read the information.

This document describes a fragmentation protocol doing this work. It is located in OSI layer 4 on top of the CAN layers. It is designed for use within the real-time publisher/subscriber model on the Controller Area Network introduced in [ISORC].

2 Overview

Basically there are two different types of transfer: copying and streaming. Imagine a node with a digital datasheet describing its specification. Another node is interested in this datasheet, i.e. it wants a copy of the document. Thus the top requirements are data integrity and completeness. If this requirements are met one can try to increase speed. Furthermore this scenario describes a point-to-point communication.

Now imagine a robot using a web cam to see its environment. A microcontroller running AI software reads the cam's images to come to its decisions. To give visual feedback to the developer another microcontroller reads the same data and displays the pictures on a monitor. In contrast to the above case we now have a point-to-multipoint communication. Another difference is that data can become old. Furthermore it is not fatal if a reader misses some parts, i.e. data completeness is not required. But still we want to guarantee data integrity for the received parts. Speed only matters when considering resolution of time, such as 30 frames per second for video data for instance. This transfer takes place in the so called streaming mode.

In either case the data is split up. Multiple CAN messages with up to 8 byte of data each form a so called frame. The frame size denotes the amount of CAN messages forming that frame. A 15 bit CRC checksum protects each frame against any kind of errors. As the underlying CAN Bus does error detection and recovery by its own frame errors are unlikely to happen. But as CAN has a design flaw which can lead to multiple reception of a message there must be error detection and correction in the fragmentation layer as well.

When a node wants to read any information it subscribes for the particular channel. This is one of the basic ideas of the publisher/subscriber model and is left untouched when adding the fragmentation layer. For each file transfer a separate channel is defined when designed the system. The sender of this file, e.g. the server, announces to its specific channel and the receiver or client subscribes to it. In the case of streaming an arbitrary number of clients can coexist. When copying a file this is different. As there must be some kind of handshaking and flow control between server and client this has to be a point-to-point communication. But you can time multiplex different server client connections. To support this two copy modes are introduced: when there is one server and multiple clients we call the channel being in upload mode. When there is one client and multiple servers we say the channel runs in download

mode. The fragmentation and the reassembly of the files are almost the same in this two modes. But there are differences concerning the handshaking. As copy transfers take place at demand there must be some way to initialize them. Therefore there exists one system wide request channel, where all potential servers are subscribed to and any node can announce to.

3 Streaming

In streaming mode there is no specific start or end of transmission. As soon as and as long as the channel exists there can be streaming data on it as the publisher sends new data whenever there is some. In this mode the server determines the frame size. Whenever it decides to terminate the frame it sends a message without payload and DLC being zero which is the end of frame signal. The next message carries the check sum over the frame's payload. Immediately after this it can send the next frame.

When a subscriber starts to read from a streaming channel it waits for the end of frame signal to get synchronized. He discards the next message which carries the check sum over the frame it did not completely receive. After that it reads the frame until its termination and computes the checksum. If it differs from the publishers checksum it rejects the whole frame and starts waiting for the next. Otherwise the frame is accepted and passed to the application. Besides the frame size the server also determines the transmission rate for the single CAN messages. Both values should be selected proper to the type of data. If either of both values is to high for a particular node this node is not able to read the streamed data.

3.1 Messages

This is the legend for the specification of the messages:

dat etag of the data channel
 prod producer's short ID

- Data

priority	node ID	etag	data
0xFF	prod	dat	payload

- End Of Frame

priority	node ID	etag	data
0xFF	prod	dat	-

- Checksum

priority	node ID	etag	data
0xFF	prod	dat	crc

4 Copying

In this mode every first byte of the first CAN message of an frame is referred to as control byte and contains protocol control informations. The frame size is

determined by the subscriber when requesting the transfer from the server. He also determines the transfer rate. This is to ensure that the subscriber is capable to process the published data. But the determined frame size and transfer rate are only upper bounds. If it is the publisher who reaches its capabilities it can reduce the frame size and transfer rate dynamically at any time proper to its system load.

When the publisher transmits a frame it meets the maximum transfer rate by delaying consecutive messages. Whenever the publisher wants to finish a frame or the maximum frame size is reached it indicates the end of frame by a single message without payload, i.e. with DLC equals to zero. Now the publisher waits for the subscriber to acknowledge.

The subscriber reads the messages and assembles them together until the end of frame message is received. He computes a check sum over the frame's payload and sends it to the publisher using the request channel. When the acknowledgment times out the transmission is canceled by the publisher. If not the publisher computes the checksum of the frame's payload and compares it with the subscriber's checksum. If they differ, the last frame is retransmitted while indicating this by a retransmission flag in the control byte. If they don't the publisher continues with the next frame with a cleared retransmission flag.

The consumer may append new values for the frame size and the transfer rate to the acknowledge message. This is useful if for some reason its capabilities change.

When the last frame was sent and positively acknowledged the publisher sends one last message carrying only the control byte and having the end of transmission flag set. When the subscriber receives this message it knows the transfer was completed and passes the data to the application.

4.1 State machines

The protocol sequence can be illustrated by specifying a finite state machine for both publisher and subscriber. This state machines assume that handshaking has been done already.

Publisher's state machine				
state	descr.	event	action	next
0	start	true	load first frame	5
1	sending	frame complete frame incomplete	send end of frame message send next message	3 2
2	delay	timer	set timer	1
3	waiting	receive acknowledge time out	compute checksum cancel transfer	4 -
4	verify	positive checksum negative checksum	load next frame set retransmission flag	5 1
5	load	no more frames frames left	send end of transmission message set timer	- 1

Subscriber's state machine				
state	descr.	event	action	next
0	start	true	open first frame	1
1	receive	recv end of frame	send acknowledge	2
		recv message	assemble	1
2	ack	recv end of transmission	pass to application	-
		recv retransmission	reset frame	1
		else	save frame, open next	1

4.2 Upload

In upload mode one producer serves multiple consumers. One node may not be the consumer of two upload transfers at the same time. This is supported by download mode. The consumer initiates the transfer by sending a request message to the producer. The producer responds by sending the first frame and everything goes on as it is described in section 4.1. Normally a CAN message carries the sender's short node ID as a part of its header. This would be the producers short node id. But the a client must be able to distinguish if a messages belongs to the frame it is actually receiving. Therefore the server puts the client's short node id into the header. This enables time multiplexed asynchronous transfers for multiple clients.

4.2.1 Messages

This is the legend for the specification of the messages:

req etag of the request channel
 dat etag of the data channel
 cons consumer's short ID
 prod producer's short ID

- Request

priority	node ID	etag	data				
0xFF	cons	req	0x01	dat	size	rate	select

type this byte indicates the message type
 dat addressing the server
 size the maximum frame size
 rate the maximum transfer rate
 select a selector for the case that the producer hosts multiple data sources

- Data

priority	node ID	etag	data				
0xFF	cons	dat	framing data				

data data according to the fragmentation protocol

- Acknowledge

priority	node ID	etag	data				
0xFF	cons	req	0x02	dat	crc	[size]	[rate]

type indicating the message type
 dat addressing the server
 crc CRC check sum over the last frame's payload
 size optionally new frame size
 rate optimally new frame rate

node ID notice that the semantic of this field is different for this message.
 It is not the sender's but the receiver's node ID.
 data data according to the fragmentation protocol.

- End Of Frame

priority	node ID	etag	data
0xFF	prod	dat	-

4.3 Download

In download mode there is one consumer which receives data from multiple producers. A node can not be the producer for two consumers at the same time. This is supported by upload mode. A consumer who requests for transfer does not know if the producer already serves someone else. Therefore the server gives a response to the request indicating whether it is busy or not. If it is not the data follows immediately corresponding section 4.1. If it is busy the client postpones the request.

Before the client can send a request it has to know which server is online and which short node id this server has. Therefore it can send a discover message through the request channel. This message carries its own short node id and the event tag of the channel it is interested in. Each server who can publish on this channel responses with its unique node id. This helps the client to distinguish the servers and select the right one. As the answer also carries the short node id of the sender, the client can immediately post its request to him. The responses need a separate channel as they can not be distinguished from data messages.

4.3.1 Messages

This is the legend for the specification of the messages:

req etag of the request channel
 dat etag of the data channel
 res etag of the discovery response channel
 cons consumer's short ID
 prod producer's short ID

- Discover

priority	node ID	etag	data
0xFF	cons	req	dat

dat addressing the servers

- Discovery Response

priority	node ID	etag	data	
0xFF	prod	res	unique node id	

- Request

priority	node ID	etag	data					
0xFF	cons	req	0x01	dat	size	rate	select	prod

type indicating the message type
 dat channel information for the producer
 size the maximum frame size
 rate the maximum transfer rate
 select a selector for the case that the producer hosts multiple data sources
 prod addressing the producer

- Acknowledge

priority	node ID	etag	data					
0xFF	cons	req	0x02	dat	crc	prod	[size]	[rate]

type indicating the message type
 dat addressing the server
 crc CRC check sum over the last frame's payload
 prod addressing the producer
 size optionally the frame size
 rate optionally the frame rate

- Response

priority	node ID	etag	data	
0xFF	prod	dat	cons	resp

cons addressing the consumer
 resp response is OK or DENY.

- Data

priority	node ID	etag	data	
0xFF	prod	dat	framing data	

data data according to the fragmentation protocol.

- End Of Frame

priority	node ID	etag	data	
0xFF	prod	dat	-	

References

- [ISORC] Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)
J. Kaiser - University of Ulm
M. Mock - GMD - German National Research Center for Information Technology