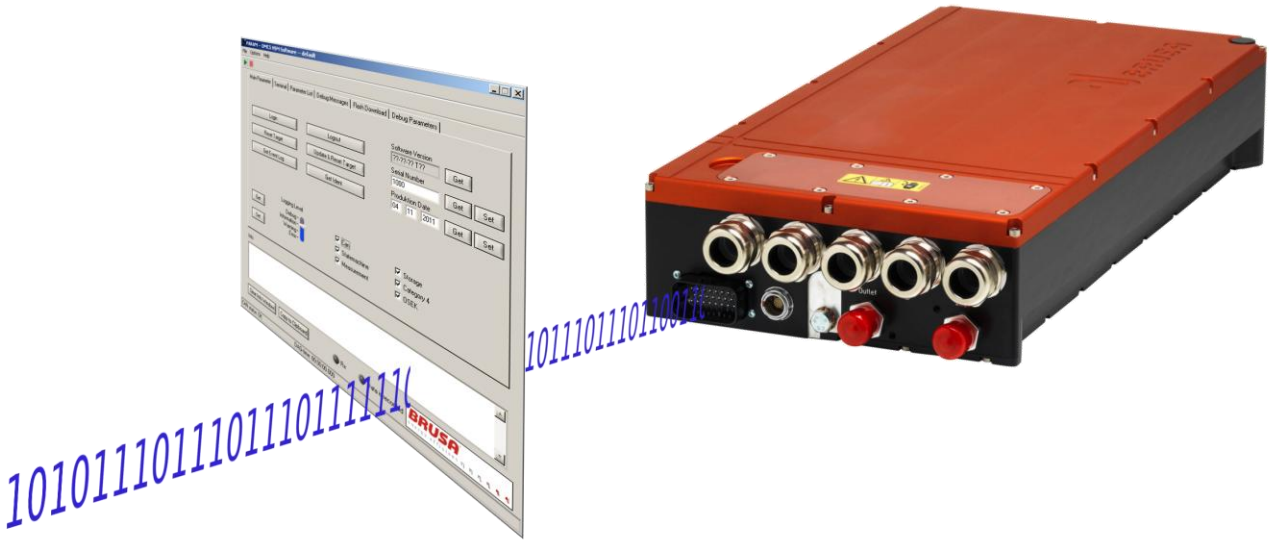
PARAM
How to use the protocol

Version 0.2



BRUSA Elektronik AG
Neudorf 14
CH-9466 Sennwald
+4181 758 19 00
info@brusa.biz
www.brusa.biz

1 Document History

Version	Date	Name	Comment
0.1	14.03.2011	Peter Oehry	New document
0.2	17.03.2011	Peter Oehry	Changes due to document review.

2 Introduction

This document gives an overview over the PARAM protocol. It will explain how to login and get or set parameters.

3 Table of contents

4 Principle operation of PARAM protocol

The PARAM protocol is a simple master slave communication to get and set parameters and other data in a control unit. One CAN ID is used to send messages from the master to the ECU and another ID for the respond message from the ECU to the master. The most communication is initiated by master side. The only messages that are sent sporadic by the control unit are a HELLO or a TRAP message. A HELLO Message is generated on start-up. A TRAP message occurs on a system crash and will normally never occur.
The messages are processed at a regular interval (4ms for the DMC for example) which means that this limits the maximum possible network load during communication. Every request of the master will lead into one or more respond messages.

4.1 Parameter

Every parameter has a 2-byte parameter number and consists of 4 byte of data associated to the number. Depending on the actual parameter the data have to be interpreted in different ways. They can be an integer or even a float number. The 4 bytes are in Motorola byte order (msb first). Every parameter is associated with access control rules that control if a parameter can be read or written. The access is controlled by the login. The actual parameter set is defined in the xml file that is released together with every software version.

5 Operating the PARAM protocol

All messages from ECU → Master have the same ID (TxId). The same is true for the respond message (RxId). The first byte of a CAN frame (byte 0) identifies the command code of the message.

5.1 Status Respond message

If no other special respond is defined for a request a status message will be sent as an answer. This status message can mean that everything was done or inform the master that there was an error.

StatRespond (0xF0)		**ECU → Master**					
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
1111 0000		ErrorCode		Data			

Name	**ErrorCode**	**Bedeutung**
DBG_ERR_ACK	0x00	Acknowledge to a set param.
DBG_ERR_INV_CMD	0x01	The command (byte 0) was not known to be a valid command. The byte 5 shows the command that was received.
DBG_ERR_ABORT	0x02	Status to inform about an aborted on-going sequence. This can be a Table Read for example.
DBG_ERR_OUT_OF_SYNC	0x03	Not used yet.
DBG_ERR_ACCESS	0x04	No access to the requested operation.
DBG_ERR_ACCESS_TYPE	0x05	The type of access is not valid. This occurs when the access type of a table read request is not known.
DBG_ERR_INV_ADDRESS	0x06	This occurs when the address of a table read request does not match the access type.
DBG_ERR_RANGE	0x10	This respond is generated when the parameter in a param set operation is out of valid range. The parameter was not written.
DBG_ERR_DATA_CHANGED	0x11	This respond is generated when the parameter in a param set operation is out of valid range but a changed value was written to the storage. The Data section (byte 4-7) holds the actual written data.

5.2 Set Param

A set param is done by sending a SetParam frame to the ECU. The respond will be a StatSespond message.

SetParam (0x03)		**Master → ECU**					
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
0000 0011		Param Nr		Param Data			

SetParam → StatRespond(ErrorCode) (Error or Acknolegke)

5.3 Get Param

A GetParam frame is sent to request a parameter. The respond is either a GetPrameRespond which includes the data or a StatRespond with the appropriate error code.

GetParam (0x04)		**Master → ECU**					
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
0000 0100		Param Nr					

GetParam → GetParamRespond
GetParam → StatRespond(ErrorCode)

GetParamRespond (0x84)		**ECU → Master**					
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
1000 0100		Param Nr		Param Data			

5.4 HELLO

The Hello message is used to show the start-up of an ECU. The frame also includes the information of the configured PARAM rx and tx id's

Hello (0x89)			**ECU → Master**				
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
1000 1001	0xA5	0xA5	0xA5	TxId (right justified)		RxId (right justified)	

5.5 Getting table data

The Param protocol gives a possibility to get a bigger amount of data out of the ECU. This is used to get the event log data structure for example. A TableReadRq consist of the type, the length (number of bytes) and the physical address of the data to be read. As a respond to a TableReadRq as many TableReadData frames are generated as need to transfer the requested data. The process can be aborted by sending another request to the ECU

TableReadRq (0x01)	**Master → ECU**						
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
0000 0001	Type	Length		Address			

Type: access size of the data:

0 → default (byte)
1 → byte
2 → short
4 → long
All others are invalid

TableReadRq → TableReadData x N/5
TableReadRq → StatRespond(ErrorCode)

TableReadData (0x81)	**ECU → Master**						
BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
1000 0001	N (Position)		Data(N)	Data(N+1)	Data(N+2)	Data(N+3)	Data(N+4)

5.6 Terminal emulation

The PARAM also supports a terminal emulation over CAN. This is intended to be a HMI (human machine interface) and not to be used by an automated process. For that reason the terminal emulation is not described further.

5.7 Flash download

The PARAM also supports a protocol to send flash data. This is used to program the motor table in a DMC5. For that the PARAM tool shall be used. The sub-protocol is not described in this document.

6 Use cases

6.1 Log in / log out

To log in as “monitor” the following two parameters have to be set:

Parameter Nr	Value
0x0000	0xE1159985
0x0001	0x1EEA667A

To log out the parameters have to be set to zero. These two parameters are read only parameters. The value cannot be read back.

6.2 Update / Reset

A special parameter is reserved to initiate an update of the CRC checksums in the ECU. The same parameter can be used to perform a reset.

NOTE: The request to update a CRC table will make the CRC valid but does not check if the parameters are making sense in the context.

NOTE: Log in is required to set the parameter.

NOTE: When requesting a reset it is not sure that the respond to the SetParam will be sent before the reset occurs. Thus the answer could be either a StatRespond and a HELLO or only a HELLO.

Parameter Nr	Value	Comment
0x40FF	0xC1A08000	Reset ECU
0x40FF	0xC1A000FF	Update CRC 0 – CRC 7 The 0xFF is a bitmap that will be associated to separate CRC tables. Sending 0xFF will update all CRC. Sending another mask will select single CRC sections.
0x40FF	0xC1A080FF	Perform Update and Reset at the same time.

7 Event Log

The Event Log is an array of a data structure that is organised as a ring buffer. The location and size in the memory and the index of the actual position can be read out by parameter. The event log data can be read by using a TableReadRq.

7.1 Parameter

The following parameters are assigned to the Event Log.

Parameter Nr	Name	Comment
0x0010	EV_LOG_TABLE	Address of Event Log data structure
0x0011	EV_LOG_TBL_SIZE	Size of Array (number of elements)
0x0012	EV_LOG_TBL_POS	Zero based Index for next event log entry.
0x0013	EV_LOG_CATEGORY	Select logging category
0x0014	EV_LOG_LEVEL	Select logging level

7.2 Data structure

This is the structure of an event log entry. This means that every event entry uses 8 bytes of data. The data are in Motorola byte order (msb first).

```
typedef struct dstEventLogEntry{
  uint16 usEventIdentifier;
  uint16 usAssignedData;
  uint32 ulTimestamp;
}tstEventLogEntry;
```

The whole event log is organised as an array. The data block is defined as follows:

```
struct dstEventLogEntry mg_stEventLogEntry[EV_LOG_TABLE_SIZE];
```

Description of the data Fields of the dstEventLogEntry.

usEventIdentifier	Event identification code
usAssignedData	16 bit of assigned event data.
ulTimestamp	Standby time of ECU in 0.2 sec ticks.

7.3 Getting the Event Log

It is necessary to be logged in to get the event log. To know the size and the position of the data structure the Parameters (0x0010, 0x0011 and 0x0012) have to be read. A TableReadRq is used to get the whole data structure. If event entries at index >= EV_LOG_TBL_POS are not zero they are the oldest event in the ring buffer. To have a right interpretation the ring buffer has to be reordered. This can be done by sorting the timestamp or by looking at the EV_LOG_TBL_POS. When finished with PARAM operations a log out should be performed.

7.4 Getting the Event Log incremental

If the Event Log shall be read out regularly it is possible to calculate the position and size of the 1 to 2 TableReadRq to get only the new events since the last read out. Therefore the EV_LOG_TBL_POS of the last read out has to be stored. Depending on the old and the actual position 1 or 2 RableReadRq have to be performed. The following table shows the decision to make.

No overflow in the ring buffer	**Overflow in the ring buffer**
An overflow has not occurred if (EV_LOG_TBL_POS > Old_POS)	An overflow has occurred if (EV_LOG_TBL_POS < Old_POS)
Only 1 TableReadRq is needed.	2 TableReadRq are needed.
Calculate Address: ADR = EV_LOG_TABLE + (Old_POS * 8)	Calculate Addresses: ADR1 =EV_LOG_TABLE + (Old_POS * 8) ADR2 = EV_LOG_TABLE
Calculate Length Length = (EV_LOG_TBL_POS - Old_POS) * 8	Calculate Lengths Length1 = (EV_LOG_TBL_SIZE - Old_POS) * 8 Length2= EV_LOG_TBL_POS * 8
Get incremental Event Table Data Get Table (ADR,Length)	Get incremental Event Table Data Get Table (ADR1,Length1) Get Table (ADR2,Length2)

8 Examples

Some explained trace examples of the PARAM communication.

8.1 HELLO

ID	DLC	Data	Comment
0x0400	8	89 A5 A5 A5 04 01 04 00	Hello(0x0401, 0x0400)

8.2 Log in / Log out

Log in

ID	DLC	Data	Comment
0x0400	8	03 00 00 00 E1 15 99 85	SetParam(0x0000, 0xE1159985)
0x0401	8	F0 00 00 00 00 00 00 00	StatRespond(DBG_ERR_ACK)
0x0400	8	03 00 00 01 1E EA 66 7A	SetParam(0x0001, 0x1EEA667A)
0x0401	8	F0 00 00 00 00 00 00 00	StatRespond(DBG_ERR_ACK)

Log out

ID	DLC	Data	Comment
0x0400	8	03 00 00 00 00 00 00 00	SetParam(0x0000, 0x00000000)
0x0401	8	F0 00 00 00 00 00 00 00	StatRespond(DBG_ERR_ACK)
0x0400	8	03 00 00 01 00 00 00 00	SetParam(0x0001, 0x00000000)
0x0401	8	F0 00 00 00 00 00 00 00	StatRespond(DBG_ERR_ACK)

8.3 Get / Set Parameter

Getting Serial Number and Production Date

ID	DLC	Data	Comment
0x0400	8	04 00 00 05 00 00 00 00	GetParam(0x0005) Serial Number
0x0401	8	84 00 00 05 00 00 03 E8	GetParamRespond(0x0005, 0x000003E8) Serial Number = 1000
0x0400	8	04 00 00 04 00 00 00 00	GetParam(0x0004) Production Date
0x0401	8	84 00 00 04 05 0B 07 DA	GetParamRespond(0x0004, 0x050B07DA) Date = 05.11.2010

Set parameter (float value)

ID	DLC	Data	Comment
0x0400	8	03 00 02 32 41 53 33 33	SetParam(0x0232, 0x41533333) Set Rotor offset to 13.2° (float value)
0x0401	8	F0 00 00 00 00 00 00 00	StatRespond(DBG_ERR_ACK)

Set parameter while access is not given

ID	DLC	Data	Comment
0x0400	8	03 00 02 32 41 53 33 33	SetParam(0x0232, 0x41533333) Set Rotor offset to 13.2° (float value)
0x0401	8	F0 00 00 04 00 00 00 00	StatRespond(DBG_ERR_ ACCESS)

Setting a parameter out of range

ID	DLC	Data	Comment
0x0400	8	03 00 02 32 43 3A 00 00	SetParam(0x0232, 0x433A0000) Set Rotor offset to 186° (float value)
0x0401	8	F0 00 00 11 FF FF FF FF	StatRespond(DBG_ERR_DATA_CHANGED ,0xFFFFFF) Data out of range → NaN has been written.

8.4 Get Event Log

ID	DLC	Data	Comment
0x0400	8	03 00 00 00 E1 15 99 85	Login
0x0401	8	F0 00 00 00 00 00 00 00	
0x0400	8	03 00 00 01 1E EA 66 7A	
0x0401	8	F0 00 00 00 00 00 00 00	
0x0400	8	04 00 00 11 00 00 00 00	GetParam(0x0011) EV_LOG_TBL_SIZE
0x0401	8	84 00 00 11 00 00 04 00	Event Log has a size of 0x400
0x0400	8	04 00 00 12 00 00 00 00	GetParam(0x0012) EV_LOG_TBL_POS
0x0401	8	84 00 00 12 00 00 01 8B	Last event log index is 0x018B
0x0400	8	04 00 00 10 00 00 00 00	GetParam(0x0010) EV_LOG_TABLE
0x0401	8	84 00 00 10 02 00 80 00	Event Log is stored @ 0x02008000
0x0400	8	01 00 20 00 02 00 80 00	TableReadRq(0x2000,0x02008000) Get 8192 bytes of data beginning at memory address 0x02008000
0x0401	8	81 00 00 C2 01 00 70 00	1639 respond messages with 5 byte of data each.
0x0401	8	81 00 05 AC 46 23 60 00	They are sent at an interval of 4ms
0x0401	8	81 00 0A 00 01 00 AC 46	
0x0401	8	81 00 0F 23 42 05 00 10	
0x0401	8	81 00 14 00 AC 46 23 42	
…	…	…	
0x0401	8	81 0C 4E 9E 7C 42 05 00	Last Event Log in the table.
0x0401	8	81 0C 53 20 00 D4 9E 7C	INFO: State changed (State) (0x4205) Data: 0x0020 Time 32d 06:07:20.4
0x0401	8	81 0C 58 00 00 00 00 00	0x18B X 8 = 0xC58 → next free event log
…	…	…	
0x0401	8	81 1F F4 00 00 00 00 00	
0x0401	8	81 1F F9 00 00 00 00 00	
0x0401	8	81 1F FE 00 00 00 00 00	Last message
0x0400	8	03 00 00 00 00 00 00 00	Logout
0x0401	8	F0 00 00 00 00 00 00 00	
0x0400	8	03 00 00 01 00 00 00 00	
0x0401	8	F0 00 00 00 00 00 00 00	