MicroControl

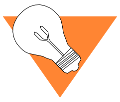
Systemhaus für Automatisierung

J1939 Protocol Stack

Version 3.00

Document conventions

For better handling of this manual the following icons and headlines are used:



This symbol marks a paragraph containing useful information about the protocol stack operation or giving hints on configuration.

This symbol marks a paragraph which describes actions to be executed by the user of the source code package.

This symbol marks a paragraph which explains possible danger. This danger might cause a damage to the system or damage to personnel. Read these sections carefully!

Keywords

Important keywords appear in the border column to help the reader when browsing through this document.

Syntax, Examples

`For function syntax and code examples the font face Source Code Pro is used.`

MicroControl GmbH & Co. KG
Junkersring 23
D-53844 Troisdorf
Fon: +49 / 2241 / 25 65 9 - 0
Fax: +49 / 2241 / 25 65 9 - 11
http://www.microcontrol.net

1. Scope

The J1939 Protocol Stack manual describes the Application Programming Interface (API) for access to the J1939 services. The API provides functionality for the J1939 following J1939 standards:

- J1939-21
- J1939-71
- J1939-73
- J1939-81

The J1939 Protocol Stack is independent from the used CAN hardware and operating system. Access to the CAN hardware is done via the CANpie API, which is available for a wide range of CAN controllers. The CANpie API is not subject of this manual.

1

1.1 References

/1/ SAE J1939-21, Data Link Layer
http://www.sae.org

/2/ SAE J1939-71, Vehicle Application Layer
http://www.sae.org

/3/ SAE J1939-73, Application Layer - Diagnostics
http://www.sae.org

/4/ SAE J1939-81, Network Management
http://www.sae.org

/5/ CANpie user manual, Version 3.0, MicroControl GmbH & Co. KG
http://www.microcontrol.net/en/source-code/canpie.html

1.2 Abbreviations

API	Application Programming Interface
CAN	Controller area network
CAN-ID	CAN identifier
CRC	Cyclic redundancy check
DM	Diagnostic Message
DTC	Diagnostic trouble code
ECU	Electronic control unit
LSB	Least significant bit/byte
MSB	Most significant bit/byte
OSI	Open systems interconnection
PGN	Parameter Group Number
RTR	Remote transmission request

1.3 Definitions

CAN base frame

message that contains up to 8 byte and is identified by 11 bits as defined in ISO 11898-1

CAN extended frame

message that contains up to 8 byte and is identified by 29 bits as defined in ISO 11898-1

CAN-ID

identifier for CAN data and remote frames as defined in ISO 11898-1

1.4 Introduction to CAN

CAN (Controller Area Network) is an international standard defined in the ISO 11898 for high speed and ISO 11519-2 for low speed.

CAN is based on a broadcast communication mechanism. This broadcast communication is achieved by using a message oriented transmission protocol. These messages are identified by using a message identifier. Such a message identifier has to be unique within the whole network and it defines not only the content but also the priority of the message.

The priority at which a message is transmitted compared to another less urgent message is specified by the identifier of each message. The priorities are laid down during system design in the form of corresponding binary values and cannot be changed dynamically. The identifier with the lowest binary number has the highest priority. Bus access conflicts are resolved by bit-wise arbitration on the identifiers involved by each node observing the bus level bit for bit. This happens in accordance with the "wired and" mechanism, by which the dominant state overwrites the recessive state. The competition for bus allocation is lost by all nodes with recessive transmission and dominant observation. All the "losers" automatically become receivers of the message with the highest priority and do not re-attempt transmission until the bus is available again.

The CAN protocol supports two message frame formats, the only essential difference being in the length of the identifier. The CAN standard frame, also known as CAN 2.0 A, supports a length of 11 bits for the identifier, and the CAN extended frame, also known as CAN 2.0 B, supports a length of 29 bits for the identifier.

1

1.5 Introduction to SAE J1939

The SAE J1939 communications network is developed for use in heavy-duty environments and suitable for horizontally integrated vehicle industries. The SAE J1939 communications network is applicable for light-duty, medium-duty, and heavy- duty vehicles used on-road or off-road, and for appropriate stationary applications which use vehicle derived components (e.g., generator sets). Vehicles of interest include, but are not limited to, on-highway and off-highway trucks and their trailers, construction equipment, and agricultural equipment and implements.

2. J1939 Overview

The following figure shows an overview of the J1939 functionality. Each J1939 service is described in a separate chapter.

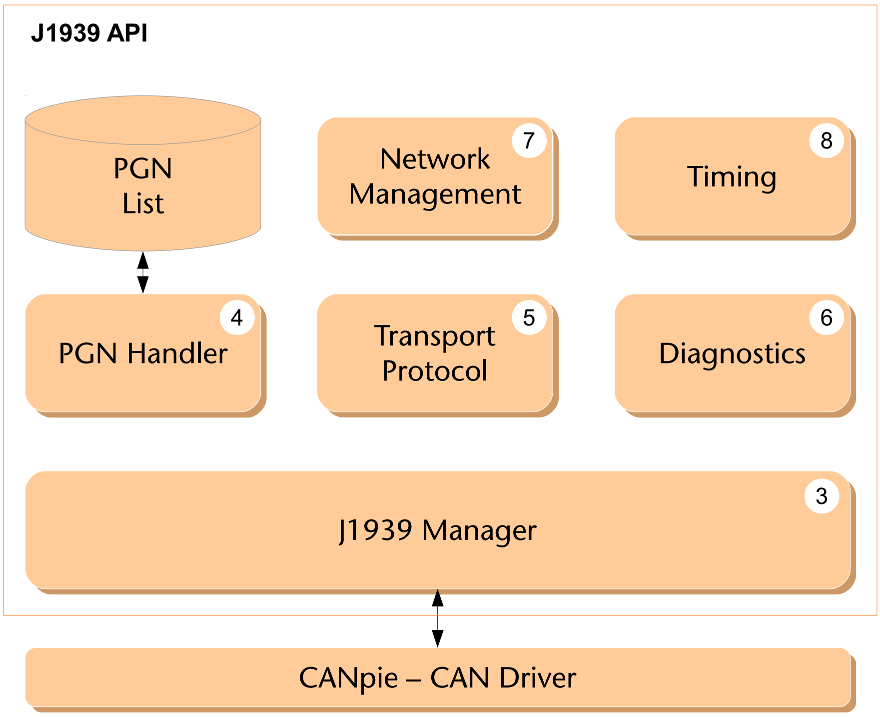


Fig. 1: J1939 overview

The J1939 Protocol Stack uses a well-defined CAN API (CANpie) to the CAN interface and thus can be adopted to any kind of CAN controller. The CANpie API is not described in this manual, for more information refer to /5/.

2.1 Naming Conventions

All functions, structures, defines and constant value of the J1939 stack have the prefix "**J1939_**". The following table shows the used nomenclature:

J1939 stack	*Prefix*
Functions	J1939_<service><name>
Enumeration	eJ1939_<name>
Structures	J1939_<name>_s
Defines	J1939_<service>_<name>
Error Codes	eJ1939_ERR_<name>

Table 1: Naming conventions

2

2.2 Message Router

The message router is responsible for reading and writing CAN messages between the J1939 Protocol Stack and the CAN bus. The CANpie API /5/ and its buffer concept is used to access the CAN interface on the different target platforms.

CAN messages are transmitted and received by different CAN message buffers. Depending on the J1939 service, a specific CAN message buffer will be selected.

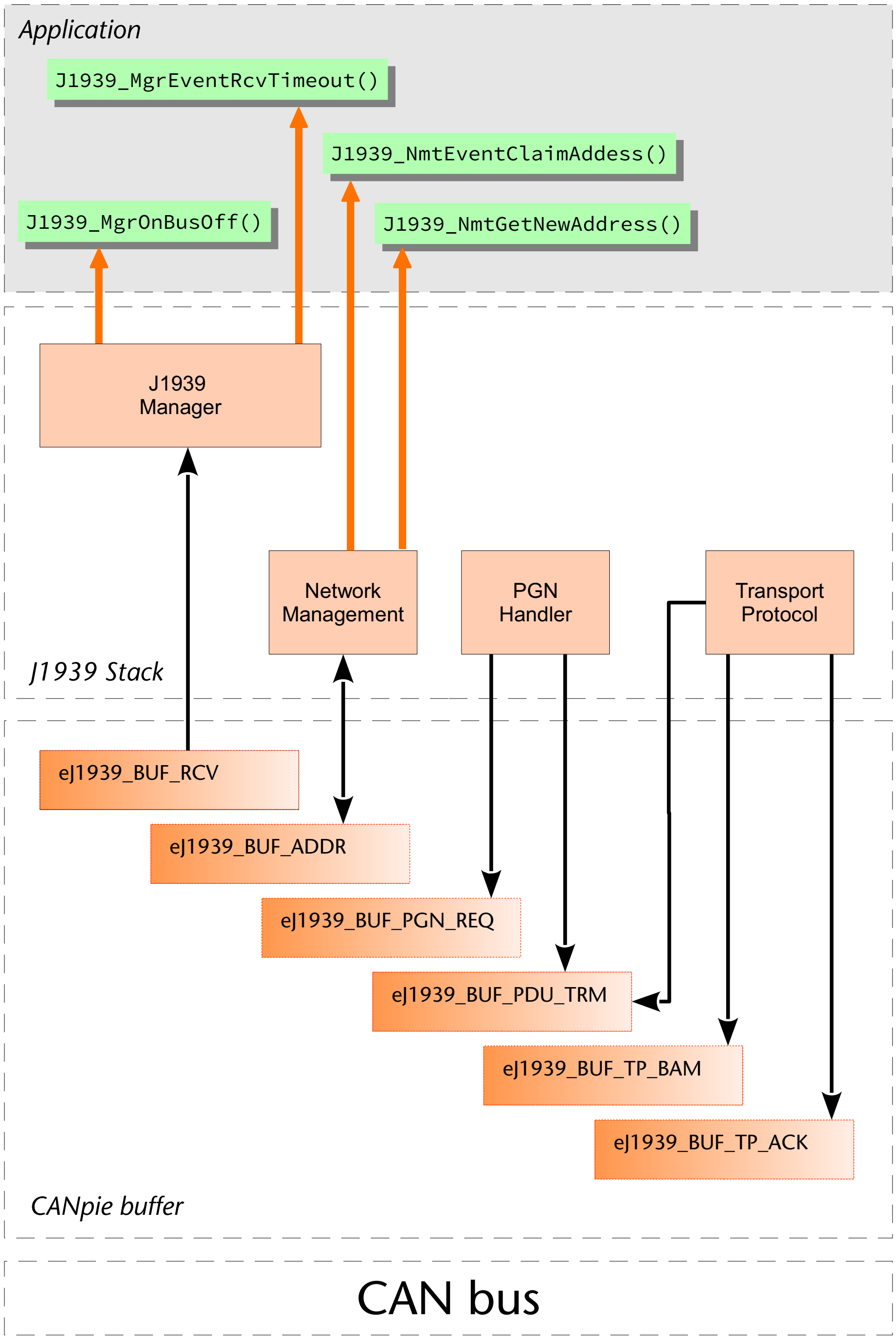


Fig. 2: Detail view of CANpie buffers and associated J1939 services

2.3 File Structure

All header files and implementation files of the J1939 Protocol Stack package are located in the **`source`** directory.

File	***Function***
j1939_bam.c / h	Broadcast announce messages
j1939_conf.h	Configuration file
j1939_diag_mem.c / h	Diagnostics, access to memory
j1939_diag.c / h	Diagnostics
j1939_mgr.c / h	J1939 manager
j1939_msg.c / h	CAN message support functions
j1939_nmt.c / h	Network management
j1939_pgn.c / h	Parameter group management
j1939_tmr.c / h	Timer services
j1939_tp.c / h	Transport protocol support functions
j1939_user.c	Application functions / event handler

Table 2: J1939 Protocol Stack files

The file **`j1939_user.c`** contains all variables and functions that require an adoption to the target system.

2.4 Initialisation Process

The J1939 Protocol Stack is initialised by calling **J1939_MgrInit()**. This function will setup the CAN controller and initialise all necessary services. Afterwards the protocol stack can be started by calling the **J1939_MgrStart()** function.

In summary three steps are necessary to run the J1939 Protocol Stack:

- Initialise J1939 Protocol Stack
- Start J1939 Protocol Stack
- Start the timer event function

```
void MyJ1939Init(void)
{
   //------------------------------------------------------
   // Initialize the J1939 stack, ECU address 1,
   // bit-rate 250 kBit/s
   //
   J1939_MgrInit(eCP_CHANNEL_1, 1, J1939_CONF_BITRATE_250K);

   //------------------------------------------------------
   // Start the J1939 stack,
   //
   J1939_MgrStart();

   //------------------------------------------------------
   // now the J1939 stack is initialized and has to be
   // triggered by calling J1939_MgrTmrEvent() with
   // a cycle time of J1939_TIMER_PERIOD

}
```

Example 1: Initialization process

The initialisation functions of the J1939 Protocol Stack have to be executed prior to any other API functions.

3. J1939 Manager

The J1939 Manager covers the initialization and control of the protocol stack. It also manages the initialization of the CAN interface via the CANpie driver.

3.1 Initialisation

The J1939 protocol stack is initialized by calling the two functions **J1939_MgrInit()** and **J1939_MgrStart()**.

3

```
//-------------------------------------------------------
// Initialize the J1939 Protocol Stack,
// use ECU address 1
//
J1939_MgrInit(eCP_CHANNEL_1, 1, J1939_CONF_BITRATE_250K);

//-------------------------------------------------------
// initialization for timer and other peripheral
// can be done here


//-------------------------------------------------------
// Start the J1939 Protocol Stack,
//
J139_MgrStart();
```

Example 2: Initialization of J1939 Protocol Stack

After calling **J1939_MgrStart()** the J1939 Protocol Stack is running and an Address Claiming Message (ACM) is send on the CAN bus (i.e. the identifier $18EEFF00_h$ + ECU-address).

The next example shows a complete generic initialisation of the protocol inside the main function. Additional functions for the microcontroller and timer are provided by the MicroControl Library (MCL), they are shown for a better understanding of the example code.

```
//--------------------------------------------------------
// Initialize the target CPU
//
McCpuInit();

//--------------------------------------------------------
// Initialize the J1939 Protocol Stack
//
J1939_MgrInit(eCP_CHANNEL_1, 1, J1939_CONF_BITRATE_250K);

//--------------------------------------------------------
// Initialize the timer resource on the target CPU
//
McTmrInit();
McTmrFunctionInit(J1939_TmrEvent,
                  McTmrTimeToTicks(1000),
                  eTMR_CTRL_START);
//--------------------------------------------------------
// Start the J1939 stack
//
J139_MgrStart();

//--------------------------------------------------------
// this is the main loop of the embedded application
//
while (1)
{
   //-----------------------------------------------------
   // check the result of the J1939 manager call
   //
   if (J1939_MgrProcess() != eJ1939_ERR_RESET)
   {

   }
   else
   {
      J1939_MgrRelease();
  }
}  // end while (1)
```

Example 3: Complete generic initialization of J1939 Protocol Stack

3.2 Manager Configuration Options

The file ***j1939_conf.h*** contains definitions for the configuration of the Manager module. Please set the symbols to an appropriate value in order to achieve a specific J1939 Manager behaviour. For a detailed specification please refer to the HTML documentation.

3.2.1 J1939_MGR_INT

With this symbol it is possible to switch the CAN message handler (message reception) between Polling- and IRQ-mode. In Polling mode the messages are read from the buffer during the main loop. The default mode is the IRQ-mode: received CAN messages are processed inside the CAN IRQ-handler.

Prior to changing this symbol make sure that the CANpie driver supports the requested method.

3.2.2 J1939_TMR_INT

With this symbol it is possible to switch the timer function between Polling- and IRQ-mode. In Polling mode the timer value is checked within the main loop. The default mode is the IRQ-mode: the function `J1939_TmrEvent()` is called within the timer interrupt.

3.3 Manager Functions

The manager functions (prefix `J1939_Mgr`) provide general control over the J1939 Protocol Stack.

Function name	Description
`J1939_MgrEventRcvTimeout()`	PDU reception time-out
`J1939_MgrGetAddress()`	Read actual ECU address setting
`J1939_MgrInit()`	Initialise J1939 Protocol Stack and CAN interface
`J1939_MgrOnBusOff()`	CAN bus status change to Bus-Off, Callback function (j1939_user.c)
`J1939_MgrRelease()`	Shutdown J1939 Protocol Stack and CAN interface
`J1939_MgrStart()`	Start J1939 Protocol Stack

Table 3: Functions of J1939 Manager

3.3.1 J1939_MgrEventRcvTimeout

Syntax

```
void J1939_MgrEventRcvTimeout(uint32_t uwPgnV)
```

Function

This function is called by the J1939 Protocol Stack if a time-out of a receive PDU occurs. The parameter **uwPgnV** defines the Parameter Group Number of the PDU where the time-out occurred.

User

Since the behaviour of this function is application specific, the implementation is available in the file **j1939_user.c**.

3

Parameters

uwPgnV Parameter Group Number

Return value

None

3.3.2 J1939_MgrGetAddress

Syntax

```
uint8_t J1939_MgrGetAddress(void)
```

Function

This function reads the actual ECU address setting. The code of this function has to be adopted to the target inside the **j1939_user.c** file. The function returns a value in the range from 0 to 254.

User

Since the behaviour of this function is application specific, the implementation is available in the file **j1939_user.c**.

Parameters

None

Return value

ECU address

3

3.3.3 J1939_MgrInit

Syntax

```
uint8_t J1939_MgrInit(
    uint8_t         ubCanIfV,
    uint8_t         ubAddressV,
    uint16_t        uwConfigV)
```

Function

Initialize J1939 Protocol Stack

This function initialises the J1939 Protocol Stack and must be called prior to any other function. The function assigns the J1939 Protocol Stack to the CAN interface given by **ubCanIfV** using the ECU address given by **ubAddressV**.

The parameter **uwConfigV** can have the following values:

Value	Description
J1939_CONF_BITRATE_250K	Start J1939 Protocol Stack with a bit-rate value of 250 kBit/s
J1939_CONF_BITRATE_500K	Start J1939 Protocol Stack with a bit-rate value of 500 kBit/s
J1939_CONF_FD	Start J1939 Protocol Stack in CAN FD mode

Table 4: Configuration of J1939 Protocol Stack

The usage of this function is shown by an example in "Initialisation" on page 13.

Parameters

ubCanIfV	CAN interface index
ubAddressV	ECU address
uwConfigV	Stack configuration options

Return value

On success the value **eJ1939_ERR_NONE** is returned.

3.3.4 J1939_MgrOnBusOff

Syntax

```
void J1939_MgrOnBusOff(void)
```

Function

This function handles a bus-off condition. The code of this function has to be adopted to the specific target inside the **j1939_user.c** file.

User

Since the behaviour of this function is application specific, the implementation is available in the file **j1939_user.c**.

3

Parameters None

Return value None

3.3.5 J1939_MgrRelease

Syntax `uint8_t J1939_MgrRelease(void)`

Function Shut down the J1939 Protocol Stack

This function performs a shut down of the J1939 Protocol Stack.

3

Parameters None

Return value On success the value **`eJ1939_ERR_NONE`** is returned.

3.3.6 J1939_MgrStart

Syntax `int8_t J1939_MgrStart(void)`

Function Start the J1939 Protocol Stack

This functions starts the J1939 Protocol Stack. An address claiming message (ACM, ID = $18EEFF00_h$ + ECU-address) is generated on the CAN bus.

Parameters None

Return value On success the value **`eJ1939_ERR_NONE`** is returned.

4. J1939 Parameter Group Number

A parameter group (PG) is a set of parameters belonging to the same topic and sharing the same transmission rate. The definition of the application relevant parameter groups and parameters can be found in application layer document /2/.

The length of a parameter group is not limited to the length of a CAN frame. Usually a parameter group has a minimum length of 8 bytes up to 1785 bytes.

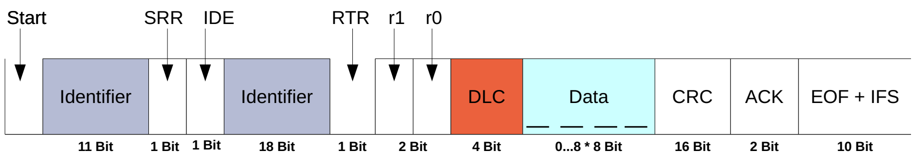


Fig. 3: J1939 Protocol Data Unit (PDU)

The CAN identifier of a J1939 Protocol Data Unit (PDU) contains a Parameter Group Number (PGN), source address, priority, data page bit, extended data page bit and a destination address (only for a peer-to-peer PG, i.e. PDU1 format).

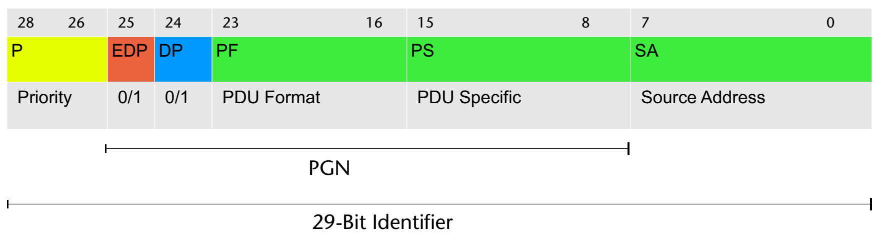


Fig. 4: J1939 fields inside J1939 PDU

4

4.1 PGN List

Within the J1939 Protocol Stack all available PGNs are kept inside a global list (**atsPgnListG[]**). Each entry of the PGN list is defined by the structure **J1939_PGN_Entry_s**.

```
struct J1939_PGN_Entry_s {

   /*! Holds additional information for PGN              */
   uint16_t          uwControl;

   /*!   Parameter group number                          */
   uint32_t          ulPGN;

   /*!   Cycle time in ticks (counter reload value)      */
   uint16_t          uwCycle;

   /*!   Counter                                         */
   uint16_t          uwCount;

   /*!   Source or Destination address                   */
   uint8_t           ubAddr;

   uint8_t           ubReserved;

   uint16_t          uwDataSize;

   /*! Pointer to handler function                       */
   PgnHandler_fn     pfnPgnHandler;

};
```

Example 4: PGN entry

The handler **PgnHandler_fn** is called by the stack if a PGN is available in the CA. The lower 11 bits of the parameter **uwSizeV** denote the size of the data to be transmitted.

Within the handler the mask **J1939_PGN_HANDLER_SIZE_MASK** can be used to retrieve this data size.

The highest bit of the parameter **uwSizeV** defines the data direction (see definition **J1939_PGN_HANDLER_WRITE**).

```
typedef int16_t (* PgnHandler_fn) (uint8_t * pubDataV,
                                   uint16_t  uwSizeV,
                                   uint8_t   ubSrcAddrV);
```

Example 5: PGN handler

4.2 PGN Functions

The PGN functions of the J1939 Protocol Stack have the prefix **J1939_Pgn** and are located in the file **j1939_pgn.c** within the source directory.

Function name	Description
`J1939_PgnAppEvent()`	PDU reception time-out
`J1939_PgnRequest()`	Read actual ECU address setting
`J1939_PgnSetCycleTime()`	Initialise J1939 Protocol Stack and CAN interface
`J1939_PgnSetDataSize()`	CAN bus status change to Bus-Off, Callback function (j1939_user.c)

Table 5: Functions for J1939 PGN handling

4.2.1 J1939_PgnAppEvent

Syntax `J1939_Status_tv J1939_PgnAppEvent(uint32_t ulPgnV)`

Function This function is used to trigger the transmission of a PGN by the application.

Parameters `ulPgnV` PGN value

Return value On success the value **eJ1939_ERR_NONE** is returned.

4.2.2 J1939_PgnRequest

Syntax

```
J1939_Status_tv J1939_PgnRequest(
    uint32_t        ulPgnV,
    uint8_t         ubDestAddrV)
```

Function

The function sends a J1939 Request message (RQST) for a PGN defined by parameter **ulPgnV**. The Request message is send to an ECU with address **ubDestAddrV**.

Parameters

ulPgnV	PGN value
ubDestAddrV	Destination address

4

Return value

On success the value **eJ1939_ERR_NONE** is returned.

4.2.3 J1939_PgnSetCycleTime

Syntax

```
J1939_Status_tv J1939_PgnSetCycleTime(
    uint32_t        ulPgnV,
    uint16_t        uwTimeV)(
```

Function

The function modifies the cycle time of a PGN defined by the parameter **ulPgnV**. If the PGN is available in the global PGN list (**atsPgnListG**[]) the function returns **eJ1939_ERR_NONE**.

Parameters

ulPgnV	PGN value
uwTimeV	Cycle time

Return value

On success the value **eJ1939_ERR_NONE** is returned.

4.2.4 J1939_PgnSetDataSize

Syntax

```
J1939_Status_tv J1939_PgnSetDataSize(
    uint32_t        ulPgnV,
    uint16_t        uwSizeV)(
```

Function

The function modifies the data size of a PGN defined by the parameter **ulPgnV**. If the PGN is available in the global PGN list (**atsPgnListG**[]) the function returns **eJ1939_ERR_NONE**.

Parameters

ulPgnV	PGN value
uwSizeV	Data size

Return value

On success the value **eJ1939_ERR_NONE** is returned.

5. J1939 Transport Protocol

Parameter groups that contain more than 8 data bytes are transmitted by means of a transport protocol.

For peer-to-peer and broadcast transmission, there are two different protocols. The transport protocols utilize two special parameter groups which are used for the connection management (TP.CM) and the transmission of the data (TP.DT).

For broadcast transmission, the BAM (Broadcast Announce Message) protocol is used. Here, after a BAM-PG, the transmitter sends all data PGs at a minimum interval of 50ms.

5.1 Configuration Options

The file ***j1939_conf.h*** contains definitions for the configuration of the Network Management module. Please set the symbols to an appropriate value in order to achieve a specific J1939 timing behaviour. For a detailed specification please refer to the HTML documentation.

5.1.1 J1939_BAM_TRM_BUFFER_SIZE

This symbol defines the size (in byte) of the message buffer for BAM messages. The maximum value is 1792.

5.1.2 J1939_TP_RCV_BUFFER_SIZE

This symbol defines the size (in byte) for reception of TP.DT messages. The maximum value is 1792.

5.1.3 J1939_TP_TRM_BUFFER_SIZE

This symbol defines the size (in byte) for transmission of TP.DT messages. The maximum value is 1792.

5.2 Transport Protocol Functions

The Transport Protocol functions of the J1939 Protocol Stack have the prefix **J1939_Bam**. resp. **J1939_Tp**.

Function name	Description
`J1939_BamSetDelay()`	Delay between broadcast messages

Table 6: Transport Protocol functions

5.2.1 J1939_BamSetDelay

Syntax

```
void J1939_BamSetDelay(
    uint16_t        uwDelayTimeV)
```

5

Function

The function sets the delay between single broadcast messages. The parameter **uwDelayTimeV** denotes the time in milliseconds. The default (initial) value is 50 ms.

Parameters

uwDelayTimeV Delay between broadcast messages

Return value

None

6. J1939 Diagnostics

SAE J1939-73 Diagnostics Application Layer defines the SAE J1939 messages to accomplish diagnostic services and identifies the diagnostic connector to be used for the vehicle service tool interface. Diagnostic messages (DMs) provide the utility needed when the vehicle is being repaired. Diagnostic messages are also used during vehicle operation by the networked electronic control modules to allow them to report diagnostic information and self-compensate as appropriate, based on information received. Diagnostic messages include services such as periodically broadcasting active diagnostic trouble codes, identifying operator diagnostic lamp status, reading or clearing diagnostic trouble codes, reading or writing control module memory, providing a security function, stopping/starting message broadcasts, reporting diagnostic readiness, monitoring engine parametric data, etc. California, EPA, or EU regulated OBD requirements are satisfied with a subset of the specified connector and the defined messages. /3/

6.1 Configuration Options

The file ***j1939_conf.h*** contains definitions for the configuration of the Network Management module. Please set the symbols to an appropriate value in order to achieve a specific J1939 timing behaviour. For a detailed specification please refer to the HTML documentation.

6.1.1 J1939_DM1_DTC_SIZE

This symbol defines the number of diagnostic trouble codes (DTC) that can be transferred by a DM1 message.

6.1.2 J1939_DM2_DTC_SIZE

This symbol defines the number of diagnostic trouble codes (DTC) that can be transferred by a DM2 message.

6.1.3 J1939_DM16_BIN_SIZE

This symbol defines the maximum number of bytes that can be transferred by a DM16 message.

6.2 Diagnostic Functions

The Network Management functions of the J1939 Protocol Stack have the prefix `J1939_Diag`.

Function name	Description
J1939_DiagAddDtc()	Add diagnostic trouble code for DM1
J1939_DiagAddDtcPrev()	Add diagnostic trouble code for DM2
J1939_DiagClearDtc()	Clear diagnostic trouble code for DM1
J1939_DiagClearDtcPrev()	Clear diagnostic trouble code for DM2
J1939_DiagMemRequest()	Memory access function (DM16)
J1939_DiagSetLamp()	Set lamp code for DM1

Table 7: Diagnostic functions

6

6.2.1 J1939_DiagAddDtc

Syntax

```
void J1939_DiagAddDtc(
    uint32_t        ulSpnV,
    uint8_t         ubFmiV,
    uint8_t         ubCountV)
```

Function

This function adds a diagnostics trouble code to the buffer for DM1 messages.

Parameters

ulSpnV	Suspect parameter number
ubFmiV	Failure mode identifier
ubCountV	Failure occurrence count

Return value

None

6.2.2 J1939_DiagAddDtcPrev

Syntax

```
void J1939_DiagAddDtcPrev(
    uint32_t        ulSpnV,
    uint8_t         ubFmiV,
    uint8_t         ubCountV)
```

Function

This function adds a diagnostics trouble code to the buffer for DM2 messages.

Parameters

ulSpnV	Suspect parameter number
ubFmiV	Failure mode identifier
ubCountV	Failure occurrence count

Return value

None

6.2.3 J1939_DiagClearDtc

Syntax

```
void J1939_DiagClearDtc(void)
```

Function

This function clears the buffer for DM1 trouble codes. Transmission of DM1 messages is automatically stopped.

Parameters

None

Return value

None

6.2.4 J1939_DiagClearDtc

Syntax

```
void J1939_DiagClearDtc(void)
```

Function

This function clears the buffer for DM1 trouble codes. Transmission of DM1 messages is automatically stopped.

Parameters

None

Return value

None

6.2.5 J1939_DiagMemRequest

Syntax

```
void J1939_DiagMemRequest(
    uint8_t                 ubSrcAddrV,
    J1939_MemAccess_ts *    ptsMemoryV)
```

Function

This function is called upon the occurrence of a guarding event.

User

Since the behaviour of this function is application specific, the implementation is available in the file **j1939_user.c**.

Parameters

None

Return value

None

6.2.6 J1939_DiagSetLamp

Syntax

```
void J1939_DiagSetLamp(
    uint8_t         ubLampV,
    uint8_t         ubStatusV)
```

6

Function

The function sets the status of the lamp defined by the parameter **ubLampV**.

Parameters

ubLampV	Lamp defined by 1939-73
ubStatusV	Lamp status

Return value

None

7. J1939 Network Management

The SAE standard J1939-81 Network Management defines the processes and messages associated with managing the source addresses of applications communicating on an SAE J1939 network. Network management is concerned with the management of source addresses and the association of those addresses with an actual function and with the detection and reporting of network related errors. Due to the nature of management of source addresses, network management also specifies initialization processes, requirements for reaction to brief power outages and minimum requirements for ECUs on the network. /4/

7.1 Configuration Options

The file ***j1939_conf.h*** contains definitions for the configuration of the Network Management module. Please set the symbols to an appropriate value in order to achieve a specific J1939 timing behaviour. For a detailed specification please refer to the HTML documentation.

7.1.1 J1939_NAME_ARBITRARY_ADDRESS_CAPABLE

This field indicates whether a CA can use an arbitrary source address to resolve an address claim conflict. If this symbol is set to "1", the CA will resolve an address conflict with a CA whose NAME has a higher priority (lower numeric value) by selecting an arbitrary source address from the range 128 to 247 inclusive and claiming that source address.

7

7.1.2 J1939_NAME_ECU_INSTANCE

The ECU Instance is a 3-bit field that indicates which one of a group of electronic control modules associated with a given Function is being referenced. For example, in the case where a single engine is managed by two separate control units, each of which is attached to the same SAE J1939 network, the ECU Instance Field will be set to 0 for the first ECU and 1 for the second ECU.
Note that in the case of a single or first ECU for a particular CA, the instance field should be set to zero indicating the first instance.

7.1.3 J1939_NAME_FUNCTION

Function is an 8-bit field defined and assigned by the committee. When Function has a value of 0 to 127 (See J1939 top level document, Appendix B, Table B11), its definition is not dependent on any other field. When Function has a value greater than 127, its definition depends on Vehicle System. Function, when combined with the Industry Group and the Vehicle System fields identifies a common name for a specific controller. The common name formed from the combination does not imply any specific capabilities.

7.1.4 J1939_NAME_FUNCTION_INSTANCE

The Function Instance is a 5-bit field that identifies the particular occurrence of a Function on the same Vehicle System on a given network. Note that in the case of single or first Function of a particular type, the instance field should be set to zero indicating the first instance. Individual manufacturers and integrators are advised that some agreement in the interpretation and use of Function Instances may be necessary. As an example, consider an implementation consisting of two engines and two transmissions. It may be important that engine instance 0 is physically connected to transmission instance 0 and that engine instance 1 is physically connected to transmission instance 1.

7.1.5 J1939_NAME_INDUSTRY_GROUP

Industry Group is a 3-bit field defined and assigned by the committee. Industry Group definitions may be found in Appendix B.7 of the SAE J1939 base document. The Industry Group field identifies NAMEs associated with a particular industry that uses SAE J1939, for example: On-Highway Equipment or Agricultural Equipment.

7.1.6 J1939_NAME_MANUFACTURER

The Manufacturer Code is an 11-bit field that indicates which company was responsible for the production of the electronic control module for which this NAME is being referenced. Manufacturer codes are assigned by committee and may be found in the SAE J1939 base document. See J1939 (top-level document) Appendix B, Table B10. The Manufacturer Code field is not dependent on any other field in the NAME.

7.1.7 J1939_NAME_VEHICLE_SYSTEM

Vehicle System is a 7-bit field defined and assigned by the committee, which when combined with the Industry Group can be correlated to a common name. See J1939 (top-level document) Appendix B, Table B12. Vehicle System provides a common name for a group of functions within a connected network. Examples of Vehicle Systems for currently defined Industry Groups are "tractor" in the "Common" Industry Group, "Trailer" in the On-Highway Industry Group, and planter in the "Agricultural Equipment" Industry Group.

7.1.8 J1939_NAME_VEHICLE_SYSTEM_INSTANCE

Vehicle System Instance is a 4-bit field that is used to identify a particular occurrence of a particular Vehicle System within a connected network.
Note that in the case of single or first Vehicle System of a particular type, the instance field should be set to zero indicating the first instance.

7.2 Network Management Functions

The Network Management functions (prefix **J1939_Nmt**) provide access to the J1939 NMT service.

Function name	Description
`J1939_NmtEventClaimAddress()`	Callback handler for Address Claim message
`J1939_NmtGetIdentityNumber()`	Get Identity number
`J1939_NmtGetNewAddress()`	Get new CA address
`J1939_NmtSetName()`	Set J1939 NAME structure

Table 8: Functions of J1939 NMT service

7.2.1 J1939_NmtEventClaimAddess

Syntax

```
void J1939_NmtEventClaimAddess(
     uint8_t         ubSrcAddrV,
     uint8_t *       pubDataV)
```

Function

This function is called be the J1939 stack upon reception of an address claiming message (ACM) by other ECUs which have a different source address.

Parameters

ubSrcAddrV Source address of other ECU

pubDataV Pointer to ACM data

Return value

None

7.2.2 J1939_NmtGetIdentityNumber

Syntax

```
uint32_t J1939_NmtGetIdentityNumber(void)
```

Function

The function returns the Identity number, which is required for the NAME function. The function is located inside the **j1939_user.c** file.

Parameters

None

Return value

Identity Number

7.2.3 J1939_NmtGetNewAddress

Syntax

```
uint8_t J1939_NmtGetNewAddress(void)
```

Function

This function is called by the J1939 Protocol Stack when the symbol **`J1939_ADDRESS_SELF`** is set to 1 and the device fails to claim its address. The application can then return a new address here to the J1939 Protocol Stack. The function is located inside the **`j1939_user.c`** file.

Parameters

None

Return value

New ECU address

7.2.4 J1939_NmtSetName

Syntax

```
void J1939_NmtSetName(J1939_Name_ts * ptsNameV)
```

Function

The function sets a new NAME for the ECU. The fields are defined by the structure **`J1939_Name_ts`**. The function can be used to overwrite the pre-defined values defined inside the **`j1939_conf.h`** file.

7

Parameters

`ptsNameV`	Pointer to NAME structure

Return value

None

8. J1939 Timing

The are a number of J1939 services which require an internal timer (e.g. BAM, message cycle time). The timer values for these services are set in a multiple of 1 millisecond. The stack works internally with timer ticks. A timer tick has a resolution of 1 microsecond, the time span of a timer tick is set via the definition `J1939_TIMER_PERIOD` inside the j1939_conf.h file. The requested time values are converted into timer ticks and vice versa with the functions `J1939_TmrCalcTicks()` and `J1939_TmrCalcTime()` respectively.

8.1 Configuration Options

The file ***j1939_conf.h*** contains definitions for the configuration of the TIME module. Please set the symbols to an appropriate value in order to achieve a specific J1939 timing behaviour. For a detailed specification please refer to the HTML documentation.

8.1.1 J1939_TIMER_PERIOD

This symbol defines the period of the timer interrupt. The value is a multiple of 1 microsecond. It is used for timing services. Please set this value to the timer interrupt period of the target system.

Please note that the value must be at least 1000 [microseconds], because all J1939 services use a multiple of 1 millisecond.

8.2 Timing Functions

The timing functions (prefix **`J1939_Tmr`**) provide access to the J1939 time service.

Function name	Description
`J1939_TmrCalcTicks()`	Convert time value to timer ticks
`J1939_TmrCalcTime()`	Convert timer ticks to time value
`J1939_TmrEvent()`	Execute timer-based services

Table 9: Functions of J1939 Slave time service

8.2.1 J1939_TmrCalcTicks

Syntax `uint16_t J1939_TmrCalcTicks(uint16_t uwReqTimingV)`

Function The function calculates the number of required timer ticks based on the required time **`uwReqTimingV`** (in milliseconds) and the constant value **`J1939_TIMER_PERIOD`**.

Parameters **`uwReqTimingV`** Time value in millisecond

Return value Number of ticks

8

8.2.2 J1939_TmrCalcTime

Syntax `uint16_t J1939_TmrCalcTime(uint16_t uwTicksV)`

Function The function calculates the time (in milliseconds) based on the given number of timer ticks and the constant value **`J1939_TIMER_PERIOD`**.

Parameters **`uwTicksV`** Number of timer ticks

Return value Time in milliseconds

8.2.3 J1939_TmrEvent

Syntax `void J1939_TmrEvent(void)`

Function Execute Timer-based Services

This function must be called periodically by a timer resource of the target system. It is responsible to call J1939 services that depend on a timer (e.g. BAM).

Parameters None

Return value None

```
//------------------------------------------------------------//
// Timer interrupt service routine                            //
//                                                            //
//------------------------------------------------------------//
void MyTimerInterrupt(void)
{
   //... timer services of application ...
   //--- call J1939 stack timer function -------------
   J1939_TmrEvent();

   //... retrigger the timer
}
```

Example 6: Example routine for J1939_TmrEvent()

In order to have periodical functions available (e.g. BAM), it is necessary to call the function **`J1939TmrEvent()`** cyclically. The cycle time is defined in microseconds by **`J1939_TIMER_PERIOD`** inside **`j1939_conf.h`** and must match the trigger time.

Typically **`J1939TmrEvent()`** will be called from a timer interrupt but it's also possible to call it inside the main loop. This behaviour is controlled by **`J1939_TMR_INT`** defined in **`j1939_conf.h`**.

8

8

A Source Code License

This source code license (hereinafter - "License”) is made between MicroControl GmbH & Co. KG, Junkersring 23, 53844 Troisdorf, Germany (hereinafter - "MicroControl") and you, the customer (hereinafter - "Licensee"). MicroControl and Licensee agree as follows:

1. DEFINITIONS

1.1 For purposes of this License, the following definitions shall apply:

(a) "Software" shall mean the particular Software product purchased by Licensee from MicroControl.

(b) "Source Code" shall include computer programming code or any computer instructions necessary to compile the Software.

(c) "Derivative Works" means any software programs which are developed by Licensee and which incorporate or contain modifications of any part of Source Code, and including any revision, modification, translation (including compilation or recapitulation by computer), abridgment, condensation, expansion or any other form in which Source Code, may be recast, transformed or adapted.

(d) "Purpose" means the creation of bug-fixes, corrections, enhancements, revisions, modifications and adaptations of Source Code and addition of new user interfaces, features and functionality to the Software.

2. LICENSEE RIGHTS AND RESTRICTIONS

2.1 By completing this License and subject to the restrictions and consideration stated below, MicroControl grants Licensee a non-exclusive, non-transferable, perpetual right to:

(a) use and reproduce as many copies of the Source Code as are reasonably necessary only for the purpose of exercising the rights granted under this License;

(b) modify and create Derivative Works of the Source Code for the Purpose;

(c) use, reproduce, have reproduced, sell (via sub-license), distribute (via sub-license), perform or otherwise transfer (via sub-license), directly or through distributors or resellers, Derivative Works, only in object code format, that are consistent with the Purpose and subject to the reporting and audit provisions of the License.

2.2 No right is granted to Licensee hereunder to permit, authorize, license or sub-license any third party to view or use the Source Code.

2.3 No right is granted to Licensee hereunder to permit, authorize, license or sub-license any company branches or company subsidiary to view or use the Source Code.

2.4 No right is granted to Licensee hereunder to sell, distribute, make available, publish or otherwise transfer the Source Code except as provided in section 2.1 above.

2.5 Licensee shall not use the Source Code for anything other than its intended, legitimate, and legal purpose.

2.6 Licensee shall not employ Source Code in any way that competes either directly or indirectly with MicroControl including but not limited to creation of Derivative Works that compete either directly or indirectly with Software.

2.7 Licensee shall not use the Source Code in any manner not specifically permitted under this License.

2.8 No right is granted under any patents, copyrights, trade secrets, trademarks or other proprietary rights of MicroControl, except as expressly granted herein.

2.9 The terms of this License entitles Licensee to receive support and maintenance services from MicroControl with respect to the Source Code for one year after date of purchase.

3. OWNERSHIP

3.1 MicroControl maintains title and ownership of all copyright interests in the Source Code.

4. CONFIDENTIALITY

4.1 Each party will protect the other's Confidential Information from unauthorized dissemination and use the same degree of care that such party uses to protect its own like information, but in no event less than a reasonable degree of care. Neither party will disclose to third parties the other's Confidential Information without the prior written consent of the other party. Neither party will use the other's Confidential Information for purposes other than those necessary to directly further the purposes of this License. Notwithstanding the foregoing, either party may use or disclose Confidential Information to the extent such party is legally compelled to disclose such Confidential Information provided, however, that prior to any such compelled disclosure, the disclosing party will notify the non-disclosing party and will cooperate fully with the non-disclosing party in protecting against any such disclosure and/or obtaining a protective order narrowing the scope of such disclosure and/or use of the Confidential Information. The parties agree that any breach of this Section would cause irreparable harm to the disclosing party for which monetary damages would not be adequate and therefore, the parties agree that in the event of a breach of this Section 4.1, the disclosing party shall be entitled to equitable relief in addition to any remedies it may have hereunder or at law.

4.2 In additional to the provisions of Section 4.1 above, Licensee acknowledges that the Source Code (and to the extent containing MicroControl trade secrets, the Licensee Derivative Works) constitutes a valuable asset of MicroControl and therefore agrees that only the following Licensee employees shall have access to the Source Code and the source code to the Licensee Derivative Works: those employees:

(a) employees who have a need for such access to accomplish the purposes of the distribution rights and license grants specified in Section 2 above; and

(b) with whom Licensee has a legally enforceable obligation that precludes disclosure of third-party proprietary information and is otherwise sufficient to enable Licensee to comply with all the provisions of this License. Licensee shall not grant any other individual or entity access to the Source Code.

4.3 Licensee shall implement reasonable security measures to prevent unauthorized use or disclosure of Source Code. Licensee agrees to segregate all Source Code and Confidential Information from its own confidential information and from the confidential information of others in order to prevent commingling.

4.4 Each party agrees to take appropriate action by instruction, agreement or otherwise with its employees, agents and contractors allowed access to the Confidential Information to satisfy its obligations under this Section 4.

5. LIMITATION ON LIABILITY

5.1 To the maximum extent permitted by applicable law, MicroControl shall not be liable to Licensee for any incidental, consequential, special, punitive or indirect damages, including without limitation, damages for loss of profits, business opportunity, data or use, incurred by Licensee or any third party, even if it has been advised of the possibility of such damages.

6. WARRANTY AND DISCLAIMER

6.1 MicroControl warrants that it has all right, power and authority to enter into this License and to grant the licenses granted hereunder.

6.2 Except as set forth in section 6.1 above, MicroControl makes no representations or warranties with respect to the Source Code. All express or implied representations and warranties, including without limitation any implied warranty of merchantability, of fitness for a particular purpose, of reliability or availability, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence, is hereby expressly disclaimed. Licensee specifically acknowledges that the Source Code is provided "as is" and may have bugs, errors, defects or deficiencies.

7. INDEMNITY

7.1 Licensee agrees to defend, indemnify and hold harmless MicroControl from and against any damages, costs and expenses (including, without limitation, reasonable attorneys fees and costs) arising from or relating to any third party claims, actions or demands that the sale, distribution or other transfer of any Derivative Works by Licensee or its distributors or resellers infringes the intellectual property rights of any third party.

8. TERMINATION.

8.1 This License is in effect so long as Licensee holds any copy of the Source Code on any Licensee computer or storage media either onsite or offsite.

8.2 Upon termination or expiration of this License for any reason whatsoever, Licensee shall immediately:

(a) cease all use of Product Source Code;

(b) make all reasonable effort to destroy and/or remove all copies of Source Code from Licensee computers and storage media; and

(c) return all Software, Source Code, Documentation, Confidential Information, and the source code to all Licensee Derivative Works and all related materials and copies thereof to MicroControl. In addition to the foregoing, Licensee agrees that it shall not, following termination of this License, act in any way to damage the reputation or goodwill of MicroControl or any Software, Licensee Derivative Work or other product.

8.3 Except as otherwise expressly provided herein, upon the expiration or termination of this License Licensee shall not be entitled to, and to the fullest extent permitted by law waives, any statutorily prescribed or other compensation, reimbursement or damages for loss of goodwill, clientele, prospective profits, investments or anticipated sales or commitments of any kind.

9. MISCELLANEOUS

9.1 Assignment and Effect: This License shall inure to the benefit of and be binding upon both parties, as well as their employees, employers, agents, parents, subsidiaries, representatives, licensees, and assigns.

9.2 Modifications: There will be no modifications, alterations, or amendments to this License, unless both parties agree in writing.

9.3 Governing Law: This License shall be governed by and construed under the laws of the Federal Republic of Germany.

9.4 Jurisdiction and Venue: Should any dispute arise under the terms of this License, such dispute will finally be solved under the procedure established by the laws of the Federal Republic of Germany in the German court according to the place of domicile of MicroControl.

9.5 Transfer of Rights: Without prejudice to any other rights, MicroControl shall have the right to transfer any rights and/or obligations hereunder to any third party.

Disclaimers

Life support — Products and software described in this manual are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. MicroControl customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify MicroControl for any damages resulting from such application.

Right to make changes — MicroControl reserves the right to make changes in the products - including circuits and/or software - described or contained herein in order to improve design and/or performance. MicroControl assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Copyright

No part of this functional specification may be copied, transmitted or stored in a retrieval system or reproduced in any way including, but not limited to, photography, magnetic, optic or other recording means, without prior written permission from MicroControl GmbH & Co. KG.

© 2018 MicroControl GmbH & Co. KG, Troisdorf

MicroControl
Systemhaus für Automatisierung

MicroControl GmbH & Co. KG
Junkersring 23
D-53844 Troisdorf
Fon: +49 / 2241 / 25 65 9 - 0
Fax: +49 / 2241 / 25 65 9 - 11
http://www.microcontrol.net