The Embedded I/O Company

TEWS TECHNOLOGIES

TIP903-SW-42

VxWorks Device Driver

3 Channel Extended CAN Bus

Version 3.0.x

User Manual

Issue 3.0.0

December 2011

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

TIP903-SW-42

VxWorks Device Driver

3 Channel Extended CAN Bus

Supported Modules:
TIP903-10

Issue	Description	Date
1.0	First Issue	April 1997
1.1	General Revision	November 2003
1.1.0	IPAC Carrier Driver Support	September 22, 2005
2.0.0	New Initialization Functions, File List Changed	June 21, 2006
3.0.0	64-Bit and SMP Support added, General Revision	December 14, 2011

Table of Contents

1 Introduction

1.1 Device Driver

The TIP903-SW-42 VxWorks device driver software allows the operation of the TIP903 IPAC conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), close*()* and *ioctl()* functions.

This driver invokes a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

To prevent the application program for losing data, incoming messages will be stored in a message FIFO with a depth of 100 messages.

The TIP903-SW-42 device driver supports the following features:

- extended and standard message frames
- acceptance filtering
- message objects
- remote frame requests etc.
- support for legacy and VxBus IPAC carrier driver
- SMP Support
- 64-Bit Support

The TIP903-SW-42 supports the modules listed below:

TIP903-10	3 Channel Extended CAN Bus	Industry Pack®

To get more information about the features and use of TIP903 devices it is recommended to read the manuals listed below.

TIP903 User manual
TIP903 Engineering Manual
CARRIER-SW-42 IPAC Carrier User Manual
Architectural Overview of the Intel 82527 CAN controller

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-42 is part of this TIP903-SW-42 distribution. It is located in directory CARRIER-SW-42 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-42 User Manual for a detailed description how to install and setup the CARRIER-SW-42 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP903-SW-42':

tip903drv.c	TIP903 device driver source
tip903def.h	TIP903 driver include file
tip903.h	TIP903 include file for driver and application
tip903exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP903-SW-42-3.0.0.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include Device Driver in VxWorks Projects

For including the TIP903-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Copy needed files from the distribution media into a subdirectory to your project directory.

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic. A file select box appears, and the driver files can be selected.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 System Resource Requirements

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	depends on FIFO size
Stack	< 1 KB	---
Semaphores	0	4

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O System Functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tip903Drv

NAME

tip903Drv - installs the TIP903 driver in the I/O system

SYNOPSIS

#include “tip903.h”

STATUS tip903Drv(void)

DESCRIPTION

This function installs the TIP903 driver in the I/O system.

A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tip903.h”


STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tip903Drv();
if (result == ERROR)
{
     /* Error handling */
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tip903DevCreate

NAME

tip903DevCreate – Add a TIP903 device to the VxWorks system

SYNOPSIS

```
#include “tip903.h”

STATUS tip903DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

DESCRIPTION

This routine adds the selected device (CAN channel) to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This parameter specifies the desired CAN channel. Each TIP903 contains three independent CAN channels. The CAN channels of all TIP903 modules in the system are enumerated in ascending order beginning by 0. The order of TIP903 modules in the system depends on the search order of the IPAC carrier driver

Example with two TIP903 modules in the system

Device Index	TIP903 CAN Channel
0	1st CAN channel on 1st TIP903
1	2nd CAN channel on 1st TIP903
2	3rd CAN channel on 1st TIP903
3	1st CAN channel on 2nd TIP903
4	2nd CAN channel on 2nd TIP903
5	3rd CAN channel on 2nd TIP903

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter is unused and should be set to NULL.

EXAMPLE

```
#include “tip903.h”

STATUS    result;


/*-------------------------------------------------------
  Create the device "/tip903/0" for the first CAN channel on
  the first TIP903 module
  -------------------------------------------------------*/

result = tip903DevCreate("/tip903/0", 0, 0, NULL);

if (result == ERROR)
{
     /* Error occurred when creating the device */
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_ioLib_NO_DRIVER	The TIP903 driver has not been started
EINVAL	Input parameters are invalid
EISCONN	The device has been created already

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4 I/O Functions

4.1 open

NAME

open - open a device or file.

SYNOPSIS

```
int open
(
        const char  *name,
        int         flags,
        int         mode
)
```

DESCRIPTION

Before I/O can be performed to the TIP903 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened, the name specified in tip903DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tip903/0" for I/O
  ----------------------------------------*/
fd = open("/tip903/0", 0, 0);
if (fd == ERROR)
{
     /* error handling */
}
```

RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close

NAME

close – close a device or file

SYNOPSIS

```
int close
(
      int           fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int  fd;
int  retval;

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == ERROR)
{
     /* Handle error */
}
```

RETURNS

A device descriptor number or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl

NAME

ioctl - performs an I/O control function.

SYNOPSIS

#include “tip903.h”

```
int ioctl
(
        int                     fd,
        int                     request,
        TIP903_IOCTL_ARG_T      arg
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
FIO_TIP903_READ	Read CAN message
FIO_TIP903_WRITE	Write CAN message
FIO_TIP903_BITTIMING	Configure bit timing
FIO_TIP903_SETFILTER	Set acceptance filter
FIO_TIP903_GETFILTER	Read acceptance filer
FIO_TIP903_BUSON	Set controller to bus on state
FIO_TIP903_BUSOFF	Set controller to bus off state
FIO_TIP903_FLUSH	Flush receive message FIFO
FIO_TIP903_DEFINE_MSG	Configure message object
FIO_TIP903_UPDATE_MSG	Update remote message object
FIO_TIP903_CANCEL_MSG	Cancel message object
FIO_TIP903_STATUS	Read state of message object

FIO_TIP903_CAN_STATUS	Read controller state

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.
To use this driver similarly on 32-bit and 64-bit systems typecasts for the ioctl argument (arg) should performed with the predefined type TIP903_IOCTL_ARG_T. This type is declared to int on 32-bit systems and to _Vx_ioctl_arg_t (long) on 64-bit systems.

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 FIO_TIP903_READ

This I/O control function reads a CAN message from the specified device. The function specific control parameter **arg** is a pointer on a TIP903_IO_BUFFER structure for this function.

If the device is blocked by another read request or no message is available in the read buffer, the requesting task will be blocked until a message is received or the request times out. If the flag TIP903_NOWAIT is set, read returns immediately.

```
typedef struct
{
        unsigned int        flags;
        int                 timeout;
        unsigned int        identifier;
        int                 extended;
        int                 length;
        unsigned char       data[8];
} TIP903_IO_BUFFER;
```

flags

This parameter defines a flag field used to control the read operation.

Flag	Description
TIP903_FLUSH	a flush of the device message FIFO will be performed before initiating the read request
TIP903_NOWAIT	return immediately, if the device is blocked by another FIO_TIP903_READ request or no CAN message is available

timeouts

This parameter specifies the timeout interval, in units of clock ticks. A timeout value of 0 means wait indefinitely.

identifier

This parameter returns the message identifier (standard or extended) of the message received.

extended

This parameter returns TRUE if identifier is extended or FALSE if the identifier is standard.

length

This parameter returns the size of message data received in bytes.

data

This array will be filled with the received message data bytes.

EXAMPLE

```
#include “tip903.h”

int                fd;
TIP903_IO_BUFFER   rw;
int                retval;

/*-------------------------------------------------------
  Read a message from a TIP903 device.
    - flush the input ring buffer before reading
    - if there is no message in the read buffer, the read
      request times out after 500 ticks
  -------------------------------------------------------*/
rw.flags      = TIP903_FLUSH;
rw.timeout    = 500;

retval = ioctl(fd, FIO_TIP903_READ, (TIP903_IOCTL_ARG_T)&rw);

if (result != ERROR || errnoGet() == S_tip903Drv_OVERRUN)
{
     /* process received message and check for overrun condition */
}
else
{
     /* handle the read error */
}
```

ERROR CODES

Error code	Description
S_tip903Drv_BUSOFF	The device is in bus off state
S_tip903Drv_BUSY	The device is busy
S_tip903Drv_TIMEOUT	The request timed out
S_tip903Drv_NODATA	There is no data available
S_tip903Drv_OVERRUN	One or more messages was overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.

4.3.2 FIO_TIP903_WRITE

This I/O control function writes a CAN message to specified device. The function specific control parameter **arg** is a pointer on a TIP903_IO_BUFFER structure for this function.

If the device is blocked by another write request the requesting task will be blocked until the request times out. If the flag TIP903_NOWAIT is set, write returns immediately.

```
typedef struct
{
        unsigned int        flags;
        int                 timeout;
        unsigned int        identifier;
        int                 extended;
        int                 length;
        unsigned char       data[8];
} TIP903_IO_BUFFER;
```

flags

This parameter defines a flag field used to control the write operation.

Flag	Description
TIP903_NOWAIT	Return immediately after writing the data to the device and do not wait for an acknowledge of the message.

timeouts

This parameter specifies the timeout interval, in units of clock ticks. A timeout value of 0 means wait indefinitely.

identifier

This parameter specifies the message identifier (standard or extended) of the message.

extended

This parameter specifies if an extended (TRUE) or standard (FALSE) message shall be sent.

length

This parameter specifies the size of message data in bytes.

data[]

This array supplies the message data bytes.

EXAMPLE

```
include             “tip903.h”

#define             HELLO     "HELLOOOO"

int                 fd;
TIP903_IO_BUFFER    rw;
int                 retval;


/*-----------------------------------------------------------------------
  Write a message to a TIP903 device.
  If there is a problem with the transmitter the write request times out
  -----------------------------------------------------------------------*/
rw.flags            = 0;
rw.timeout          = 100;        /* ticks */
rw.identifier       = 1234;       /* extended identifier */
rw.extended         = TRUE;
rw.length           = 8;

memcpy(rw.data, HELLO, 8);

retval = ioctl(fd, FIO_TIP903_WRITE, (TIP903_IOCTL_ARG_T)&rw);

if (retval != ERROR)
{
     /* message successfully sent */
}
else
{
     /* handle the write error */
}
```

ERROR CODES

Error Code	Description
S_tip903Drv_BUSOFF	The device is in bus off state
S_tip903Drv_IPARAM	An invalid parameter value specified
S_tip903Drv_TIMEOUT	The request timed out

4.3.3 FIO_TIP903_BITTIMING

This I/O control function modifies the Bit Timing Register of the CAN controller. The function specific control parameter **arg** is a pointer on a *TIP903_CNTRL_ARGS* structure.

```
typedef struct
{
        unsigned int        cmd;
        unsigned int        flags;
        unsigned long       arg;
} TIP903_CNTRL_ARGS;
```

cmd

This parameter is not used for this function.

flags

This parameter defines a flag field.

Flag	Description
TIP903_THREE_SAMPLES	the CAN bus is sampled three times per bit time instead of one time

arg

This parameter specifies the bit timing that will be set up. The specified value will be loaded into the Intel 82527 CAN controllers Bus Timing Register 0 and Bus Timing Register 1. Predefined bit timing values can be found in tip903.h.

EXAMPLE

```
#include “tip903.h”

int                fd;
TIP903_CNTRL_ARGS  dc;
int                retval;

/*-------------------------------------
  Setup the transfer rate to 500 Kbit/s
  -------------------------------------*/
dc.arg    = TIP903_500KBIT;
dc.flags  = 0;

retval = ioctl(fd, FIO_TIP903_BITTIMING, (TIP903_IOCTL_ARG_T)&dc);

if (retval == ERROR)
{
     /* handle the error */
}
```

4.3.4 FIO_TIP903_SETFILTER

This I/O control function sets up acceptance filter masks of the CAN controller.

The acceptance masks allow message objects to receive messages with a range of message identifiers instead of just a single message identifier. A "0" value means "don't care" or accept a "0" or "1" for that bit position. A "1" value means that the incoming bit value "must match" identically to the corresponding bit in the message identifier.

The function specific control parameter **arg** is a pointer on a *TIP903_ACCEPT_MASKS* structure.

```
typedef struct
{
        unsigned short      GlobalMaskStandard;
        unsigned int        GlobalMaskExtended;
        unsigned int        Message15Mask;
} TIP903_ACCEPT_MASKS;
```

GlobalMaskStandard

This parameter specifies the value for the Global Mask Standard Register. The Global Mask Standard Register applies only to messages using the standard CAN identifier. This 11 bit identifier appears in bit 5...15 of this parameter.

GlobalMaskExtended

This parameter specifies the value for the Global Mask Extended Register. The Global Mask Extended Register applies only to messages using the extended CAN identifier. This 29 bit identifier appears in bit 3...31 of this parameter.

Message15Mask

This parameter specifies the value for the Message 15 Mask Register. The Message 15 Mask Register is a local mask for message object 15. This 29 bit identifier appears in bit 3...31 of this parameter. The Message 15 Mask is "ANDed" with the Global Mask. This means that any bit defined as "don't care" in the Global Mask will automatically be a "don't care" bit for message 15 (see also Intel 82527 Architectural Overview).

EXAMPLE

```
#include “tip903.h”

int                     fd;
TIP903_ACCEPT_MASKS     AcceptMasks;
int                     retval;

/*-----------------------------
  Setup acceptance filter masks
  -----------------------------*/
AcceptMasks.GlobalMaskStandard   = 0xfe00;     /* bit 0..3 don't care */
AcceptMasks.GlobalMaskExtended   = 0xffffff80; /* bit 0..3 don't care */
AcceptMasks.Message15Mask        = 0xfffff800; /* bit 0..7 don't care */

retval = ioctl(fd, FIO_TIP903_SETFILTER, (TIP903_IOCTL_ARG_T)&AcceptMasks);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.5 FIO_TIP903_GETFILTER

This I/O control function returns the contents of the acceptance filter masks of the CAN controller. The function specific control parameter **arg** is a pointer on a *TIP903_ACCEPT_MASK* structure.

```
typedef struct
{
        unsigned short      GlobalMaskStandard;
        unsigned int        GlobalMaskExtended;
        unsigned int        Message15Mask;
} TIP903_ACCEPT_MASKS;
```

GlobalMaskStandard

This parameter returns the value of the Global Mask Standard Register. The Global Mask Standard Register applies only to messages using the standard CAN identifier. This 11 bit identifier appears in bit 5...15 of this parameter.

GlobalMaskExtended

This parameter returns the value of the Global Mask Extended Register. The Global Mask Extended Register applies only to messages using the extended CAN identifier. This 29 bit identifier appears in bit 3...31 of this parameter.

Message15Mask

This parameter returns the value of the Message 15 Mask Register. The Message 15 Mask Register is a local mask for message object 15. This 29 bit identifier appears in bit 3...31 of this parameter. The Message 15 Mask is "ANDed" with the Global Mask. This means that any bit defined as "don't care" in the Global Mask will automatically be a "don't care" bit for message 15 (see also Intel 82527 Architectural Overview).

EXAMPLE

```
#include “tip903.h”

int                     fd;
TIP903_ACCEPT_MASKS     AcceptMasks;
int                     retval;

/*----------------------------
  Get acceptance filter masks
  ----------------------------*/
retval = ioctl(fd, FIO_TIP903_GETFILTER, (TIP903_IOCTL_ARG_T)&AcceptMasks);

if (retval != ERROR)
{
     /* function succeeded */
     printf(“AcceptMasks: %04Xh/%08lXh/%08lXh\n”,
          AcceptMasks.GlobalMaskStandard,
          AcceptMasks.GlobalMaskExtended,
          AcceptMasks.Message15Mask);
}
else
{
     /* handle the error */
}
```

4.3.6 FIO_TIP903_BUSON

This I/O control function sets the CAN controller into bus on state. The function specific control parameter **arg** is not used for this function.

After an abnormal rate of occurrences of errors on the CAN bus, the CAN controller enters the bus off state. This I/O control function resets the init bit in the Control Register. The CAN controller begins the bus off recovery sequence. The bus recovery sequence resets transmit and receive error counters. If the CAN controller counts 128 packets of 11 consecutive recessive bits on the CAN bus, the bus off state is quitted.

EXAMPLE

```
#include “tip903.h”

int                fd;
int                retval;

/*----------------------
  Enter the bus on state
  ----------------------*/
retval = ioctl(fd, FIO_TIP903_BUSON, 0);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tip903Drv_BUSOFF	The controller cannot be set to bus on

4.3.7 FIO_TIP903_BUSOFF

This I/O control function sets the CAN controller into bus off state. The function specific control parameter **arg** is not used for this function.

After a successful execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function FIO_TIP903_BUSON is executed. Note: A pending write operation will be cancelled by a timeout.

Execute this control function before the last close to the CAN controller channel.

EXAMPLE

```
#include “tip903.h”

int               fd;
int               retval;

/*----------------------
  Enter the bus off state
  ----------------------*/
retval = ioctl(fd, FIO_TIP903_BUSOFF, 0);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.8 FIO_TIP903_FLUSH

This I/O control function flushes the receive message FIFO. All received messages are removed from the internal buffer. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip903.h”

int                fd;
int                retval;

/*------------------------------
  Flush the receive message FIFO
  ------------------------------*/
retval = ioctl(fd, FIO_TIP903_FLUSH, 0);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.3.9 FIO_TIP903_DEFINE_MSG

This I/O control function allocates and sets up user-definable message objects. The function specific control parameter **arg** is a pointer on a *TIP903_CNTRL_ARGS* structure.

```
typedef struct
{
        unsigned int        cmd;
        unsigned int        flags;
        unsigned long       arg;
} TIP903_CNTRL_ARGS;
```

cmd

This parameter is not used for this function.

flags

This parameter defines a flag field specifying the message type. The following combinations are allowed:

Value	Description
TIP903_RECEIVE	This is a receive message object, that will receive just a single message identifier or a range of message identifiers (see also Acceptance Mask). Receive message data can be read by the standard read function.
TIP903_RECEIVE | TIP903_REMOTE	This is also a receive object, but a remote frame is sent to request a remote node, to send the corresponding data.
TIP903_TRANSMIT	This is a transmit object. The transmission of the message data starts immediately after definition of the message object.
TIP903_TRANSMIT | TIP903_REMOTE	This is also a transmit object, but the transmission of the message data will be started if the corresponding remote frame (same identifier) was received.

arg

Is a pointer to a data structure (*TIP903_MSGDEF*) defining the new message object content:

```
typedef struct
{
    int             MsgNum;
    unsigned int    identifier;
    int             extended;
    int             length;
    unsigned char   data[8];
} TIP903_MSGDEF;
```

MsgNum

This parameter specifies the number of the message object to define. Allowed values are 1..13 and 15. Message object 14 is reserved for writing and can not be used.

identifier

This parameter specifies the message identifier.

extended

This parameter specifies if the message object shall be used with extended (*TRUE*) or standard (*FALSE*) identifiers.

length

This parameter is only used for transmit objects. The value specifies the length of the data message (0…8).

data[]

This array is only used for transmit objects. The array specifies the message that shall be sent.

A defined message object will be active until the I/O control function *FIO_TIP903_CANCEL_MSG* marks it as invalid to stop communication transactions.

EXAMPLE

```
#include “tip903.h”

int                fd;
TIP903_CNTRL_ARGS  dc;
TIP903_MSGDEF      MsgDef;
int                retval;

/*-----------------------------------------------
  Define message object 1
  receive message object, use extended message
  identifier, wait for receiving of a message with
  specified identifier.
  -----------------------------------------------*/
dc.arg             = (unsigned long)&MsgDef;
dc.flags           = TIP903_RECEIVE;

MsgDef.MsgNum      = 1;
MsgDef.identifier  = 10;
MsgDef.extended    = TRUE;

retval = ioctl(fd, FIO_TIP903_DEFINE_MSG, (TIP903_IOCTL_ARG_T)&dc);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

/*-----------------------------------------
  Define message object 2
  receive message object, use standard message
  identifier, send a remote frame to request a
  remote node to send the corresponding data.
  -----------------------------------------*/
dc.arg             = (unsigned long)&MsgDef;
dc.flags           = TIP903_RECEIVE | TIP903_REMOTE;

MsgDef.MsgNum      = 2;
MsgDef.identifier  = 100;
```

```
MsgDef.extended    = FALSE;

retval = ioctl(fd, FIO_TIP903_DEFINE_MSG, (TIP903_IOCTL_ARG_T)&dc);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

/*-----------------------------------------------
  Define message object 3
  transmit message object, use standard message
  identifier, start transmission.
  -----------------------------------------------*/
dc.arg             = (unsigned long)&MsgDef;
dc.flags           = TIP903_TRANSMIT;

MsgDef.MsgNum      = 3;
MsgDef.identifier  = 50;
MsgDef.extended    = FALSE;
MsgDef.length      = 1;
MsgDef.data[0]     = 'x';

retval = ioctl(fd, FIO_TIP903_DEFINE_MSG, (TIP903_IOCTL_ARG_T)&dc);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

```
/*-----------------------------------------------------
  Define message object 4
  transmit message object, use extended message
  identifier, start transmission if the corresponding
  remote frame (same identifier) was received.
  -----------------------------------------------------*/
dc.arg             = (unsigned long)&MsgDef;
dc.flags           = TIP903_TRANSMIT | TIP903_REMOTE;

MsgDef.MsgNum      = 4;
MsgDef.identifier  = 500;
MsgDef.extended    = TRUE;
MsgDef.length      = 1;
MsgDef.data[0]     = 0;

retval = ioctl(fd, FIO_TIP903_DEFINE_MSG, (TIP903_IOCTL_ARG_T)&dc);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tip903Drv_IMSGNUM	Invalid message number has been specified
S_tip903Drv_MSGBUSY	The message object is already in use
S_tip903Drv_IPARAM	Invalid flags or combinations specifies

4.3.10 FIO_TIP903_UPDATE_MSG

This I/O control function updates only the message data of a previously defined transmission object or starts transmission of a remote frame for receive message objects. The function specific control parameter **arg** is a pointer on a *TIP903_CNTRL_ARGS* structure.

```
typedef struct
{
        unsigned int        cmd;
        unsigned int        flags;
        unsigned long       arg;
} TIP903_CNTRL_ARGS;
```

cmd

This parameter is not used for this function.

flags

This parameter defines a flag field specifying the update type. If the flag TIP903_REMOTE is set, the transmission will be started by a request of a remote node, otherwise transmission of the message data starts immediately.

arg

Is a pointer to a data structure (*TIP903_MSGDEF*) defining the message object content:

```
typedef struct
{
        int             MsgNum;
        unsigned int    identifier;
        int             extended;
        int             length;
        unsigned char   data[8];
} TIP903_MSGDEF;
```

MsgNum

This parameter specifies the number of the message object. Allowed values are 1..13 and 15. Message object 14 is reserved for writing and can not be used.

identifier

This parameter is not used for this function.

extended

This parameter us not used for this function.

length

This parameter specifies the length of the new data message (0…8).

data[]

The array specifies the new message.

EXAMPLE

```
#include “tip903.h”

int                fd;
TIP903_CNTRL_ARGS  dc;
TIP903_MSGDEF      MsgDef;
int                retval;

/*-----------------------------------------------
  Update message object 3 and start transmission
  -----------------------------------------------*/
dc.arg             = (unsigned long)&MsgDef;
dc.flags           = 0;

MsgDef.MsgNum      = 3;
MsgDef.data[0]     = 'y';
MsgDef.length      = 1;

retval = ioctl(fd, FIO_TIP903_UPDATE_MSG, (TIP903_IOCTL_ARG_T)&dc);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

…
```

```
/*-----------------------------------------------
  Update message object 4
  Data will be sent by a request of a remote node
  -----------------------------------------------*/
dc.arg             = (unsigned long)&MsgDef;
dc.flags           = TIP903_REMOTE;

MsgDef.MsgNum      = 4;
MsgDef.data[0]     = 1;
MsgDef.length      = 1;

retval = ioctl(fd, FIO_TIP903_UPDATE_MSG, (TIP903_IOCTL_ARG_T)&dc);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tip903Drv_IMSGNUM	Invalid message number has been specified
S_tip903Drv_MSGBUSY	Transmission is active
S_tip903Drv_MSGNOTDEF	The message object has not been set up

4.3.11 FIO_TIP903_CANCEL_MSG

This I/O control function marks a specified message object as invalid and stops communication transactions. The function specific control parameter **arg** specifies the message object that shall be released. Allowed message object numbers are 1..13 and 15.

EXAMPLE

```
#include “tip903.h”

int                fd;
int                retval;

/*----------------------------------------
  Cancel message object 2 (not longer used)
  ----------------------------------------*/
retval = ioctl(fd, FIO_TIP903_CANCEL_MSG, 2);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tip903Drv_IMSGNUM	Invalid message number has been specified

4.3.12 FIO_TIP903_STATUS

This I/O control function returns the actual status of the specified transmission message object. The function specific parameter **arg** is a pointer on a *TIP903_STATUS* structure.

```
typedef struct
{
        unsigned int        message_sel;
        unsigned int        status;
} TIP903_STATUS;
```

message_sel

This parameter specifies the number of the message object. Allowed values are 1..13 and 15. Message object 14 is reserved for writing and can not be used.

status

This parameter returns the actual state of the specified message object. The status code is the same that would be returned by *read()* or *write()*.

EXAMPLE

```
#include “tip903.h”

int                 fd;
TIP903_STATUS       msg_stat;
int                 retval;

/*------------------------------
  Get status of message object 3
  ------------------------------*/
msg_stat.message_sel = 3;

retval = ioctl(fd, FIO_TIP903_STATUS, (TIP903_IOCTL_ARG_T)&msg_stat);

if (retval != ERROR)
{
     /* function succeeded */
     printf(“Object state: %08Xh”, msg_stat.status);
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tip903Drv_IMSGNUM	Invalid message number has been specified

4.3.13 FIO_TIP903_CAN_STATUS

This I/O control function the actual contents of the CAN Controller Status Register. The function specific control parameter **arg** is a pointer to an unsigned long value that will receive the content of the status register.

EXAMPLE

```
#include “tip903.h”

int                fd;
int                retval;
unsigned long      statVal;

/*-----------------------------------------------
  Get contents of CAN Controller Status Register
  -----------------------------------------------*/
retval = ioctl(fd, FIO_TIP903_CAN_STATUS, (TIP903_IOCTL_ARG_T)&statVal);

if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5 Appendix

5.1 Predefined Macros

Bit Timing Definitions

Name	Value	Speed	Cable Length
TIP903_1MBIT	0x0014	1 MBit/s	max. cable length: 36 m
TIP903_500KBIT	0x001c	500 KBit/s	max. cable length: 130 m
TIP903_250KBIT	0x011c	250 KBit/s	max. cable length: 270 m
TIP903_125KBIT	0x031c	125 KBit/s	max. cable length: 530 m
TIP903_100KBIT	0x432f	100 KBit/s	max. cable length: 620 m
TIP903_50KBIT	0x472f	50 KBit/s	max. cable length: 1.3 km
TIP903_20KBIT	0x532f	20 KBit/s	max. cable length: 3.3 km

I/O flags Definitions

Name	Value	Description
TIP903_RECEIVE	(1 << 0)	Message object direction is receive
TIP903_TRANSMIT	(1 << 1)	Message object direction is transmit
TIP903_REMOTE	(1 << 2)	If direction is *receive* a remote frame will be transmitted. If direction is *transmit* the message data will be send after the receive of a remote frame, which matches with the identifier.
TIP903_FLUSH	(1 << 3)	Flush the read message FIFO
TIP903_NOWAIT	(1 << 4)	Do not wait, if the device is blocked or no data are available for a read requests. Return immediately after starting of transmission for write requests.
TIP903_THREE_SAMPLES	(1 << 5)	Three samples are used for determining the valid bit value.

5.2 Predefined Status and Error Codes

Error Code	Value	Description
S_tip903Drv_IDLE	0x00000000	Status of the message object is idle
S_tip903Drv_NXIO	0x09030001	No TIP903 IP found at the specified base address
S_tip903Drv_IDEVICE	0x09030002	Invalid or duplicate minor device number
S_tip903Drv_ICMD	0x09030003	Unknown ioctl function code
S_tip903Drv_NOTINIT	0x09030004	Device was not initialized
S_tip903Drv_NOMEM	0x09030005	Unable to allocate memory
S_tip903Drv_TIMEOUT	0x09030006	I/O request times out
S_tip903Drv_BUSY	0x09030009	Device or message object is busy
S_tip903Drv_NOSEM	0x0903000A	Unable to create a semaphore
S_tip903Drv_NODATA	0x09030010	No data available, only possible if *TIP903_NOWAIT* option selected
S_tip903Drv_IPARAM	0x09030013	Invalid device initialization data
S_tip903Drv_PARAM_MISMATCH	0x09030014	Mismatch of common initialization parameter
S_tip903Drv_OVERRUN	0x09030015	Data overrun
S_tip903Drv_BUSOFF	0x09030016	Controller is in bus off state
S_tip903Drv_IMSGNUM	0x09030017	Invalid message object number
S_tip903Drv_MSGBUSY	0x09030018	Message object already defined
S_tip903Drv_MSGNOTDEF	0x09030019	Message object not defined