AcQuisition Technology bv

Headquarters:
Raadhuislaan 27a
5341 GL OSS
THE NETHERLANDS

Postal address:
P.O. Box 627
5340 AP OSS
THE NETHERLANDS

Phone: +31-412-651055
Fax: +31-412-651050
Email: info@acq.nl
WEB: http://www.acq.nl

M360

CANbus M-module

User Manual

Version 1.0

Disclaimer:

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. AcQuisition Technology does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. AcQuisition Technology products are not designed, intended, or authorized for use as components in systems intended to support or sustain life, or for any other application in which the failure of an AcQuisition Technology product could create a situation where personal injury or death may occur, including, but not limited to AcQuisition Technology products used in defence, transportation, medical or nuclear applications. Should the buyer purchase or use AcQuisition Technology products for any such unintended or unauthorized application, the buyer shall indemnify and hold AcQuisition Technology and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that AcQuisition Technology was negligent regarding the design or manufacture of the part.

Printed in The Netherlands.

CONTENTS

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

1. INTRODUCTION

1.1. VALIDITY OF THE MANUAL

The contents of this manual is valid for M360 revision 3.x (hardware rev. 3, firmware rev. x).
The software library is based on APIS, AcQ Platform Interface Software version 2.0 and up.

1.2. PURPOSE

This manual serves as instruction for the operation of the M360 CANbus M-module, the connection of peripheral devices, and the software library. Furthermore it gives the user additional information on configuration of the assembly.

1.3. SCOPE

Detailed information concerning the individual assemblies (data sheets etc.) Are not part of this manual. The bibliography can be found in the annex.

1.4. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

AcQ	AcQuisition Technology bv
APIS	AcQ Platform Interface Software
CANbus	Controller Area Network, fieldbus protocol
M-module	Mezzanine I/O concept according to the M-module specification

1.5. NOTES CONCERNING THE NOMENCLATURE

Hex numbers are marked with a leading “0x”-sign: for example: 0x20 or 0xff.

File names are represented in italic: *filename.txt*

Code example are printed in `courier` .

The jumpers are designated by a 'J', and a serial number. When specifying whether a jumper should be connected or removed it is referred to solely by this designation if it has only one position (e.g., 'J5 connected'). However, if the jumper has more than one position, it is also indicated which pins are connected to each other (e.g. 'J8,1-2'). Pin 1 of a jumper is always marked in the configuration diagram.

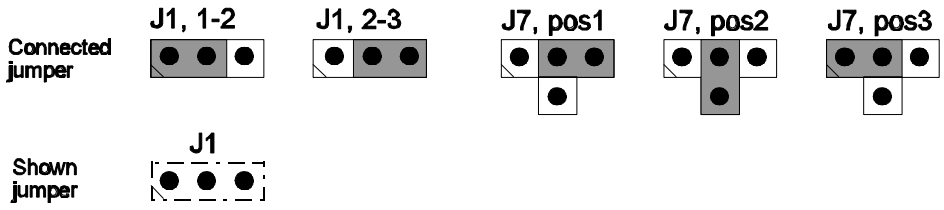

Figure 1 Example jumper nomenclature

In some illustrations jumpers are shown merely for purposes of orientation. In this case they are indicated with a dotted line. Their correct setting is described in another chapter.

Active-low signals are represented by a trailing asterisks (i.e. IACK*).

1.6. OVERVIEW

In chapter two a short description of the M360 CANbus M-module can be found. The next chapter covers the installation and setup of the module as well as the connection of the CANbus lines. In chapter 4 the functionality of the M360 CANbus M-module is described in detail. AcQ provides an APIS based ANSI C library for the M360 CANbus M-module, the software is described in chapter 5. Finally this document contains an Annex containing a bibliography, component image, technical data, document history and a programming example.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

2. PRODUCT OVERVIEW

2.1. INTRODUCTION

The M360 CANbus Controller is designed as a plug-on module with a M-module interface based on the SJA1000, a highly integrated stand-alone controller for the Controller Area Network (CAN).

CAN is used within automotive and general industrial environments. The M360 contains all necessary hardware for a high performance serial network communication. The M360 performs all the functions of the physical and data-link layers.

The use of the M360 CAN bus controller in an industrial environment, results in a reduction of wiring harness and an enhanced diagnostic and supervisory capability.

The software library is available, so an application can be programmed without knowing something of the hardware.

2.2. TECHNICAL OVERVIEW

- SJA1000, CAN bus stand-alone controller from Philips
- Galvanic isolation
- Programmable data transmission rate, up to 1 Mbit/s
- Two different physical CAN bus interfaces
- Identification EEPROM
- Two status Leds on front
- APIS based library

This page contains no essential data.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

3. INSTALLATION AND SETUP

3.1. UNPACKING THE HARDWARE

The hardware is shipped in an ESD protective container. Before unpacking the hardware, make sure that this takes place in an environment with controlled static electricity. The following recommendations should be followed:

- **!** Make sure your body is discharged to the static voltage level on the floor, table and system chassis by wearing a conductive wrist-chain connected to a common reference point.
- **!** If a conductive wrist-chain is not available, touch the surface where the board is to be put (like table, chassis etc.) before unpacking the board.
- **!** Leave the board only on surfaces with controlled static characteristics, i.e. specially designed anti static table covers.
- **!** If handling the board over to another person, touch this persons hand, wrist etc. to discharge any static potential.

IMPORTANT: Never put the hardware on top of the conductive plastic bag in which the hardware is shipped. The external surface of this bag is highly conductive and may cause rapid static discharge causing damage. (The internal surface of the bag is isolating.)

Inspect the hardware to verify that no mechanical damage appears to have occurred. Please report any discrepancies or damage to your distributor or to AcQuisition Technology immediately and do not install the hardware.

3.2. JUMPER SETTINGS

To work properly, the M360 CANbus controller module must be configured according to the particular circumstances.

3.2.1. SELECTING THE BUS TRANSCEIVER

Select the PCA82C250T CAN bus transceiver:

- **!** Place jumpers J5, J6, J8 and J9 on the 1-2 position.

Select the SN75176B RS-485 transceiver:

- **!** Place jumpers J5, J6, J8 and J9 on the 2-3 position.

3.2.2. GALVANIC ISOLATION

Select the galvanic isolation:
- ! Remove Jumpers J1, J2, J3 and J4.
- ! Select the isolated power source. Power is supplied by either the local DC/DC converter or an externally connected source.
  - ! Select the local DC/DC converter for the isolated power source:
    Place jumper J7 to position 3.
  - ! Or select the externally connected power source:
    Place jumper J7 to position 1.
- ! Insert two opto-couplers, type HP-7100, in the U8 (top up) and U9 (top down) position.
- ! If the external power supply is selected, connect it to the front D-sub connector after the module is mounted on the baseboard. The input voltage must be connected to pin 9 (external positive supply) and pin 3 (ground) and will be +7V < V+ < +13V.

Select the non galvanic isolation:
- ! Place jumpers J1, J2, J3 and J4.
- ! Place jumper J7 to position 2.

3.2.3. BUS TERMINATION

On both ends of the CAN bus the physical line must be terminated by a 124 Ohm resistance.
- ! Place jumper J10, if the M360 is at the end of the physical line.

3.2.4. EXAMPLE JUMPER SETTINGS

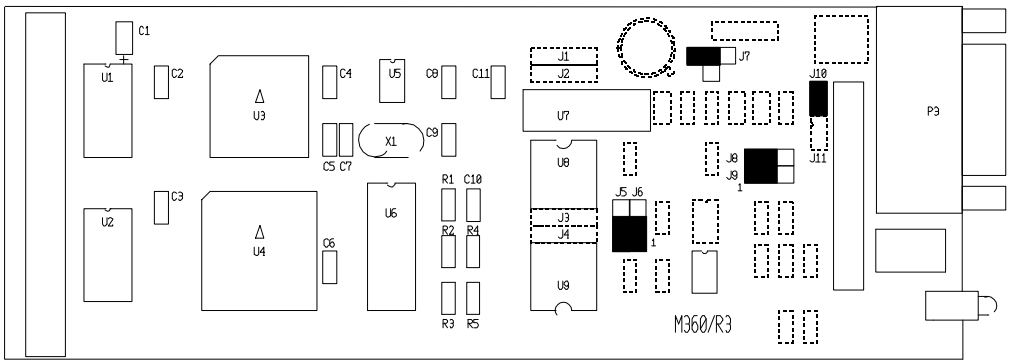

Figure 2 Example jumper settings

Figure 2 shows the jumper settings for the following case:

Standard PCA82C250T CAN bus transceiver selected,
Jumpers J5, J6, J8 and J9 are in position 1-2.
Galvanic isolation selected,
Opto-couplers(HP-7100) U8 and U9 are inserted and
jumpers J1, J2, J3 and J4 are removed and
jumper J7 is in position 3.
Bus terminated,
Jumper J10 is inserted.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

3.3. CONNECTING THE MODULE

The M360 features a 9 pole D-sub connector for interfacing with the CANbus. The M360 can also be connected with the CANbus through the 24 pole female header.

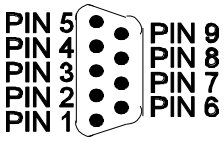

Figure 3 Front view 9 pole D-sub

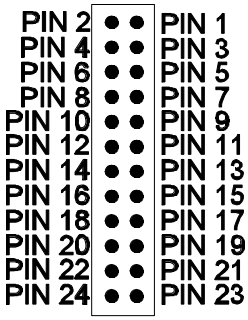

Figure 4 Top view 24 pole female header

The table below contains an overview of the connectors layout.

Signal	Description	9 pole D-sub connector	24 pole female header
CAN_L	CAN_L bus line (dominant low)	2	2
CAN_H	CAN_H bus line (dominant high)	7	1
GND	Ground (external)	3	5
VEXT	Optional external positive supply +7V < VEXT < +13V (dedicated for supply of transceiver and opto-couplers, if galvanic isolation of the bus node applies).	9	4
(GND)	Optional ground	6	6
+wireA	PCA82C250T high output	-	7
-wireA	PCA82C250T low output	-	8
+wireB	SN75176B A output	-	9
-wireB	SN75176B B output	-	10
TXOI	SJA1000 transmit line to driver	-	11
RXOI	SJA1000 receive line from driver	-	12
-	Reserved	1, 4, 5, 8	3, 13...24

3.4. SOFTWARE

M360 Example software is APIS based, therefore APIS support for the target platform is required for code generation.

Code generation is platform dependent, for information on building the software please refer to the release notes of APIS for the target platform and the APIS Programmer's Manual.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

4. FUNCTIONAL DESCRIPTION

4.1. INTRODUCING CAN

The Controller Area Network (CAN) is a data communication network designed to fit distributed real-time control applications. It was originally developed and applied by the automotive industry to solve the cabling problem inside vehicles.

The main applications for CAN-based networks in an industrial environment are:
- as a field bus in industrial plants.
- as an "in-machine" local bus in large tooling equipment, etc.

CAN is a serial communications protocol, originally specified by Bosch. The protocol handles multiple processors. Each of these processors can send and receive priority messages. CAN nodes request data by sending a Remote Frame message; the requested data is sent via a Data Frame with the same message ID. Messages are acknowledged by sending an ACK message.
Transmitted messages pass to all nodes on the bus (multi-cast), all of which take the message frame. The received messages are then filtered based on the message IDs.
CAN has only four message types: Data Frame, Remote Frame (request data), Error Frame (signals a local node error), and an Overload Frame (extend delay before next frame). As CAN nodes place requests, the highest priority one take the bus (priority is based on recessive vs dominant bits, where the dominant bits override the recessive bits).

4.2. CAN CHARACTERISTICS

- Multi master - multiple masters
- Priority-based transmissions: on message ID by node hardware
- All messages are received by all nodes; ID is the priority
- Message filtering: nodes filter messages using IDs before use
- 11- or 29-bit message ID
- Fixed message forms: variable length, to 8 bytes of data
- Built-in error detection and recovery
- Guaranteed latency for high-priority messages
- Low-power and radiation-noise implementations
- High reliability: CRC, message frame check, transmission monitoring
- Bit transmission to 1 Mbps
- Accommodates 100m between nodes (40m at 1 Mbps)

4.3. BLOCK DIAGRAM

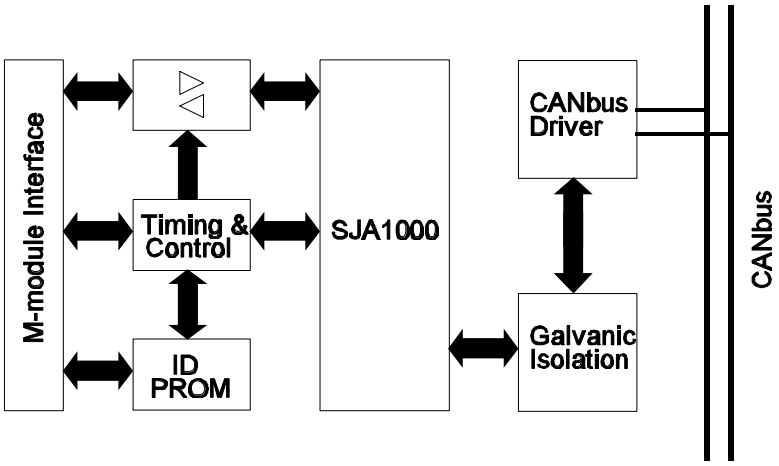

Figure 5 Example picture

4.4. M-MODULE INTERFACE

The M-module interface databus is 8 bits wide and supports low-byte transfers (d0..d7). The local bus interface to the CAN controller is based on the Intel principals with a multiplexed address/data bus.

Chip select signals, interrupt lines, enable lines and signals needed for identification readout are derived from the M-module interface by PLD's (U3 and U4).

A 16MHz crystal (default) or the M-module bus clock provides the clock signal.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

The addressable registers of the M360 are displayed in the following table.
Offset is the value which must be added to the (even) base address of the module.

Offset	Stand-alone CAN controller control 1
0x01	Control register
0x03	Command register
0x05	Status register
0x07	Interrupt register
0x09	Acceptance code register
0x0b	Acceptance mask register
0x0d	Bus timing 0 register
0x0f	Bus timing 1 register
0x11	Output control register
0x13	Test register
Offset	Transmit buffer
0x15	Identifier (10 to 3)
0x17	Identifier (2 to 0), RTR and data length code
0x19	Data byte 1
0x1b	Data byte 2
...	Data byte ...
0x27	Data byte 8
Offset	Receive buffer
0x29	Identifier (10 to 3)
0x2b	Identifier (2 to 0), RTR and data length code
0x2d	Data byte 1
0x2f	Data byte 2
...	Data byte ...
0x3b	Data byte 8
Offset	Stand-alone CAN controller control 2
0x3d	Clock divider
Offset	M360 M-module control
0x40	LEDs register
0x80	Reset register
0xff	Identification control

4.5. READING AND WRITING DATA

Reading or writing to the SJA1000 registers is done by accessing one of the registers from the table above. For a detailed description of the registers, please refer to the specification of the SJA1000 CAN controller.

4.6. STATUS LEDS

The two status LEDs at the M360 front can be turned off and on by writing to the front LED register (offset 0x40).
The GREEN LED is on when a '1' is written to bit 0 of the register.
The RED LED is on when a '1' is written to bit 1 of the register.

4.7. RESET REGISTER

After power up, the CAN controller is kept in reset state. Writing 0x01 to the reset register (offset 0x80) will release the reset.
Writing the sequence 0x00 0x01 will reset the hardware CAN controller.

4.8. INTERRUPT GENERATION

The M360 is capable of generating interrupts of type A, software-end-of-interrupt. Because the M360 is not capable of delivering an interrupt vector, this must be handled by the carrier board.

The interrupt register (offset 0x07) allows the identification of an interrupt source. When one or more bits of this register are set, the M360 will generate an interrupt to the host. The interrupt service routine must acknowledge the interrupt request by reading this register.

Interrupts can have several sources:
- **!** Receiver interrupt
- **!** Transmit interrupt
- **!** Error interrupt
- **!** Overrun interrupt
- **!** Wake-up interrupt (can not be enabled)

For a detailed description, please refer to the specification of the SJA1000 CAN controller.

4.9. MODULE IDENTIFICATION

With revision 2.1 of the M-module specification a new feature is added, called 'Module Identification'. The idea behind the feature is that a universal piece of software should be able to determine what module (and revision etc.) Is on a specific location. In order to do this an EEPROM type 93c06 (16x16 bits) is implemented.

The identification EEPROM contains the following information:

Word 0	Identification
Word 1	Module code
Word 2	Revision code
Word 3	Property of the module
Word 4	No intermodule port
Word 5...7	Reserved
Word 8...15	Dependent of the manufacturer

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

4.10. CANBUS INTERFACES

The M360 M-module contains a PCA82C250T CAN bus transceiver. This is the interface between the CAN controller and the physical bus. It is fully compatible with the ISO/DIS 11898 standard and primarily intended for high speed applications up to 1 Mbit/s.

The M360 M-module also contains a SN75176B differential bus transceiver. This meets the EIA Standards RS-422-A and RS-485 and CCITT Recommendations V.11 and X.27.

Refer to chapter ? on how to select one of the transceivers.

4.11. POWER SUPPLY

The power voltage +5 Volt is supplied through the M-module interface.

This page contains no essential data.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

5. SOFTWARE

This chapter describes the example software that is available for the M360 CANbus M-module. The example software is available in ANSI-C source code and consists mainly of an M360 function library which provides functions for easy access to the M360 and a demo program which illustrates the usage of the software library.
The M360 library functions are APIS based, physical accesses and interrupt support are handled by APIS, AcQ Platform Independent Interface Software. The next section contains general information on APIS, for detailed information please refer to the APIS Programmer's Manual.

5.1. APIS SUPPORT

AcQ produces and supports a large number of standard M-modules varying from networking and process I/O to motion control applications. Physically, the M-modules are supported by a large number of hardware platforms: VMEbus, PCI, CompactPCI as well as a wide variety of operating systems: OS-9, Windows NT, Linux etc.
APIS offers a way to program platform independent applications, example- and test software for controlling hardware. Application software written for APIS only needs re-compiling for a particular platform and is operational with little effort (provided that the application is operating system independent).

5.1.1. CONCEPT

Hardware accesses to registers and memory are handled by APIS. Some minor operating system dependent functions frequently used in hardware related software, such as interrupt handling and a delay function are also provided by APIS.
APIS platform support consists of an Application Programming Interface in the form of definition files coded in ANSI-C and platform dependent modules e.g. source files, libraries and/or drivers.
In the most simple outline, a platform dependent APIS module consist of nothing more then macro definitions in which APIS calls are substituted by direct hardware accesses. But in most cases an APIS module will consist of a library with interface routines and in some implementations a device driver is needed for interaction with the operating system.

5.1.2. API

The Application Programming Interface for APIS is implemented in two ANSI-C coded definition files: *apis.h* which contains general definitions and *platform_apis.h* which contains platform specific definitions and references to the APIS function calls.
The application source file must include the APIS header file *apis.h*. Porting of the application to a platform, consists of re-compiling the source code with a defined pre-processor macro for selection of the used platform. The APIS header file contains generic APIS definitions and includes a platform specific header file according to the platform selection macro.
API calls are translated to the platform specific calls in the APIS header file and the platform specific definition file *platform_apis.h* (*platform* is a name that identifies a hardware and operating system combination, e.g. i4000os9).
The macro PLATFORM must be defined, either via a pre-processor definition provided at compile time or via a macro-definition in the application source.

5.1.3. CODE GENERATION

APIS based example software is available in ANSI-C source code. Source code files must be compiled with the pre-processor definition PLATFORM set to a valid value, conforming the target platform. Building of the example software for the M360 is platform dependent, for details refer to the release notes of the APIS support package of the target platform and the APIS Programmer's Manual.
Examples of APIS supported platforms are i4000os9, i2000dos, i3000win etc.

5.2. TYPE DEFINITIONS AND STRUCTURES

The table below contains a list and description of all types and structures used in the M360 example software (standard ANSI-C types are not listed).

Name	Type	Description
INT8	char	8-bit signed data
UINT8	unsigned char	8-bit unsigned data
INT16	short	16-bit signed data
UINT16	unsigned short	16-bit unsigned data
INT32	long	32-bit signed data
UINT32	unsigned long	32-bit unsigned data
APIS_PATH	unsigned long	APIS physical path ID
M360_HANDLE	void*	M360 handle of a structure of information

CAN_MSG	struct {
	UINT16 id;
	UINT8 type;
	UINT8 length;
	UINT8 data[8];
	}
		CAN message contains:
		CAN identification
		Message type
		Number of valid data bytes
		Data bytes

ACC_FLT	struct {
	UINT16 mask;
	UINT16 code;
	}
		Acceptance filter contains:
		Mask
		Code

ERR_CNT	struct {
	UINT16 overflow;
	UINT16 overrun;
	UINT16 busOff;
	UINT16 txRx;
	}
		Error counters type contains:
		Overflow error counter
		Overrun error counter
		Bus off error counter
		Transmit/ receive error counter

MOD_ID	struct {
	UINT16 modNumber;
	UINT8 majorHwRev;
	UINT8 minorHwRev;
	UINT8 majorSwRev;
	UINT8 minorSwRev;
	}
		Module identification contains:
		Module number
		Major hardware revision number
		Minor hardware revision number
		Major software revision number
		Minor software revision number

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

Name	Type	Description

IOC_PRM	union {
	UINT16 baudrate;
	ACC_FLT accFlt;
	ERR_CNT errCnt;
	MOD_ID modId;
	UINT8 irqFlags;
	UINT16 parm[8];
	}
		IO-Control parameters type contains:
		Baudrate
		Acceptance filter
		Error counters
		Module identification
		Interrupt flags
		Raw parameters

5.3. THE M360 LIBRARY

The library is based on APIS, and contains 6 functions. These functions are described in the next section. Below the main aspects of the library will be discussed.

5.3.1. TRANSMIT AND RECEIVE BUFFERS

The library contains a transmit and a receive buffer. The buffers are based on a FIFO principle and each one has 32 places to store CAN messages.

When the M360 CANbus M-module receives a message, it will be stored in the receive buffer. If the receive buffer is full, the oldest message will be removed and the newest will be stored in the buffer. The oldest message of the buffer will be read with the read function of the library.

The write function examines the M360 CANbus M-module. If the module is not busy, the message will be transmitted immediately. Otherwise the message will be stored in the transmit buffer. When the buffer is full the write function will NOT store the message in the transmit buffer and returns an appropriated error code. When the M360 CANbus M-module is not busy anymore, the library is taking care of the transmitting of all messages in the transmit buffer. The oldest message of the buffer will be transmitted first.

5.3.2. ACCEPTANCE FILTER

The M360 CANbus M-module is equipped with a acceptance filter, which allows an automatic check of the identifier and data bytes. Using these effective filtering methods, messages or a group of messages not valid for a certain node can be prevented from being stored in the receive buffer. This way it is possible to reduce the processing load of the host controller.

The filter is controlled by the acceptance code and mask. The received identifier is compared bitwise with the acceptance code. The acceptance mask defines the bit positions, which are relevant for the comparison (0 = relevant, 1 = not relevant). To accept a message all relevant receive identifier bits have to match the respective bits of the acceptance code.

Note: Only the 8 most significant bits of the identifier of the CAN message are compared to the values of the acceptance code and mask.

Example:

	MSB									LSB
The acceptance code	0	1	1	1	0	0	1	0	0	1	1
The acceptance mask	0	0	1	1	1	0	0	0	1	1	0
Messages with the following 11-bit identifiers are accepted (x = don't care)	0	1	x	x	x	0	1	0	x	x	x
	ID.10										ID.0

5.3.3. ERROR COUNTERS

The library contains 4 error counters and each one will be increased when a corresponding error occurs on the M360 CANbus M-module.

The following errors will increase the error counters:

- **!** Receive buffer overflow.
  This error occurs when the receive buffer is full and a new received message must be stored.
- **!** Hardware buffer overrun.
  This error occurs when the hardware receive buffer is full and a new message is received. The buffer will be overwritten with the newest message.
- **!** Hardware transmit / receive error.
  The counter will be increased when M360 CANbus M-module detects errors on the CANbus.
- **!** Bus-off error.
  A bus-off error occurs when the M360 CANbus M-module has detected to many errors on the CANbus. The hardware of the M360 CANbus M-module will reset itself.

5.3.4. ERROR CODES

If the execution of a function is successful the function returns zero, if not the function returns an APIS error code. APIS error codes are 16-bit wide and are referred to with a symbolic name: APIS_Exxxxxxxx, where xxxxxxxx is a short description of 8 characters max.

Error code	Code	Description
APIS_NOERR	0x0000	no error
APIS_ENOTSUP	0x0001	not supported function
APIS_EPARAM	0x0010	bad parameter
APIS_EPARMOR	0x0011	parameter out of range
APIS_EPERMIT	0x0012	no permission
APIS_EWIDTH	0x0013	invalid data width
APIS_EGOS	0x0014	general operating system error
APIS_EINVREQ	0x0015	invalid request
APIS_ENOMEM	0x0016	no memory available
APIS_EMODERR	0x0017	APIS support module not found
APIS_ENOIRQH	0x0018	no interrupt handler installed
APIS_EINVPATH	0x0019	invalid path ID
APIS_ESIG	0x001a	signal error
APIS_EINVMOD	0x001b	invalid module
APIS_EINVDRV	0x001c	invalid driver
APIS_EOPENDEV	0x001d	error opening device
APIS_ELOCK	0x001e	device is already in use
APIS_EINCDEV	0x001f	incorrect device
APIS_EPCIERR	0x0020	PCI error
APIS_EINVHND	0x0021	invalid handle
APIS_EINVOFF	0x0022	invalid offset
APIS_EINTRPT	0x0023	interrupt error
APIS_ENIRQIU	0x0028	interrupt in use
APIS_ERESET	0x0201	module will not reset
APIS_EFIFOEMP	0x0202	FIFO buffer is empty
APIS_EFIFOFULL	0x0203	FIFO buffer is full
APIS_EFIFOFLOW	0x0204	FIFO buffer has an overflow

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

5.4. FUNCTION REFERENCE

m360_open () — Open a path

Syntax:

```
int m360_open ( UINT32 apisPath, M360_HANDLE *handle,
                ACC_FLT *pAccFlt, UINT16 baudrate,
                UINT16 vector, UINT16 level )
```

Description: A physical path will be opened for the requested path. The path is system dependent and can be hardware address, port ID or an index number of some kind. The interrupt service routine will be installed. This routine also initializes the M360 CANbus M-module. It initializes the receive buffer and transmit buffer, and resets the error counters.

Arguments:

UINT32 path
: The definition of the path ID is platform dependent. A path ID of zero selects the platform default ID.

M360_HANDLE *handle
: Handle pointer to a structure with information specific for a M360 CANbus M-module.

ACC_FLT *pAccFlt
: The 8 most significant bits of the identifier of the CAN message are compared to these values. Thus always groups of eight identifiers can be defined to be accepted for any node. At the bit positions containing a '1' in the Acceptance mask value, any value is allowed in the composition of the identifier. The other bit positions must be equal to the Acceptance code value.

UINT16 baudrate
: The baud rate of the CAN bus. This can be 10k, 20k, 50k, 125k, 250k, 500k, 800k or 1Mbit/s.

UINT16 vector
: Interrupt vector.

UINT16 level
: Interrupt level.

Returns:

APIS_NOERR
: The function executed successful.

APIS_Exxxxxxxx
: There was an error during the execution of this function.

Example:

```
int idx, result;
UINT16 vector = 0, level = 0;
APIS_PATH path;
M360_HANDLE handle;
ACC_FLT accFlt = {0xff, 0xff};
UINT16 baudrate = CAN_1000K;

result = m360_open (path, &handle, &accFlt, baudrate,
                          vector, level);
if (result != APIS_NOERR)
{
      /* Do something with the error code. */
}
```

m360_close() — Close a path

Syntax: `int m360_close ( M360_HANDLE handle )`

Description: A previously opened hardware path is closed. This routine reinitializes the M360 CANbus M-module. It removes the receive buffer and transmit buffer and resets the error counters. The interrupts will be disabled and the interrupt service routine will be removed.

Arguments: M360_HANDLE *handle
Handle of a structure with information specific for a M360 CANbus M-module.

Returns: APIS_NOERR
The function executed successful.
APIS_Exxxxxxxx
There was an error during the execution of this function.

Example:

```
int result;
M360_HANDLE handle;

result = m360_close ( handle );
if (result != APIS_NOERR)
{
    /* Do something with the error code. */
}
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

m360_read ()

Read a CAN message

Syntax: `int m360_read ( M360_HANDLE handle, CAN_MSG *pMsg )`

Description: This routine reads a received CAN message from the receive buffer. When there are no received CAN messages, the routine will return an "Buffer empty" error-code. When the receive buffer has an overflow, the routine will return an "Buffer overflow" error-code.

Arguments:

M360_HANDLE handle
: Handle of a structure with information specific for a M360 CANbus M-module.

CAN_MSG *pMsg
: Pointer to a received CAN message. This message includes the 11-bit identifier, frame type, data length and the data (max. 8 bytes).

Returns:

APIS_NOERR
: The function executed successful.

APIS_Exxxxxxxx
: There was an error during the execution of this function.

Example:

```
int idx, result;
M360_HANDLE handle;
CAN_MSG msg;

result = m360_read (handle, &msg);
if (result == APIS_NOERR)
{
    printf ("Received:    Id: 0x%04x ", msg.id);
    if (msg.type == DATA)
    {
        printf ("Type: data    ");
        if (msg.length > 8)
        {
            msg.length = 8;
        }
        printf ("Data: ");
        for (idx = 0; idx < msg.length; idx++)
        {
            printf ("0x%02x ", msg.data[idx]);
        }
    }
    else
    {
        printf ("Type: request ");
    }
    printf ("\n");
}
else
{
    /* Do something with the error code. */
}
```

m306_write () — Write a CAN message

Syntax: `int m360_write ( M360_HANDLE handle, CAN_MSG *pMsg )`

Description: This routine writes a CAN message to the M360 CANbus M-module. When the M360 is busy, the message will be written to the transmit buffer. This message will be transmitted when the M360 is not busy. When the transmit buffer is full, the routine will return an "Buffer full" error-code.

Arguments:

M360_HANDLE handle
: Handle of a structure with information specific for a M360 CANbus M-module.

CAN_MSG *pMsg
: Pointer to a CAN message which must be transmitted. This message includes the 11-bit identifier, frame type, data length and the data (max. 8 bytes).

Returns:

APIS_NOERR
: The function executed successful.

APIS_Exxxxxxxx
: There was an error during the execution of this function.

Example:

```
int idx, result;
M360_HANDLE handle;
CAN_MSG msg;

msg.id = 0x200;
msg.type = DATA;
msg.length = 8;

for (idx = 0; idx < msg.length; idx++)
{
    msg.data[idx] = 0x100 + idx;
}

result = m360_write (handle, &msg);
if (result != APIS_NOERR)
{
    /* Do something with the error code. */
}
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

m360_ioctrl () — Perform control commands

Syntax:

```
int m360_ioctrl ( M360_HANDLE handle, UINT16 command,
                  IOC_PRM *pParm )
```

Description: This routine performs control commands with the information stored in the control parameters. Or gives information back through these control parameters. The following control commands will be available:

Command	Parameters	Description
SET_BAUDRATE	pParm->baudrate	This command sets the baud rate of the CAN bus. This can be 10k, 20k, 50k, 125k, 250k, 500k, 800k or 1Mbit/s.
GET_BAUDRATE	pParm->baudrate	This command returns the established baud rate of the CAN bus.
SET_ACCFLT	pParm->accFlt.code pParm->accFlt.mask	This command sets the acceptance code and mask of the M360 CAN bus M-module.
GET_ACCFLT	pParm->accFlt.code pParm->accFlt.mask	This command returns the established acceptance code and mask.
GET_ERRCNT	pParm->errCnt.overflow pParm->errCnt.overrun pParm->errCnt.busOff pParm->errCnt.txRx	This command returns the error counters of the M360 CAN bus M-module.
RESET_IRQF	pParm->irqFlags	This command resets the interrupt flags of the M360 CAN bus M-module.
GET_IRQF	pParm->irqFlags	This command returns the interrupt flags of the M360 CAN bus M-module.
GET_MODID	pParm->modId.modNumber pParm->modId.majorHwRev pParm->modId.minorHwRev pParm->modId.majorSwRev pParm->modId.minorSwRev	This command returns the module identification and the hardware- and software-revision.
FLUSH_BUF	-	This command clears the receive and transmit buffers.

Arguments: M360_HANDLE handle
Handle of a structure with information specific for a M360 CANbus M-module.
UINT16 command
This control command will be performed.
IOC_PRM *pParm
The information parameters of the control command.

Returns: APIS_NOERR
The function executed successful.
APIS_Exxxxxxxx
There was an error during the execution of this function.

Example:

```
int result;
M360_HANDLE handle;
IOC_PRM parm;

result = m360_ioctrl ( handle, GET_MODID, &parm);
if (result == APIS_NOERR)
{
    printf ("M-module          M%d\n", parm.modId.modNumber);
    printf ("Hardware revision %d.%d\n",
             parm.modId.majorHwRev, parm.modId.minorHwRev);
    printf ("Software revision %d.%d\n",
             parm.modId.majorSwRev, parm.modId.minorSwRev);
}
else
{
    /* Do something with the error code. */
}

parm.baudrate = CAN_20K;
result = m360_ioctrl ( handle, SET_BAUDRATE, &parm);
if (result != APIS_NOERR)
{
    /* Do something with the error code. */
}
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

m360_waitforirq ()

Wait for interrupt

Syntax: `int m360_waitforirq ()`

Description: Suspend current process. When a signal is received that is sent by an APIS interrupt service routine, this function returns with APIS_NOERR as result code. If a signal is received not caused by APIS (e.g. keyboard interrupt), the function returns with APIS_ESIG as result code. The interrupts received before m360_waitforirq is called are not missed.

Arguments: .

No parameters.

Returns:

APIS_NOERR
: The function executed successful.

APIS_ESIG
: There was an interrupt received, but not caused by APIS.

APIS_Exxxxxxxx
: There was an error during the execution of this function.

Example:

```
int result;
M360_HANDLE handle;
CAN_MSG msg;
IOC_PRM parm;

printf ("Waiting...\n");
do
{
    result = m360_waitforirq();
    m360_ioctrl ( handle, GET_IRQF, &parm);
} while ((result == APIS_NOERR) &&
         ((parm.irqFlags & M360_RX_IRQ) != M360_RX_IRQ));

if (result == APIS_ESIG)
{
    /*
     * A signal is received, but not caused by APIS
     * (e.g. keyboard interrupt).
     */
}
```

```
else if ((parm.irqFlags & M360_RX_IRQ) == M360_RX_IRQ)
{
    /* The M360 M-module has received a message. */
    parm.irqFlags = M360_RX_IRQ;
    m360_ioctrl ( handle, RESET_IRQF, &parm);

    result = m360_read (handle, &msg);
    if (result == APIS_NOERR)
    {
        /* Do something with the message. */
        printf ("Received:    Id: 0x%04x ", msg.id);
        if (msg.type == DATA)
        {
            printf ("Type: data    ");

            if (msg.length > 8)
            {
                msg.length = 8;
            }
            printf ("Data: ");
            for (idx = 0; idx < msg.length; idx++)
            {
                printf ("0x%02x ", msg.data[idx]);
            }
        }
        else
        {
            printf ("Type: request ");
        }
        printf ("\n");
    }
    else
    {
        /* Do something with the error code. */
    }
}
else
{
    /* Do something with the error code. */
}
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

5.5. SOFTWARE DISTRIBUTION

This section gives an overview of the software distribution.

File	Description
m360rel.txt	Release notes
m360lib.c	APIS based ANSI-C M360 software library
m360lib.h	Definitions for M360 software library
M360loc.h	Local definitions for M360 software library
m360defs.h	Definitions of M360 M-module
fifo.c	APIS based ANSI-C FIFO buffer
fifo.h	Definitions for FIFO buffer
m360main.c	Example application source code
modideep.c	APIS based ANSI-C M-module ID EEPROM software library
ideeprom.h	M-module ID EEPROM definitions

The software is distributed on a 3.5" 1,44MB PC-HD format floppy disk. Other media are available on request.

M360 example software is APIS based, therefore APIS support for the target platform is required for code generation.

The following figure is an example of the M360 software integrated in the APIS environment with as target platform the i4000/OS-9.

```
+---PROJECT                          AcQ's distribution
     +---APIS                        APIS basis distribution
     |    +---DOC                    APIS documentation
     |    +---SOFTWARE
     |         | readme.txt          Distribution overview
     |         +---COMMON
     |         |    +---DEFS
     |         |    |    apis.h      General definitions
     |         |    +---OS9TRAP
     |         |    |   +--CMDS
     |         |    |       apistrap  OS-9 trap handler
     |         |    +---....
     |         +---I4000OS9          i4000/OS-9 support
     |         |    relnotes.txt     Release notes / version info
     |         |    apis_i4000os9.h  Platform specific definitions
     |         |    apis_i4000os9.c  Platform support library
     |         +---....
     +---M360                        M360 distribution
     |    +---DOC                    M360 documentation
     |    +---SOFTWARE
     |         +---LIB               M360 support libraries
     |         |    m360rel.txt      Release notes/version info
     |         |    m360lib.c        APIS based ANSI-C library
     |         |    m360lib.h        Definitions for library
     |         |    m360loc.h        Local definitions for library
     |         |    m360defs.h       Definitions of M360 M-module
     |         |    fifo.c           APIS based ANSI-C FIFO buffer
     |         |    fifo.h           Definitions for FIFO buffer
     |         +---EXAMPLE
     |              m360main.c       Example source code
     +---MMODID
          +---SOFTWARE
               +---LIB
                    modideep.c      M-module ID EEPROM library
                    ideeprom.h      M-module ID definitions
```

Code generation is platform dependent, for information on building the software please refer to the release notes of the target platform and the APIS Programmer's Manual.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

6. ANNEX

6.1. BIBLIOGRAPHY

Specification for M-module interface and physical dimensions
M-module specification manual, April 1996, MUMM.
Simon-Schöffel-Strasse 21, D-90427 Nürnberg, Germany.

APIS Programmer's Manual
AcQuisition Technology
P.O. Box 627, 5340 AP Oss, The Netherlands.

Data Sheet of the SJA1000
Data sheet SJA1000 Stand-alone CAN controller
Philips Electronics NV, Nov. 04 1997
P.O.Box 218, 5600 MD, Eindhoven, The Netherlands

Application note of the SJA1000
Application Note SJA1000 Stand-alone CAN controller, AN97076
Philips Electronics NV, Dec. 15 1997
P.O.Box 218, 5600 MD, Eindhoven, The Netherlands

6.2. COMPONENT IMAGE

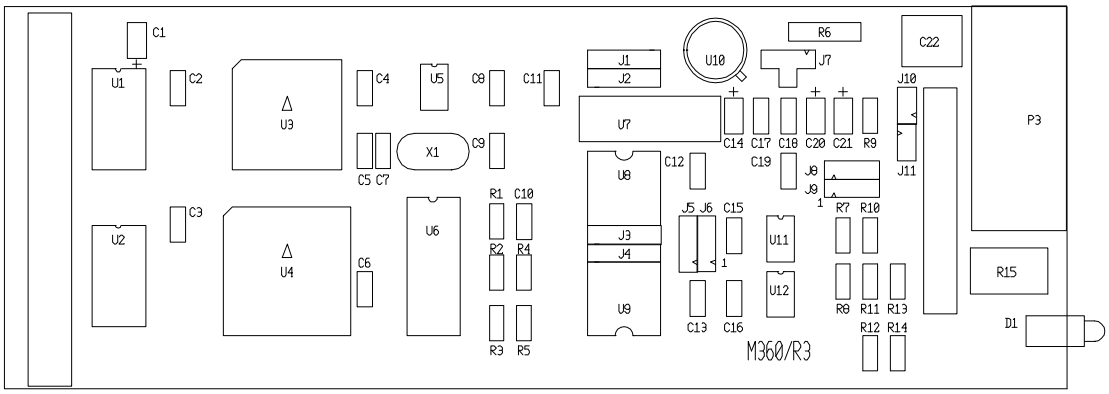

Figure 6 M360 Components position

6.3. TECHNICAL DATA

Slots on the base-board:
Requires one 16-bit M-module slot.

Interrupt:
Interrupt of type A, software-end-of-interrupt.

Connection:
To base-board via 40 pole M-module interface.
To peripheral on the front via 9 pole D-sub connector, or via 24 pole header connector.

Power supply:
+5VDC ±10%, typical 175mA (from M-module baseboard, galvanic isolation voltage supply from local DC/DC converter).

Temperature range:
Operating: 0..+60°C.
Storage : -20..+70°C.

Humidity:
Class F, non-condensing.

6.4. DOCUMENT HISTORY

! Revision 1.0
First release, this release replaces the Hardware Manual Revision 3.0 and Software Manual Revision 3.0.

6.5. EXAMPLE CODE

```
/*
 * File:            m360main.c
 * Revision:        1.0
 * Date:            11-01-00
 * Authors:         MQ
 * ------------------------------------------------------------------------
 * Demo application
 *
 * The file contains the main function of the demo application of
 * the M360 CANbus M-module.
 *
 * ------------------------------------------------------------------------
 * Copyright 2000 by AcQuisition Technology bv.
 * All Rights Reserved
 * Reproduced Under License
 *
 * This source code is the proprietary confidential property of
 * AcQuisition Technology bv., and is provided to the licensee
 * for documentation and educational purposes only. Reproduction,
 * publication, or any form of distribution to any party other than
 * the licensee is strictly prohibited.
 * ------------------------------------------------------------------------
 * Edition History
 *
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

```
 * ##.##  Data        Comments                                    By
 * -----  ----------  ------------------------------------------  ------
 * 00.01  01-12-1999  Initial version                             MQ
 * 01.00  11-01-2000  First release                               MQ
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include "../LIB/m360lib.h"

/* Forward declarations */
void usage( char *pname );
void printError ( int errorCode );

/*
 * Function:        main
 *
 * Description:     This is the main routine of the Demo application.
 *
 * Parameters:      int argc,
 *                  char *argv[]
 *                          Parameters throughput
 *
 * Return:          0
 *                          Always zero.
 *
 */
int main (int argc, char *argv[])
{
    int idx1, idx2;
    int readMsg = 0, writeMsg = 0;
    UINT32 base = 0;
    UINT32 pass = 1, passed = 0;
    int result;
    APIS_PATH path = 0;
    M360_HANDLE handle;
    ACC_FLT accFlt = {0xff, 0xff};
    UINT16 baudrate = CAN_1000K;
    CAN_MSG msg;
    IOC_PRM parm;
    int data[9];

    printf("\nM360 CANbus M-module test\n");
    printf("by AcQuisition Technology B.V. 2000\n\n");

    for (idx1 = 1; idx1 < argc; idx1++)
    {
        if (argv[idx1][0] == '-')
        {
            for (idx2 = 1; argv[idx1][idx2]; idx2++)
            {
                switch (tolower(argv[idx1][idx2]))
                {
                    case 'p':
                        /* Set up a path to the hardware */
                        if (argv[idx1][++idx2] == '=')
                        {
                            idx2++;
```

```
                            sscanf(argv[idx1]+idx2, "%lx", &base);
                            if (base != 0) {
                                path = (APIS_PATH)base;
                            }
                            while(argv[idx1][idx2])
                            {
                                idx2++;
                            }
                            idx2--;
                        }
                        break;
                    case 'r':
                        /* We want to receive x messages */
                        readMsg = 1;
                        writeMsg = 0;
                        if (argv[idx1][++idx2] == '=')
                        {
                            idx2++;
                            sscanf(argv[idx1]+idx2, "%ld", &pass);
                            while(argv[idx1][idx2])
                            {
                                idx2++;
                            }
                            idx2--;
                        }
                        break;
                    case 'w':
                        /* We want to transmit x messages */
                        writeMsg = 1;
                        readMsg = 0;
                        if (argv[idx1][++idx2] == '=')
                        {
                            idx2++;
                            sscanf(argv[idx1]+idx2, "%ld", &pass);
                            while(argv[idx1][idx2])
                            {
                                idx2++;
                            }
                            idx2--;
                        }
                        break;
                    case 'b':
                        /* Change the baudrate */
                        if (argv[idx1][++idx2] == '=')
                        {
                            idx2++;
                            sscanf(argv[idx1]+idx2, "%d", &baudrate);
                            while(argv[idx1][idx2])
                            {
                                idx2++;
                            }
                            idx2--;
                        }
                        break;
                    default:
                        usage(argv[0]);
                }
            }
        }
    }
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

```
    if (readMsg == writeMsg)
    {
        /* only read OR write */
        usage(argv[0]);
    }

    /* Open a path the the M360 CAN bus M-module */
    result = m360_open (path, &handle, &accFlt, baudrate, 0, 0);
    if (result != APIS_NOERR) {
        printError (result);
        return 0;
    }

    /* Get the module number and its revision number */
    m360_ioctrl ( handle, GET_MODID, &parm);
    printf ("M-module          M%d\n", parm.modId.modNumber);
    printf ("Hardware revision %d.%d\n", parm.modId.majorHwRev,
             parm.modId.minorHwRev);
    printf ("Software revision %d.%d\n", parm.modId.majorSwRev,
             parm.modId.minorSwRev);

    /* Transmit x CAN messages */
    if (writeMsg == 1)
    {
        do
        {
            idx1 = 0;
            printf ("Give CAN id: ");
            scanf ("%d", &data[idx1]);
            msg.id = (UINT16) data[0];
            msg.type = DATA;

            printf ("Give message length (max 8): ");
            scanf ("%d", &data[idx1]);
            if (data[idx1] > 8)
            {
                msg.length = 8;
            }
            else
            {
                msg.length = (UINT8)data[idx1];
            }

            for (idx1 = 0; idx1 < msg.length; idx1++)
            {
                printf ("Give data[%d]: ", idx1);
                scanf ("%d", &data[idx1]);
                msg.data[idx1] = (UINT8)(data[idx1] & 0x00ff);
            }

            printf ("Transmitting the CAN message.\n");
            result = m360_write (handle, &msg);
            printError (result);

            passed++;
        } while (passed < pass);
    }

    /* Receive x CAN messages */
    if (readMsg == 1)
    {
```

```
        do
        {
            printf ("Waiting...\n");
            do
            {
                /* Wait for an interrupt */
                result = m360_waitforirq();
                m360_ioctrl ( handle, GET_IRQF, &parm);
                /* Is it ours? */
            } while ((result == APIS_NOERR) &&
                     ((parm.irqFlags & M360_RX_IRQ) != M360_RX_IRQ));

            if (result == APIS_ESIG)
            {
                /*
                 * A signal is received, but not caused by APIS
                 * (e.g. keyboard interrupt).
                 */
                passed = pass;
            }
            else if ((parm.irqFlags & M360_RX_IRQ) == M360_RX_IRQ)
            {
                /* The M360 M-module has received a message. */
                /* Reset the receive interrupt flag */
                parm.irqFlags = M360_RX_IRQ;
                m360_ioctrl ( handle, RESET_IRQF, &parm);

                /* Read the message from the M360 CAN bus M-module */
                result = m360_read (handle, &msg);
                if (result == APIS_NOERR)
                {
                    /* Do something with the message. */
                    printf ("Id: 0x%04x ", msg.id);
                    if (msg.type == DATA)
                    {
                        printf ("Type: data    ");
                        if (msg.length > 8)
                        {
                            msg.length = 8;
                        }
                        printf ("Data: ", msg.data[idx1]);
                        for (idx1 = 0; idx1 < msg.length; idx1++)
                        {
                            printf ("0x%02x ", msg.data[idx1]);
                        }
                    }
                    else
                    {
                        printf ("Type: request ");
                    }
                    printf ("\n");
                }
                else
                {
                    /* The read function goes wrong */
                    /* Do something with the error code. */
                    printError (result);
                }
                passed++;
            }
            else
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

```
            {
                /* The waitforirq function goes wrong */
                /* Do something with the error code. */
                printError (result);
            }
        } while (passed < pass);
    }

    /* Close the path to the M360 CAN bus M-module */
    m360_close (handle);
    printf ("%s is stopped\n", argv[0]);

    return 0;
}

/*
 * Function:        usage
 *
 * Description:     Displays the usage of this application and closes it.
 *
 * Parameters:      char *pName
 *                          Application name.
 *
 * Return:          0
 *                          Always zero.
 *
 */
void usage (char *pName)
{
    printf("Syntax: %s [<opts>]\n", pName);
    printf("Options:\n");
    printf("\t-p=<path>     module base path in hex (default = 0)\n");
    printf("\t-r=<pass>     display incoming CAN messages or\n");
    printf("\t-w=<pass>     send a CAN message\n");
    printf("\t-b=<baudrate> set the baudrate: 1 = 10 Kbits/s\n");
    printf("\t                                2 = 20 Kbits/s\n");
    printf("\t                                3 = 50 Kbits/s\n");
    printf("\t                                4 = 125 Kbits/s\n");
    printf("\t                                5 = 250 Kbits/s\n");
    printf("\t                                6 = 500 Kbits/s\n");
    printf("\t                                7 = 800 Kbits/s\n");
    printf("\t                                8 = 1 Mbits/s (default)\n");
    printf("\t-?            this help\n");
    exit(0);
}

/*
 * Function:        printError
 *
 * Description:     Displays the meaning of the error code.
 *
 * Parameters:      int errorCode
 *                          An error number.
 *
 * Return:          .
 *                          None.
 *
 */
void printError (int errorCode)
{
    switch (errorCode)
```

```
{
    case APIS_ENOTSUP:
        printf ("not supported function\n");
        break;
    case APIS_EPARAM:
        printf ("bad parameter\n");
        break;
    case APIS_EPARMOR:
        printf ("parameter out of range\n");
        break;
    case APIS_EPERMIT:
        printf ("no permission\n");
        break;
    case APIS_EWIDTH:
        printf ("invalid data width\n");
        break;
    case APIS_EGOS:
        printf ("general operating system error\n");
        break;
    case APIS_EINVREQ:
        printf ("invalid request\n");
        break;
    case APIS_ENOMEM:
        printf ("no memory available\n");
        break;
    case APIS_EMODERR:
        printf ("APIS support module not found\n");
        break;
    case APIS_ENOIRQH:
        printf ("no interrupt handler installed\n");
        break;
    case APIS_ENIRQIU:
        printf ("interrupt in use\n");
        break;
    case APIS_EINVPATH:
        printf ("invalid path ID\n");
        break;
    case APIS_ESIG:
        printf ("signal error\n");
        break;
    case APIS_EINVMOD:
        printf ("invalid module\n");
        break;
    case APIS_EINVDRV:
        printf ("invalid driver\n");
        break;
    case APIS_EOPENDEV:
        printf ("error opening device\n");
        break;
    case APIS_ELOCK:
        printf ("device is already in use\n");
        break;
    case APIS_EINCDEV:
        printf ("incorrect device\n");
        break;
    case APIS_EPCIERR:
        printf ("PCI error\n");
        break;
    case APIS_EINVHND:
        printf ("invalid handle\n");
        break;
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

```
        case APIS_EINVOFF:
            printf ("invalid offset\n");
            break;
        case APIS_EINTRPT:
            printf ("interrupt error\n");
            break;
        case APIS_EFIFOEMP:
            printf ("fifo buffer is empty\n");
            break;
        case APIS_EFIFOFULL:
            printf ("fifo buffer is full\n");
            break;
        case APIS_EFIFOFLOW:
            printf ("fifo buffer has an overflow\n");
            break;
    }
}

/* End of file */
```