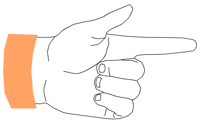
Controller **A**rea **N**etwork **P**rogramming **I**nterface **E**nvironment
Version 2.00

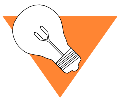
Users Guide

Document Conventions

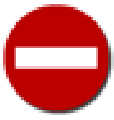
Icons on the border of the page are used to mark certain paragraphs in this document. The following icons are used:



Note
This icon designates a note relating to the surrounding text. It is recommended to read these sections.



Tip
This icon designates a helpful tip relating to the surrounding text.



Warning
This icon designates a warning relating to the surrounding text. Please 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 courier is used.`

MicroControl GmbH & Co. KG
Lindlaustrasse 2c
53844 Troisdorf
Germay

URL: http://canpie.sourceforge.net
eMail: canpie@microcontrol.net

1. Scope

The goal of this project is to define a "Standard" Application Programming Interface (API) for access to the CAN bus. The API provides functionality for ISO/OSI Layer-2 (Data Link Layer). It is not the intention of CANpie to incorporate higher layer functionality (e.g. CANopen, J1939, DeviceNet).

Whereever it is possible CANpie is independent from the used hardware and operating system. The function calls are unique for different kinds of hardware. Also CANpie provides a method to gather information about the features of the CAN hardware. This is especially important for an application designer, who wants to write the code only once.

The API is designed for embedded control applications as well as for PC interface boards.

1.1 References

/ISO11898-1/	ISO 11898-1, Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling
/ISO11898-2/	ISO 11898-2, Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit
/DS-301/	CANopen specification DS-301, version 4.1, CAN in Automation
/ATMEL01/	Datasheet ATMEL microcontroller AT89C51CC01, CAN controller section

1.2 Abbreviations

CAN	Controller area network
CAN-ID	CAN identifier
CRC	Cyclic redundancy check
LSB	Least significant bit/byte
MSB	Most significant bit/byte
OSI	Open systems interconnection
PLS	Physical layer signaling
PMA	Physical medium attachment
RTR	Remote transmission request

1

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

The CAN (Controller Area Network) protocol 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.

1.5 License

CANpie is **free software**; you can redistribute it and/or modify it under the terms of the **GNU Lesser General Public License** as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This code / library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

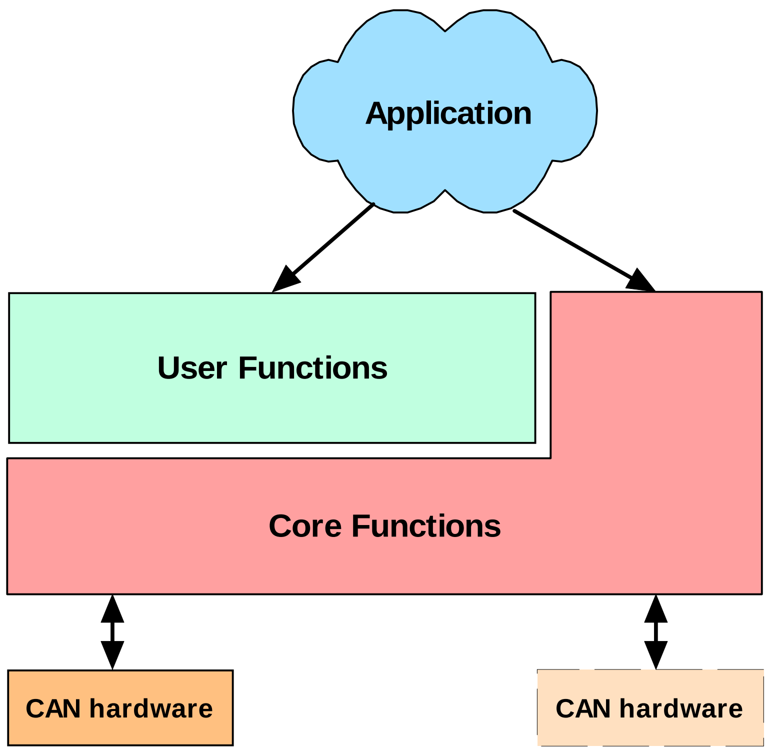
The license can be found as appendix to this manual.

1

2. Driver Principle

One of the ideas of CANpie is to keep it independent from the hardware. This is of course difficult to achieve, due to many different target platforms:
- CAN interface for embedded control
- CAN interface for PC (without local processor)
- CAN interface for PC (with local processor)

CANpie tries to meet this requirement by providing a two-level API, consisting of core functions and user functions.



CANpie Structure

User Functions

The user functions always call the core functions, they never access the hardware directly. That means the user functions do not have to be modified when implementing the CANpie on an existing hardware.

Core Functions

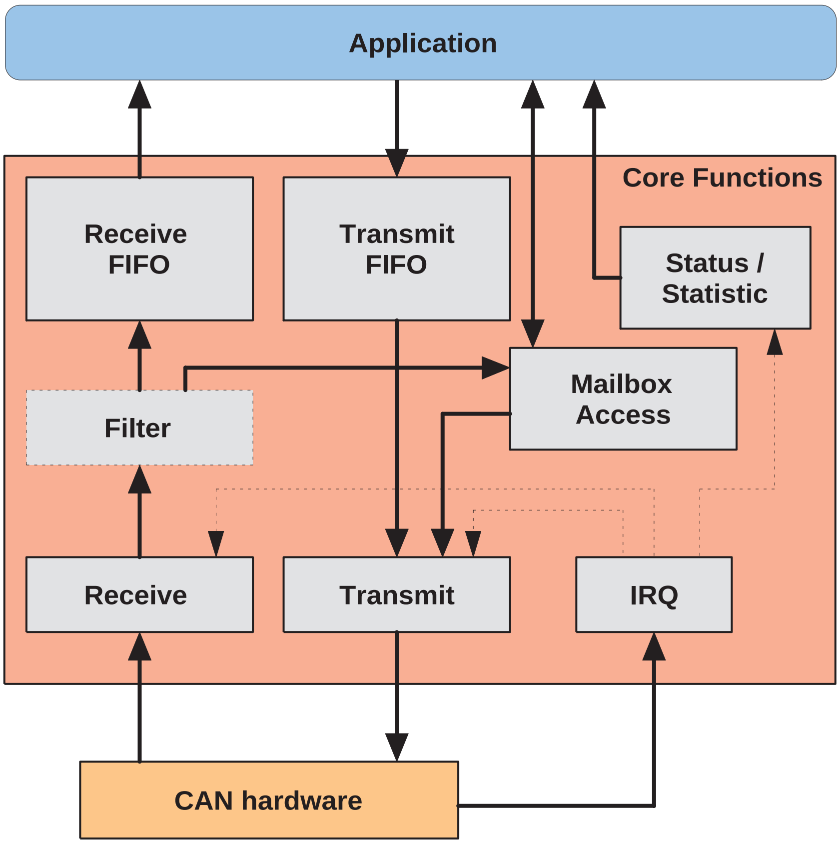
The core functions access the hardware directly, so an adaption is neccessary when implementing on a piece of hardware. Core functions may also be called by the application.

CANpie supports more than one CAN channel on the hardware. The actual number of CAN channels can be gathered via the Hardware Description Interface (refer to “Hardware Description Interface” on page 12).

2

2.1 Message Distribution

The message distribution is responsible for reading and writing CAN messages. The key component for message distribution is the Interrupt Handler. The Interrupt Handler is started by a hardware interrupt from the CAN controller. The Interrupt Handler has to determine the interrupt type (receive / transmit / status change).



Message Handler

Interrupt Handler

In case of a receive interrupt the handler uses the Receive Message routine to get the CAN message from the controller and put it into the Receive FIFO (First-In-First-Out). The Receive FIFO must be initialized by the application. If the Receive FIFO is full, no further messages will be queued and an error-signal will be submitted.

In case of a transmit interrupt the Transmit FIFO is checked. If there are messages in this queue, the Transmit Message routine will write the next waiting message to the CAN controller. If the Transmit FIFO is empty and the application puts a CAN message into the queue, the Transmit Message routine will be called automatically.

Callback Functions

The occurence of an interrupt may call a user defined handler function. Handler functions are possible for the following conditions:

- Receive interrupt
- Transmit interrupt
- Status change interrupt

2.2 Data types

Due to different implementations of data types in the world of C compilers, the following data types are used for CANpie API. The data types are defined in the header file "**compiler.h**".

Data Types

Data Type	Definition
_BIT	Boolean value, True or False
_U08	1 Byte value, value range 0 .. 2^8 - 1 (0 .. 255)
_S08	1 Byte value, value range -2^7 .. 2^7 - 1 (-128 .. 127)
_U16	2 Byte value, value range 0 .. 2^{16} - 1 (0 .. 65535)
_S16	2 Byte value, value range -2^{15} .. 2^{15} - 1
_U32	4 Byte value, value range 0 .. 2^{32} - 1
_S32	4 Byte value, value range -2^{31} .. 2^{31} - 1

Table 1: Data Type definitions

2.3 Naming Conventions

As mentioned in chapter 2, CANpie is divided in core and user functions. All functions, structures, defines and constants in CANpie have the prefix "**Cp**". The following table shows the used nomenclature:

CANpie	Prefix
Core functions	CpCore
User functions	CpUser
Message access functions	CpMsg
Structures	_TsCp
Constants / Defines	CP
Error Codes	CpErr

Table 2: Naming conventions

All constants, defines and error codes can be found in the header file "**canpie.h**".

2.4 Initialization Process

The CAN driver is initialized with the function **CpCoreDriverInit()**. This routine will setup the CAN controller, but not configure a certain bitrate nor switch the CAN controller to active operation. The following core functions must be called in that order:

- `CpCoreDriverInit()`
- `CpCoreBaudrate() / CpCoreBittiming()`
- `CpCoreCanMode()`

```
void MyCanInit(void)
{
   _TsCpPort  tsCanPortT;   // logical CAN port

   //-------------------------------------------------------
   // setup the CAN controller / open a physical CAN
   // port
   //
   CpCoreDriverInit(CP_CHANNEL_1, &tsCanPortT);

   //-------------------------------------------------------
   // setup 500 kBit/s
   //
   CpCoreBaudrate(&tsCanPortT, CP_BAUD_500K);

   //-------------------------------------------------------
   // start CAN operation
   //
   CpCoreCanMode(&tsCanPortT, CP_MODE_START);


   //.. now we are operational

}
```

Example 1: Initialization process of the CAN interface

The function **CpCoreDriverInit()** must be called before any other core function in order to have a valid handle / pointer to the open CAN interface.

3. API Overview

This chapter gives an overview of the CANpie API. It also discusses the used structures in detail.

3.1 Physical CAN interface

A target system may have more than one physical CAN interface. The physical CAN interfaces are numbered from 0 .. N-1 (N: total number of physical CAN interfaces on the target system). The header file canpie.h provides an enumeration for the physical CAN interface (the first CAN interface is **CP_CHANNEL_1**). A physical CAN interface is opened via the function **CpCoreDriverInit()**. The function will setup a pointer to the structure **_TsCpPort** for further operations. The elements of the structure **_TsCpPort** depend on the used target system and are defined in the header file cp_arch.h (which defines data types and structures for different architectures).

```
/*--------------------------------------------------------*/
/*!
** \struct  CpPortLinux_s   cp_arch.h
** \brief   Port structure for Linux
**
*/
struct CpPortLinux_s {

   /*!   logical CAN interface number,
   **    first index is 0, value -1 denotes not assigned
   */
   int      slLogIf;

   /*!   physical CAN interface number,
   **    first index is 0, value -1 denotes not assigned
   */
   int      slPhyIf;

   /*!   CAN message queue number,
   **    first index is 0, value -1 denotes not assigned
   */
   int      slQueue;
};

.....

typedef struct CpPortLinux_s     _TsCpPort;
```

Example 2: CAN port structure for a LINUX target

For an embedded application with only one physical CAN interface the parameter to the CAN port can be omitted. This reduces the code size and also increases execution speed. This option is configured via the symbol CP_SMALL_CODE during the compilation process.

3.2 Hardware Description Interface

The Hardware Description Interface provides a method to gather information about the CAN hardware and the functionality of the driver. For this purpose the following structure is defined:

```
struct CpHdi_s{
    _U16       uwVersionNumber;
    _U16       uwSupportFlags;
    _U16       uwControllerType;
    _U16       uwIRQNumber;
    _U16       uwBufferMax;
    _U16       uwRes;
    _U32       ulTimeStampRes;
    _U32       ulCanClock;
    _U32       ulBitrate;

};

typedef struct CpHdi_s   _TsCpHdi;
```

The hardware description structure is available for every physical CAN channel.

Support Flags

7	6	5	4	3	2	1	0
res.	User Data	Timestamp	Software ID-Filter	IRQ-Handler	FullCAN	Frametype (2.0A / 2.0B)	

Frametype

Bit 0 and Bit 1 of the structure member **uwSupportFlags** describe the frame support of the CAN controller. The following values are defined:
0: Standard Frame (11-bit identifier), 2.0A
1: Extended Frame (29-bit identifier), 2.0B passive
2: Extended Frame (29-bit identifier), 2.0B active

FullCAN

If the flag "FullCAN" is set to "1", the CAN controller has more than one receive buffer and one transmit buffer.

Interrupt Handler

If the flag "IRQHandler" is set to "1", the driver will use a hardware interrupt. If set to "0", no interrupt handler is implemented. This also means, that no callback functions can be used (polling).

Software ID-Filter

If the flag "Software ID-Filter" is set to "1", the driver has implemented the software ID filter for standard frames. If the member is set to "0", the software filter is not available.

Timestamp

If this flag is set to "1", the CAN driver will set a time stamp to all received messages. The time stamp has a resolution of 1 microsecond (Siehe “Time Stamp” auf Seite 15.).

User Data

If this flag is set to "1", the element **ulUserData** of the structure **CpCanMsg_s** is valid.

Controller Type A constant that identifies the used CAN controller chip. Possible values for this member are listed in the header file "**cp_cc.h**".

IRQNumber Defines the number of the interrupt which is used. If the flag "IRQHandler" is set to "0", the value of "IRQNumber" will be undefined.

VersionMajor Holds the major version number of the CANpie driver release.

VersionMinor Holds the minor version number of the CANpie driver release.

3.3 Structure of a CAN message

For transmission and reception of CAN messages a structure which holds all necessary informations is used (***CpCanMsg_s***). The structure is defined in the header file canpie.h and has the following data fields:

```
struct CpCanMsg_s{
    // identifier field (11/29 bit)
    union {
            _U16  uwStd;
            _U32  ulExt;
    } tuMsgId;

    // data field (8 bytes)
    union {
            _U08  aubByte[8];
            _U16  auwWord[4];
            _U32  aulLong[2];
    } tuMsgData;

    // Data length code
    _U08  ubMsgDLC

    // Extended frame / remote frame
    _U08  ubMsgCtrl

    #if CP_CAN_MSG_TIME == 1
    _TsCpTime  tsMsgTime;
    #endif

    #if CP_CAN_MSG_USER == 1
    _U32  ulMsgUser;
    #endif

};

// typedef for this structure:
typedef struct CpCanMsg_s   _TsCpCanMsg;


struct CpTime_s {
    _U32  ulSec1970;
    _U32  ulNanoSec;
}
```

Identifier

The identifier field (**union tuMsgId**) may have 11 bits for standard frames (CAN specification 2.0A) or 29 bits for extended frames (CAN specification 2.0B). The three most significant bits are reserved.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved (0)																				11-Bit Identifier											
res. (0)			29-Bit Identifier																												

Data Fields

The data fields (**union tuMsgData**) may contain up to eight bytes for a CAN message. If the data length code is less than 8, the value of the unused data bytes will be undefined.

3

Data Length Code

The data length code field (**ubMsgDLC**) holds the number of valid bytes in the data field array. The allowed range is 0 to 8.

Message Control

The message control field (**ubMsgCtrl**) contains information about the frame type code. The EXT bit (bit 0) defines an Extended Frame (29 bit identifier) if set. The RTR bit (bit 1) defines a Remote Transmission Request if set. The OVR bit (bit 2) defines a Overrun during message reception if set.

7	6	5	4	3	2	1	0
reserved					OVR	RTR	EXT

Time Stamp

The time stamp field (**tsMsgTime**) defines the time when a CAN message was received by the CAN controller. The time stamp is an absolute value, based on Jan 1st 1970 00:00. The lowest possible resolution is one nanosecond (1 ns). This is an optional field.

The structure **`CpTime_s`** is defined similar to **`struct timeval`** (header file time.h) in the LINUX kernel

User Data

The field user data (**ulUserData**) can hold a 32 bit value, which is defined by the user. This is an optional field.

It is recommended to access the structure elements via function calls or macros, rather than dealing with bitmasks. Please refer to “CAN Message Functions” on page 41 for a detailed description.

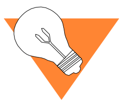
3

3.4 Bittiming

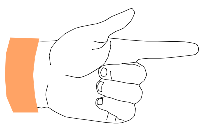
To ensure correct sampling up to the last bit, a CAN node needs to re-synchronize throughout the entire frame. This is done at the beginning of each message with the falling edge SOF and on each recessive to dominant edge.

One CAN bit time is specified as four non-overlapping time segments. Each segment is constructed from an integer multiple of the Time Quantum. The Time Quantum or TQ is the smallest discrete timing resolution used by a CAN node. The four time segments are:
- the Synchronization Segment
- the Propagation Time Segment
- the Phase Segment 1
- and the Phase Segment 2

The sample point is the point of time at which the bus level is read and interpreted as the value (recessive or dominant) of the respective bit. Its location is at the end of Phase Segment 1 (between the two Phase Segments).



Programming of the sample point allows "tuning" of the characteristics to suit the bus. Early sampling allows more Time Quanta in the Phase Segment 2 so the Synchronization Jump Width can be programmed to its maximum. This maximum capacity to shorten or lengthen the bit time decreases the sensitivity to node oscillator tolerances, so that lower cost oscillators such as ceramic resonators may be used. Late sampling allows more Time Quanta in the Propagation Time Segment which allows a poorer bus topology and maximum bus length.



In order to allow interoperability between CAN nodes of different vendors it is essential that both - the absolute bit length (e.g. 1µs) **and** the sample point - are within certian limits. The following table gives an overview of recommended bittiming setups.

Bitrate	Bittime	Valid range for sample point location	Recommended sample point location
1 MBit/s	1 µs	75% .. 90%	87,5%
800 kBit/s	1,25 µs	75% .. 90%	87,5%
500 kBit/s	2 µs	85% .. 90%	87,5%
250 kBit/s	4 µs	85% .. 90%	87,5%
125 kBit/s	8 µs	85% .. 90%	87,5%
50 kBit/s	20 µs	85% .. 90%	87,5%
20 kBit/s	50 µs	85% .. 90%	87,5%
10 kBit/s	100 µs	85% .. 90%	87,5%

Table 3: Recommended bit timing setup

The default baudrates defined in table 3 can be setup via the core function **`CpCoreBaudrate()`**. The supplied parameter for the baudrate selection are taken from the enumeration **`CP_BAUD`** (canpie.h).

Baudrate	Definition for default baudrate
10 kBit/s	CP_BAUD_10K
20 kBit/s	CP_BAUD_20K
50 kBit/s	CP_BAUD_50K
100 kBit/s	CP_BAUD_100K
125 kBit/s	CP_BAUD_125K
250 kBit/s	CP_BAUD_250K
500 kBit/s	CP_BAUD_500K
800 kBit/s	CP_BAUD_800K
1 MBit/s	CP_BAUD_1M

Table 4: Standard baudrates

If the pre-defined baudrates do not meet the requirements, it is possible to setup the CAN bittiming individually via the **`CpCoreBittiming()`** function.

3.5 CAN statistic information

Statistic information about a physical CAN interface can be gathered via the function **`CpCoreStatistic()`**. All counters are set to 0 upon initialisation of the CAN interface (**`CpCoreDriverInit()`**).

```
struct CpStats_s{

     // Total number of received data & remote frames
     _U32      ulRcvMsgCount;

     // Total number of transmitted data & remote
     // frames
     _U32      ulTrmMsgCount;

     // Total number of state change / error events
     _U32      ulErrMsgCount;
};

// typedef for this structure:
typedef struct CpStats_s   _TsCpStats;
```

3.6 Error Codes

All functions that may cause an error condition will return an error code. The CANpie error codes are within the value range from 0 to 127. The designer of the core functions might extend the error code table with hardware specific error codes, which must be in the range from 128 to 255.

Error Code	Description
CpErr_OK	No error occured
CpErr_GENERIC	Reason is not specified
CpErr_HARDWARE	Hardware failure
CpErr_INIT_FAIL	Initialisation failed
CpErr_CAN_MESSAGE	CAN message format is not valid
CpErr_CAN_ID	identifier is not valid
CpErr_CAN_DLC	data length code is not valid
CpErr_FIFO_EMPTY	FIFO (read or write) is empty
CpErr_FIFO_WAIT	message waiting in FIFO (read or write)
CpErr_FIFO_FULL	FIFO (read or write) is full
CpErr_FIFO_SIZE	not enough memory for FIFO
CpErr_BUS_PASSIVE	CAN controller is in bus passive state
CpErr_BUS_OFF	CAN controller is in bus off state
CpErr_BUS_WARNING	CAN controller is in warning state
CpErr_CHANNEL	channel number is out of range
CpErr_REGISTER	register address out of range
CpErr_BITRATE	bitrate is out of range / not supported
CpErr_BUFFER	buffer index is out of range
CpErr_NOT_SUPPORTED	the function is not supported

Table 5: CANpie error codes

4. Core Functions

The core functions provide the direct interface to the CAN controller (hardware). Please note that due to hardware limitations maybe certain functions are not implemented. A call to an unsupported function will always return the error code '`CpErr_NOT_SUPPORTED`'.

Function	Description
CpCoreAcceptance()	Setup acceptance filter
CpCoreAutobaud()	Start automatic baudrate detection
CpCoreBaudrate()	Set the bitrate of the CAN controller via pre-defined values
CpCoreBittiming()	Set the bitrate of the CAN controller via the bit timing registers / constant value
CpCoreCanMode	Set the mode of CAN controller
CpCoreCanState()	Retrieve the mode of CAN controller
CpCoreDriverInit()	Initialize the CAN driver
CpCoreDriverRelease()	Stop the CAN driver
CpCoreHDI()	Read the Hardware Description Information (HDI structure)
CpCoreIntFunctions()	Install callback functions for different CAN controller interrupts
CpCoreMsgRead()	Get a received message out of the CAN controller and put it into the Read FIFO
CpCoreMsgWrite()	Get a message from the Write FIFO and put it into the CAN controller (transmit)
CpCoreStatistic()	Get statistical information

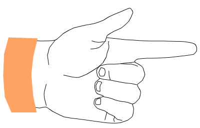
Table 6: Basic core functions

4

For a "FullCAN" controller (i.e. a CAN controller that has several message buffers) an extended set of powerful functions is provided.

Function	Description
CpCoreBufferEnable()	Temporarily enable / disable a message buffer
CpCoreBufferGetData()	Get message data from buffer
CpCoreBufferGetDlc()	Get data length code from buffer
CpCoreBufferInit()	Initialize message buffer in a FullCAN controller
CpCoreBufferRelease()	Release messge buffer in a FullCAN controller
CpCoreBufferSetData()	Set message data
CpCoreBufferSetDlc()	Set data length code
CpCoreBufferSend()	Send message out of specified buffer

Table 7: Core functions for buffer manipulation



Because the core functions are highly dependent on the hardware environment and the used operating system, the CANpie source package can only supply function bodies for these functions.

4.1 CpCoreBaudrate

Syntax

```
_TvCpStatus CpCoreBaudrate(
     _TsCpPort *      ptsPortV
     _U08             ubBaudSelV)
```

Function

Set Baudrate of CAN controller

This function initializes the bit timing registers of a CAN controller to pre-defined values. The values are defined in the "**canpie.h**" headerfile (enumeration CP_BAUD). Please refer to “Bittiming” on page 16 for a detailled description of common bittiming values.

Parameters

ptsPortV	Pointer to CAN port handle
ubBaudSelV	Baudrate selection (enumeration CP_BAUD)

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

Example

```
void MyCanInit(void)
{
   _TsCpPort  tsCanPortT;   // logical CAN port

   CpCoreDriverInit(CP_CHANNEL_1, &tsCanPortT);

   //--------------------------------------------------
   // setup 500 kBit/s
   //
   CpCoreBaudrate(&tsCanPortT, CP_BAUD_500K);

   //.. now we have a new baudrate setting

}
```

Example 3: Setup of baudrate

4.2 CpCoreBittiming

Syntax

```
_TvCpStatus CpCoreBittiming(
    _TsCpPort *     ptsPortV
    _TsCpBitTiming *ptsBitrateV);
```

Function

Set bittiming of CAN controller

This function directly writes to the bit timing registers of the CAN controller. Usage of the function requires a detailed knowledge of the used CAN controller hardware.

4

Parameters

ptsPortV	Pointer to CAN port handle
ptsBitrateV	Pointer to bit timing structure

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

Example

```
void SetCustomBaudrate(void)
{
   _TsCpPort           tsCanPortT;   // logical CAN port
   _TsCpBitTiming      tsBitTimeT;


   CpCoreDriverInit(CP_CHANNEL_1, &tsCanPortT);


   //--------------------------------------------------
   // setup Btr0 and Btr1 with user defined values
   //
   tsBitTimeT.ubBtr0 = 0x3F;
   tsBitTimeT.ubBtr1 = 0x1C;


   CpCoreBittiming(&tsCanPortT, &tsBitTimeT);

   //.. now we have a new baudrate setting

}
```

Example 4: Setup of user-defined bittiming

4.3 CpCoreBufferEnable

Syntax

```
_TvCpStatus CpCoreBufferEnable(
    _TsCpPort *     ptsPortV
    _U08            ubBufferIdxV,
    _U08            ubEnableV)
```

Function

Enable / Disable a message buffer

The functions enables or disables a message buffer for reception and transmission of messages. The message buffer has to be configured by **`CpCoreBufferInit()`** in advance.

In contrast to the **`CpCoreBufferRelease()`** function the contents message buffer is not deleted, but message reception and transmission can be suppressed by setting '**`ubEnableV`**' to 0.

4

Parameters

ptsPortV	Pointer to CAN port handle
ubBufferIdxV	Index of message buffer
ubEnableV	Flag to enable/disable message buffer

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**`CpErr_OK`**'.

4.4 CpCoreBufferGetData

Syntax

```
_TvCpStatus CpCoreBufferGetData(
    _TsCpPort *     ptsPortV
    _U08            ubBufferIdxV,
    _U08 *          pubDestDataV)
```

Function

Get data from message buffer

The functions copies 8 data bytes from the FullCAN message buffer defined by '**ubBufferIdxV**'. The destination buffer must have space for 8 bytes. The buffer has to be configured by **CpCoreBufferInit()** in advance.

4

Parameters

ptsPortV	Pointer to CAN port handle
ubBufferIdxV	Index of message buffer
pubDestDataV	Pointer to destination buffer

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4.5 CpCoreBufferGetDlc

Syntax

```
_TvCpStatus CpCoreBufferGetDlc(
    _TsCpPort *     ptsPortV,
    _U08            ubBufferIdxV,
    _U08 *          pubDlcV)
```

Function

Get DLC of specified buffer

This function retrieves the Data Length Code (DLC) of the specified buffer '**`ubBufferIdxV`**'.

Parameters

ptsPortV	Pointer to CAN port handle
ubBufferIdxV	Index of message buffer
pubDlcV	Pointer to destination buffer for DLC

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**`CpErr_OK`**'.

4.6 CpCoreBufferInit

Syntax

```
_TvCpStatus CpCoreBufferInit(
     _TsCpPort *       ptsPortV
     _TsCpCanMsg *     ptsCanMsgV,
     _U08              ubBufferIdxV,
     _U08              ubDirectionV)
```

Function

Initialise a message buffer (mailbox) in a FullCAN controller

This function initialises a message buffer in a FullCAN controller. The number of the message buffer inside the FullCAN controller is specified via the parameter '**ubBufferIdxV**'. A message buffer can be relased via the function **CpCoreBufferRelease()**.

4

Parameters

ptsPortV	Pointer to CAN port handle
ptsCanMsgV	Pointer to a CAN message structure
ubBufferIdxV	Index of message buffer
ubDirectionV	Direction of message (receive or transmit) **CP_BUFFER_DIR_RX**: receive **CP_BUFFER_DIR_TX**: transmit

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return **CpErr_OK**.

Example

```
void MyAllocationFunction(_TsCpPort * ptsCanPortV)
{
   _TsCpCanMsg  tsMyCanMsgT;   // temporary CAN message

   //--------------------------------------------------
   // set message buffer 1 as transmit buffer,
   // ID = 120, DLC = 2

   CpMsgSetStdId(&tsMyCanMsgT, 120);  // ID = 120
   CpMsgSetDlc(&tsMyCanMsgT, 2);
   CpCoreBufferInit(ptsCanPortV, &tsCanMsgT,
                    CP_BUFFER_1, CP_BUFFER_DIR_TX);


}
```

Example 5: Allocation of a message buffer

4.7 CpCoreBufferRelease

Syntax

```
_TvCpStatus CpCoreBufferRelease(
    _TsCpPort *      ptsPortV
    _U08             ubBufferIdxV)
```

Function

Release message buffer of FullCAN controller

The function releases the allocated message buffer specified by the parameter '**ubBufferIdxV**'. Both - reception and transmission - will be disabled on the specified message buffer afterwards.

Parameters

ptsPortV	Pointer to CAN port handle
ubBufferIdxV	Index of message buffer

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4.8 CpCoreBufferSetData

Syntax

```
_TvCpStatus CpCoreBufferSetData(
    _TsCpPort *     ptsPortV
    _U08            ubBufferIdxV,
    _U08 *          pubSrcDataV)
```

Function

Set data in message buffer

This function copies 8 data bytes into the message buffer defined by the parameter '**ubBufferIdxV**'. The message buffer has to be configured by **CpCoreBufferInit()** in advance. The source data pointer '**pubSrcDataV**' must point to an array of 8 bytes length.

4

Parameters

ptsPortV	Pointer to CAN port handle
ubBufferIdxV	Index of message buffer
pubSrcDataV	Pointer to source data buffer

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

```
_U08 aubDataT[8];       // buffer for 8 bytes

aubDataT[0] = 0x11;     // byte 0: set to 11hex
aubDataT[1] = 0x22;     // byte 1: set to 22hex

//--- copy the data to message buffer 1 ----------
CpCoreBufferSetData(ptsCanPortV, CP_BUFFER_1, &aubDataT);

//--- send this message --------------------------
CpCoreBufferSend(ptsCanPortV, CP_BUFFER_1);
```

Example 6: Manipulation of data in message buffer

4.9 CpCoreBufferSetDlc

Syntax

```
_TvCpStatus CpCoreBufferSetDlc(
    _TsCpPort *      ptsPortV
    _U08             ubBufferIdxV,
    _U08             ubDlcV)
```

Function

Set Data Length Code (DLC) of specified message buffer

This function sets the Data Length Code (DLC) of the specified message buffer '**ubBufferIdxV**'. The DLC value '**ubDlcV**' must be in the range from 0 to 8. The message buffer has to be configured by a call to **CpCoreBufferInit()** in advance.

4

Parameters

ptsPortV	Pointer to CAN port handle
ubBufferIdxV	Index of message buffer
ubDlcV	DLC value

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4.10 CpCoreBufferSend

Syntax

```
_TvCpStatus CpCoreBufferSend(
    _TsCpPort *      ptsPortV
    _U08             ubBufferIdxV)
```

Function

Send message from message buffer

This function transmits a message from the specified message buffer '**ubBufferIdxV**'. The message buffer has to be configured by a call to **CpCoreBufferInit()** in advance.

4

Parameters

ptsPortV	Pointer to CAN port handle
ubBufferIdxV	Index of message buffer

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4.11 CpCoreCanMode

Syntax

```
_TvCpStatus CpCoreCanMode(
    _TsCpPort *     ptsPortV
    _U08            ubModeV)
```

Function

Set operating mode of CAN controller

This function changes the operating mode of the CAN controller. Possible values for mode are defined in the CP_MODE enumeration. At least the modes **CP_MODE_STOP** and **CP_MODE_START** are supported. Other modes depend on the CAN controller capabilities.

Parameters

ptsPortV Pointer to CAN port handle

ubModeV New CAN controller mode

Parameter "ubModeV"	Description
CP_MODE_STOP	set controller into 'Stop' mode
CP_MODE_START	set controller into 'Operational' mode
CP_MODE_LISTEN_ONLY	set controller into 'Listen Only' mode
CP_MODE_SLEEP	set controller into sleep mode

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4.12 CpCoreCanStatus

Syntax

```
_TvCpStatus CpCoreCanState(
    _TsCpPort *      ptsPortV
    _U08 *           pubStateV)
```

Function

Retrieve state of CAN controller

This function retrieved the present state of the CAN controller. Possible values are defined in the CP_STATE enumeration. The state of the CAN controller is copied to the variable pointer '**pubStateV**'.

Parameters

4

ptsPortV Pointer to CAN port handle

pubStateV Pointer to CAN controller state variable

Possible state values	Description
CP_STATE ACTIVE	CAN controller is active, no errors
CP_STATE STOPPED	CAN controller is in stopped mode
CP_STATE_SLEEPING	CAN controller is in Sleep mode
CP_STATE_BUS_WARN	Warning level is reached
CP_STATE_BUS_PASSIVE	CAN controller is error passive
CP_STATE_BUS_OFF	CAN controller went into Bus-Off
CP_STATE_PHY_FAULT	General failure of physical layer detected
CP_STATE_PHY_H	Fault on CAN-H (Low Speed CAN)
CP_STATE_PHY_L	Fault on CAN-L (Low Speed CAN)

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4.13 CpCoreDriverInit

Syntax

```
_TvCpStatus CpCoreDriverInit(
    _U08            ubPhyIfV,
    _TsCpPort *     ptsPortV)
```

Function

Initialize the CAN driver

Parameters

ubPhyIfV	CAN channel of the hardware
ptsPortV	Pointer to CAN port handle

4

Return value

Error code defined in the "**canpie.h**" headerfile. Possible return values are:

- **CpErr_HARDWARE**
  Hardware failure occured, initialisation is not possible
- **CpErr_INIT_FAIL**
  Software failure occured, initialisation is not possible
- **CpErr_OK**
  Function returned without error condition

4.14 CpCoreDriverRelease

Syntax

```
_TvCpStatus CpCoreDriverRelease(
    _TsCpPort * ptsPortV)
```

Function

Release the CAN driver

Parameters

ptsPortV Pointer to CAN port handle

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4

4.15 CpCoreHDI

Syntax

```
_TvCpStatus CpCoreHDI(
    _TsCpPort *      ptsPortV
    _TsCpHdi *       ptsHdiV)
```

Function

Get Hardware Description Information

This function retrieves information about the used hardware.

Parameters

ptsPortV	Pointer to CAN port handle
ptsHdiV	Pointer to the "Hardware Description" structure

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

4.16 CpCoreIntFunctions

Syntax

```
_TvCpStatus CpCoreIntFunctions(
    _TsCpPort * ptsPortV,
    _U08 (* pfnRcvHandler) (_TsCpCanMsg *, _U08),
    _U08 (* pfnTrmHandler) (_TsCpCanMsg *, _U08),
    _U08 (* pfnErrHandler) (_U08)
```

Function

Install callback functions

This function will install three different callback routines in the interrupt handler. If you do not want to install a callback for a certain interrupt condition the parameter must be set to NULL.

The callback functions for receive and transmit interrupt have the following syntax:

```
    _U08 Handler(   _TsCpCanMsg * ptsCanMsgV,
                    _U08 ubBufferIdxV)
```

The callback function for the CAN status-change / error interrupt has the following syntax:

```
    _U08 Handler(   _U08 ubStateV)
```

4

Parameters

ptsPortV	Pointer to CAN port handle
pfnRcvHandler	Pointer to callback function for receive interrupt
pfnTrmHandler	Pointer to callback function for transmit interrupt
pfnErrHandler	Pointer to callback function for error interrupt

Return value

Error code defined in the "**canpie.h**" headerfile.

```
_U08 MyCanReceive(_TsCpCanMsg * ptsCanMsgV, _U08 ubBufferIdxV)
{
   switch( CpMsgGetStdId(ptsCanMsgV) )
   {
      case 0x022:
         // do something with ID 0x022
         break;
   }
}

main()
{
   //....
   CpCoreIntFunctions(&tsCanPortT, MyReceiveFunc, 0L, 0L);
   //...
}
```

Example 7: Install a callback for receive interrupt

4.17 CpCoreMsgRead

Syntax

```
_TvCpStatus CpCoreMsgRead(
     _TsCpPort *      ptsPortV,
     _TsCpCanMsg *    ptsBufferV,
     _U32 *           ulBufferSizeV)
```

Function

Read CAN message from queue

This function reads up to **ulBufferSizeV** number of CAN messages from the receive queue of the CAN driver and stores it into the location pointed by **ptsBufferV**. The parameter **ulBufferSizeV** holds the size of the buffer (in number of messages) before the function is called and the actual number of messages copied in the buffer after the function is called.

Parameters

ptsPortV	Pointer to CAN port handle
ptsBufferV	Pointer to buffer for CAN messages
ulBufferSizeV	Size of the message buffer

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '**CpErr_OK**'.

```
#define RCV_BUFFER_SIZE    64

static _TsCpCanMsg   atsRcvBufferS[RCV_BUFFER_SIZE]

_U08 MyCanRead(_TsCpPort * ptsCanPortV)
{
   // maximum receive buffer size
   _U32  ulBufferSizeT = RCV_BUFFER_SIZE;

   CpCoreMsgRead(ptsCanPortV, atsRcvBufferS, &ulBufferSizeT);

   if(ulBufferSizeT == 0)
   {
      // ... no messages in the receive queue
   }
   else
   {

   }
}
```

Example 8: Message read operation

4.18 CpCoreMsgWrite

Syntax

```
_U08 CpCoreMsgWrite(
     _TsCpPort *      ptsPortV,
     _TsCpCanMsg *    ptsBufferV,
     _U32 *           ulBufferSizeV)
```

Function

Write CAN message to queue

This function gets the next CAN message out of the Transmit FIFO and writes it to the appropriate registers of the CAN controller.

4

Parameters

ptsPortV	Pointer to CAN port handle
ptsBufferV	Pointer to buffer for CAN messages
ulBufferSizeV	Size of the message buffer

Return value

Error code defined in the "**canpie.h**" headerfile. If no error occured, the function will return '`CpErr_OK`'.

4.19 CpCoreStatistic

Syntax

```
_TvCpStatus CpCoreStatistic(
    _TsCpPort *     ptsPortV,
    _TsCpStats *    ptsStatsV )
```

Function

Get statistic information from CAN controller

This function copies statistic information into the structure _TsStats, which is passed via the pointer ptsStatsV.

Parameters

ptsPortV	Pointer to CAN port handle
ptsStatsV	Pointer to CAN statistic structure

4

Return value

Error code defined in the "**canpie.h**" headerfile. Possible return values are:

- **CpErr_CHANNEL**
  Channel number is out of range
- **CpErr_SUPPORTED**
  Function is not supported
- **CpErr_OK**
  Function returned without error condition

5. CAN Message Functions

Access to the members of the CAN message structure ***CpCanMsg_s*** shall be performed via macros or functions calls. This ensures - upon change of the CAN message structure - that the application does not have to be adopted.

The CAN message functions are implemented as conventionell functions as well as macros. The symbol **`CP_CAN_MSG_MACRO`** defines which implementation is used.

5.1 CpMsgGetData

Syntax

```
_U08 CpMsgGetData(
     _TsCpCanMsg *   ptsCanMsgV,
     _U08            ubPosV)
```

Function

Read data bytes from CAN message

This function retrieves the data of a CAN message. The parameter **'ubPosV'** must be within the range 0 .. 7.

Parameters

ptsCanMsgV	Pointer to CAN message structure
ubPosV	Zero based index of byte position

5

Return value

Data value at specified position.

```
void MyDataRead(_TsCpCanMsg * ptsCanMsgV)
{
   _U08 ubByte0T;
   ....

   //------------------------------------------------
   // read first data byte from CAN message, check
   // the data length code (DLC) first
   //
   if( CpMsgGetDlc(ptsCanMsgV) > 0 )
   {
      ubByte0T = CpMsgGetData(ptsCanMsgV, 0);

      ....
   }

   ....

}
```

Example 9: Get data byte from CAN message structure

5.2 CpMsgGetDlc

Syntax

```
_U08 CpMsgGetData(
     _TsCpCanMsg *    ptsCanMsgV)
```

Function

Read DLC value from CAN message

This function retrieves the data length code (DLC) of a CAN message.

Parameters

ptsCanMsgV	Pointer to CAN message structure

Return value

Data length code

```
void MyDataRead(_TsCpCanMsg * ptsCanMsgV)
{
   _U08 ubByte0T;
   ....

   //-------------------------------------------------
   // read first data byte from CAN message, check
   // the data length code (DLC) first
   //
   if( CpMsgGetDlc(ptsCanMsgV) > 0 )
   {
      ubByte0T = CpMsgGetData(ptsCanMsgV, 0);

      ....
   }

   ....

}
```

Example 10: Check data length code from CAN message structure

5

5.3 CpMsgGetExtId

Syntax

```
_U32 CpMsgGetExtId(
     _TsCpCanMsg *   ptsCanMsgV)
```

Function

Read extended identifier

The function retrieves the value for the identifier of an extended frame (CAN 2.0B).

Parameters

ptsCanMsgV	Pointer to CAN message structure

Return value

Value for extended identifier in the range 0 .. 1FFFFFFFh

5

5.4 CpMsgGetStdId

Syntax

```
_U16 CpMsgGetStdId(
    _TsCpCanMsg *   ptsCanMsgV)
```

Function

Read standard identifier

The function retrieves the value for the identifier of a standard frame (CAN 2.0A).

Parameters

ptsCanMsgV	Pointer to CAN message structure

Return value

Value for standard identifier in the range 0 .. 7FFh

5.5 CpMsgIsExtended

Syntax

```
_U08 CpMsgIsExtended(
    _TsCpCanMsg *   ptsCanMsgV)
```

Function

Test for extended frame

Parameters

ptsCanMsgV Pointer to CAN message structure

Return value

TRUE on extended frame, FALSE on standard frame.

5

5.6 CpMsgIsRemote

Syntax

```
__U08 CpMsgIsRemote(
    _TsCpCanMsg *   ptsCanMsgV)
```

Function

Test for remote frame

Parameters

ptsCanMsgV Pointer to CAN message structure

Return value

TRUE on remote frame, FALSE on data frame.

5.7 CpMsgClear

Syntax

```
void CpMsgClear(
    _TsCpCanMsg *   ptsCanMsgV)
```

Function

Clear CAN message structure

The function clears the elements of a CAN message structure.

Parameters

ptsCanMsgV Pointer to CAN message structure

Return value

None

5.8 CpMsgSetData

Syntax

```
void CpMsgSetData(
    _TsCpCanMsg *   ptsCanMsgV,
    _U08            ubPosV,
    _U08            ubValueV)
```

Function

Write data bytes to CAN message

This function sets the data of a CAN message. The parameter **'ubPosV'** must be within the range 0 .. 7.

Parameters

ptsCanMsgV	Pointer to CAN message structure
ubPosV	Zero based index of byte position
ubValueV	Data value for CAN message

Return value

None

5.9 CpMsgSetDlc

Syntax

```
_U08 CpMsgSetDlc(
     _TsCpCanMsg *    ptsCanMsgV,
     _U08             ubDlcV)
```

Function

Set DLC value of CAN message

This function sets the data length code (DLC) of a CAN message. The value must be within the range 0 .. 8.

Parameters

ptsCanMsgV	Pointer to CAN message structure
ubDlcV	Data length code value

5

Return value

None

```
_TsCpCanMsg  tsMyCanMsgT;  // temporary CAN message struct.

//-----------------------------------------------------
// clear message and setup CAN-ID = 100h and DLC = 4
CpMsgClear(&tsMyCanMsgT);
CpMsgSetStdId(&tsMyCanMsgT, 0x0100);   // set ID = 0x0100
CpMsgSetDlc(&tsMyCanMsgT, 4);          // set DLC = 4
```

Example 11: Setup the data length code

5.10 CpMsgSetExtId

Syntax

```
void CpMsgSetExtId(
    _TsCpCanMsg *   ptsCanMsgV,
    _U32            ulExtIdV)
```

Function

Set 29-bit identifier value

Parameters

ptsCanMsgV	pointer to CAN message structure
ulExtIdV	extended identifier value

Return value

None

5.11 CpMsgSetRemote

Syntax

```
void CpMsgSetRemote(
    _TsCpCanMsg *   ptsCanMsgV)
```

Function

Set RTR bit

This function sets the remote transmission bit (RTR) in the CAN message structure.

Parameters

ptsCanMsgV	Pointer to CAN message structure

Return value

None

5

5.12 CpMsgSetStdId

Syntax

```
void CpMsgSetStdId(
    _TsCpCanMsg *   ptsCanMsgV,
    _U16            uwStdIdV)
```

Function

Set 11-bit identifier value

Parameters

ptsCanMsgV	pointer to CAN message structure
uwStdIdV	standard identifier value

Return value

None

5

A LGPL LICENSE

Version 2.1, February 1999

This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software -- to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE

A

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Section 0

This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

Section 1

You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.
b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

   (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

A

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contestyour rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

Section 4

You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

Section 5

A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

Section 6

As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

Section 7

You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

Section 8

You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 9

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

Section 10

Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

Section 11

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 12

If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 13

The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

Section 14

If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

Section 15

BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Section 16

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A

B Index

B

C

S

I

MicroControl
Systemhaus für Automatisierung

MicroControl GmbH & Co. KG
Lindlaustraße 2c
D-53842 Troisdorf
Fon: +49 / 2241 / 25 65 9 - 0
Fax: +49 / 2241 / 25 65 9 - 11
http://www.microcontrol.net