

CANpie

Controller Area Network Programming Interface Environment
Version 0.8 Rev. A

User Manual

Contents

1 Overview	1
1.1 Document Conventions	1
1.2 Project Goals	2
1.3 License	2
1.4 Compiler and OS support	3
1.5 History	4
1.6 References	5
2 Driver Principle	6
2.1 Message Distribution	7
2.2 Compiler / Operating System Settings	9
2.3 Hardware Description Interface	10
2.4 Structure of a CAN message	12
2.5 Initialization Process	14
3 API Overview	15
3.1 CAN Message Access	16
3.1.1 CAN Message Access Macros	16
3.1.2 CAN Message Programming Examples	16
3.2 Error Codes	17
4 User-Functions	18
4.1 CpUserAppInit	19
4.2 CpUserAppDeInit	20
4.3 CpUserBaudrate	21
4.4 CpUserFifoClear	22
4.5 CpUserFilterAll	23
4.6 CpUserFilterMsg	24
4.7 CpUserIntFunctions	25
4.8 CpUserMsgRead	26
4.9 CpUserMsgWrite	27

5	Core-Functions	28
5.1	CpCoreAllocBuffer	30
5.2	CpCoreBaudrate	31
5.3	CpCoreBufferGetData	32
5.4	CpCoreBufferGetDlc	33
5.5	CpCoreBufferSetData	34
5.6	CpCoreBufferSetDlc	35
5.7	CpCoreBufferSend	36
5.8	CpCoreBufferTransmit	37
5.9	CpCoreCanMode	38
5.10	CpCoreCanState	39
5.11	CpCoreDeAllocBuffer	40
5.12	CpCoreDeInitDriver	41
5.13	CpCoreFilterAll	42
5.14	CpCoreFilterMsg	43
5.15	CpCoreHDI	44
5.16	CpCoreInitDriver	45
5.17	CpCoreIntHandler	46
5.18	CpCoreMsgReceive	47
5.19	CpCoreMsgTransmit	48
5.20	CpCoreRegRead	49
5.21	CpCoreRegWrite	50

1. Overview

This document describes the project "Controller Area Network Programming Interface Environment" (CANpie). It is divided in five chapters:

Chapter 1 includes conventions and explains the project goals. It also tells something about the copyright and references.

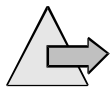
Chapter 2 describes the function principle of CANpie. The concepts explained here are the basics for the API in the following chapters.

Chapter 3 gives an insight into the basic structures of the API and explains the access to the defined CAN messages.

Chapter 4 describes the CANpie User-Functions in detail.

Chapter 5 describes the CANpie Core-Functions in detail.

1.1 Document Conventions



Note

Text marked with the symbol **Note** describes useful hints or future extensions of the API. It is recommended to read these sections.



Attention !

Text marked with the symbol **Attention** describes important information concerning the API. Please read these sections carefully.

Keywords

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

Syntax, Examples

For function syntax and code examples the font face `courier` is used.

1.2 Project Goals

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, SDS, DeviceNet).

Wherever 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.

1.3 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.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

1.4 Compiler and OS support

To compile CANpie you simply need a C compiler. CANpie is tested against the following compiler / operating systems:

Microsoft Visual C++ (Windows)	tested
Keil C (for 8051 family)	tested
Borland C++ (DOS)	tested
GCC (Linux)	tested

Table 1: CANpie compiler / OS support

1.5 History

Version	Comment
0.1	Initial Version
0.2	<ul style="list-style-type: none">- Support for FullCAN controller- Callback functions
0.3	<ul style="list-style-type: none">- Changed structures- New data type definition- New file layout
0.4	<ul style="list-style-type: none">- Bugfix in FIFO- new function for FullCAN support- adaption to Keil-C compiler
0.5	<ul style="list-style-type: none">- New symbols for cpconfig.h file
0.6	<ul style="list-style-type: none">- Introduction of macros for message manipulation- static FIFO support- New definitions for CAN message structure
0.7	<ul style="list-style-type: none">- New Error codes- Bugfix for static FIFOs
0.8	<ul style="list-style-type: none">- Rewrite of FIFO functions

Table 2: Version History

1.6 References

[1] CAN Specification, Version 2.0, September 1991, Robert Bosch GmbH,

<http://www.MicroControl.net/download/can2spec.pdf>

[2] PPCAN Device Driver, Programmer's Guide and Reference, Mecel AB 1997

<http://www.mecel.se/assets/images/guide32.pdf>

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.

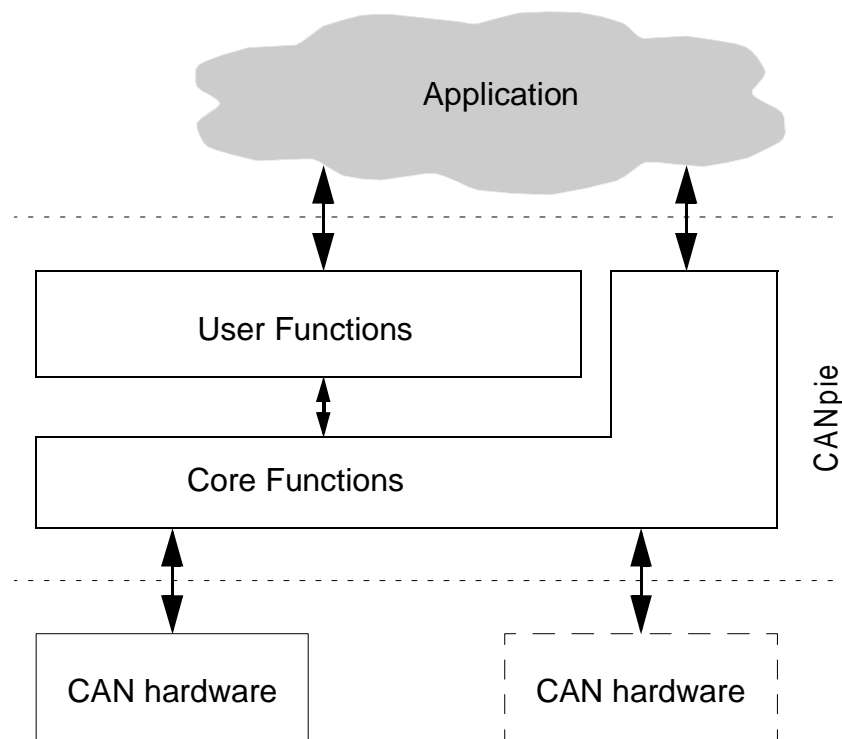


Figure 2.1: CANpie Structure

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. A typical user function is the writing of a CAN message to the FIFO buffer (refer to “CpUserMsgWrite” on page 27).

Core Functions

The core functions access the hardware directly, so an adaption is necessary when implementing on a piece of hardware. Core functions may also be called by the application (in that case the application engineer must have a good knowledge about the CAN hardware). A typical core function is reading from a CAN controller register (refer to “CpCoreRegRead” on page 49).

CANpie supports more than one CAN channel on the hardware. How-

ever, the total number of CAN channels is limited to 16. The actual number of CAN channels can be gathered via the Hardware Description Interface (refer to “Hardware Description Interface” on page 10).

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 or transmit).

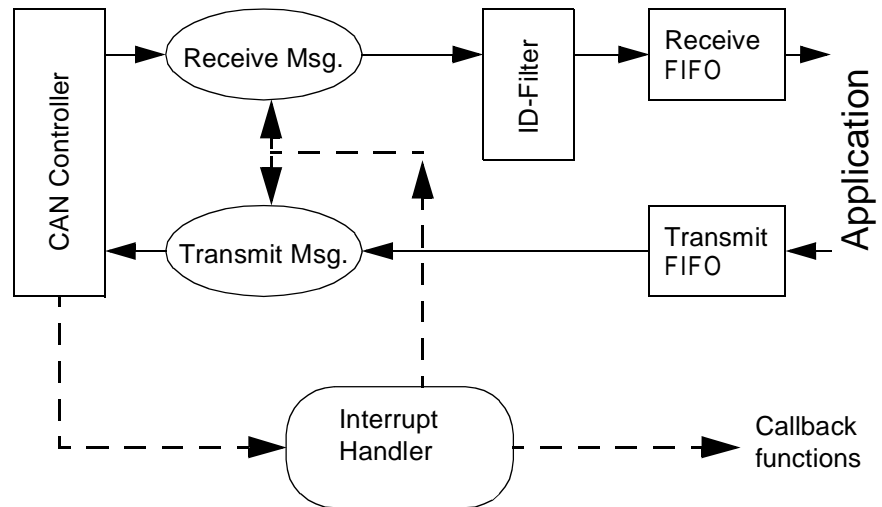


Figure 2.2: Message Distribution

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.

Callback Functions

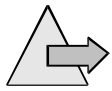
In case of a transmit interrupt the Transmit FIFO is checked. If there are messages in this queue, the Transmit Message routine will write the next waiting message to the CAN controller. If the Transmit FIFO is empty and the application puts a CAN message into the queue, the Transmit Message routine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

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

- Receive interrupt
- Transmit interrupt
- CAN controller error

Driver Principle

Message Distribution

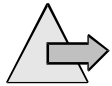


Note

The handler functions are installed by the `CpUserIntFunctions()` function call. It is possible to configure CANpie not using hardware interrupts. In this case the state of the Receive FIFO has to be checked by polling.

The size of the FIFO buffers can be configured during the initialization process. The maximum size is limited to 65535 messages for each buffer.

Identifier Filter



Note

On its way to the Receive FIFO the CAN message must pass an optional software identifier filter. This software filter can be used by the programmer to enhance the capabilities of the CAN controller. The software filter only supports CAN standard frames (2.0A), because otherwise the overhead for extended frames would be too big. For transmission of messages the filter is not used.

2.2 Compiler / Operating System Settings

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

Data Types

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

Table 3: Data Type definitions

Function Prefix

For certain operating systems or C compiler a function specifier (function prefix) is needed. This prefix (CpPREFIX) is also defined in the file "**compiler.h**". For example, a Windows implementation might have the form:

```
#define CpPREFIX _export far pascal

#define CpEXPORT _export far pascal
```

2.3 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{
    _U08 v_SupportFlags;
    _U08 v_ControllerType;
    _U08 v_IRQNumber;
    _U08 v_VersionMajor;
    _U08 v_VersionMinor;
    char *v_DriverName;
    char *v_VersionNumber;
};

typedef struct CpHdi_s_TsCpHdi;

typedef struct CpHdi_sCpStruct_HDI;
```

All items in the structure "**CpStruct_HDI**" are constant and must be supplied by the designer of the CAN driver. 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 **v_SupportFlags** describe the possibilities 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 (refer to "Time Stamp" on page 13).

User Data

If this flag is set to "1", the element **v_UserData** 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 "**cpconst.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.

Driver Name

A null terminated string which holds the name of the specific CAN driver. The string can be used to distinguish between different vendors.

Version Number

A null terminated string which holds the version of the specific CAN driver (e.g. DLL version, firmware version).

2.4 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 has the following data fields:

```

struct CpCanMsg_s{
    _U32 v_MsgId;      /* Identifier          */
    _U32 v_MsgFlags; /* Flags              */
    _U08 v_MsgData[8];/* Data Fields        */
    _U32 v_MsgTime;   /* Time Stamp (opt.) */
    _U32 v_UserData;  /* User Data (opt.)  */
};

// typedef for this structure:
typedef struct CpCanMsg_s_TsCpCanMsg;

// typedef for compatibility with older versions:
typedef struct CpCanMsg_sCpStruct_CAN;
    
```

Identifier

The identifier field (**v_MsgId**) 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 for special functionality (the LSB is Bit 0, the MSB is Bit 31):

- Bit 31: Bit value "1" marks the identifier as an extended frame. Bit value "0" marks the identifier as a standard frame.
- Bit 30: Bit value "1" marks the identifier as an remote transmission (RTR).
- Bit 29: Reserved for future use

Message Flags

The message flags field (**v_MsgFlags**) contains the data length code (DLC) of the CAN message and the buffer number when using a Full-CAN controller.

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
User defined data																reserved				Message Buffer		DLC									

The data length code (**DLC**) contains the number of data bytes which are transmitted by a message. The possible value range for the data length code is from 0 to 8 (bytes).

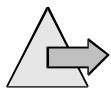
A FullCAN controller (e.g. AN82527) has more than only one transmit and one receive buffer and offers more sophisticated message filtering. The field message buffer (Bit 4 - Bit 7) specifies the buffer for message transmission or reception.

The high word (Bit 16 - Bit 31) of the member **v_MsgFlags** is reserved for user defined data.

Driver Principle

Structure of a CAN message

Data Fields	The data fields (v_MsgData[]) 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.
Time Stamp	The time stamp field (v_MsgTime) defines the time when a CAN message was received by the CAN controller. The time stamp is a relative value, which is created by a free running timer. The time base is one microsecond (1 μ s). This means a maximum time span of 4294,96 seconds (1 hour 11 minutes) between two messages can be measured. This is an optional field.
User Data	The field user data (v_UsrData) can hold a 32 bit value, which is defined by the user. This is an optional field.



Note

It is recommended to access the structure elements via function calls or macros, rather than dealing with bitmasks.

2.5 Initialization Process

The CAN driver is initialized with the function **CpUserApplnit()**. This routine will call several core functions for the correct set up of the CAN controller. The core functions called are:

- CpFifoSetup()
- CpCoreInitDriver()
- CpUserBaudrate()
- CpUserFilterAll()

The initial baudrate is set to 20kBit/s by default.



Attention !

The function **CpUserApplnit()** must be called before any other CANpie function.

3. API Overview

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:

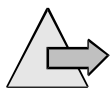
CANpie	Prefix
User Functions	CpUser
Core Functions	CpCore
Structures	_TsCp
Constants / Defines	CP
Error Codes	CpErr

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

CAN "channel" Parameter

Most of the core and user functions use the parameter "channel", which defines the CAN channel of the hardware. The channel number is defined in the following way:

```
0    first channel
1    second channel
...
```



Note

For an embedded application with only one CAN channel this parameter is not only useless, it also consumes code space. In a future CANpie release the source code will be modified (by using a compiler switch) so that the parameter "channel" does not occupy code or RAM space. However, the function calling conventions (including the "channel" parameter) will stay the same!

3.1 CAN Message Access

The access functions for the structure **CpCanMsg_s** are defined in the header file `cpmsg.h`.

3.1.1 CAN Message Access Macros

The access macros for the structure **CpCanMsg_s** are defined in the header file `cpmacro.h`.

3.1.2 CAN Message Programming Examples

The following example shows how to set up a CAN message in both ways: with function calls and with macros.

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

//-----
// Access with function calls
CpMsgClear(&tsMyCanMsgT);
CpMsgSetStdId(&tsMyCanMsgT, 0x0100); // set ID = 0x0100
CpMsgSetDlc(&tsMyCanMsgT, 4); // set DLC = 4
```

Example 1: Access to CAN message structure

3.2 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 occurred
CpErr_GENERIC	Reason is not specified
CpErr_HARDWARE	Hardware failure
CpErr_INIT_FAIL	Initialisation failed
CpErr_INIT_READY	Initialisation was already done
CpErr_INIT_OK	Initialisation was already done
CpErr_CAN_MESSAGE	CAN message format is not valid
CpErr_CAN_ID	identifier 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_BAUDRATE	baudrate is out of range
CpErr_NOT_SUPPORTED	the function is not supported

4. User-Functions

The CANpie User-Functions are suited for applications that require an abstraction layer over a CAN hardware.

The following table shows the available CANpie User-Functions.

Function	Description
CpUserAppInit()	Initialization routine which is called by the application
CpUserAppDeInit()	De-Initialization routine which is called by the application
CpUserBaudrate()	Set the baudrate of a CAN controller via a constant value (pre-defined baudrates)
CpUserFifoClear()	Clear the contents of a FIFO buffer (Receive or Transmit)
CpUserFilterAll()	Enable or disable the reception of all standard frame CAN messages
CpUserFilterMsg()	Enable or disable the reception of a single CAN message (standard frame ID)
CpUserIntFunctions()	Install callback functions for different CAN controller interrupts
CpUserMsgRead()	Read a CAN message from the Receive FIFO buffer
CpUserMsgWrite()	Write a CAN message to the Transmit FIFO buffer

4.1 CpUserApplInit

Syntax

```
_U08 CpUserAppInit(  
    _U08 ubChannelV,  
    _U16 uwRcvFifoSizeV, _U16 uwTrmFifoSizeV,  
    _U16 uwTimeoutV)
```

Function

Initialize the CAN driver

This function must be called by the application program prior to all other functions. It is responsible for initializing the CAN controller interface.

Parameters

ubChannelV CAN channel of the hardware

uwRcvFifoSizeV Size of the FIFO for receiving CAN messages (number of messages).

uwTrmFifoSizeV Size of the FIFO for sending CAN messages (number of messages)

uwTimeoutV Timeout value for transmission of messages in milliseconds

Return value

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

4.2 CpUserAppDeInit

Syntax	<code>_U08 CpUserAppDeInit(_U08 ubChannelV)</code>
Function	<p>De-Initialize the CAN driver</p> <p>This function must be called when the application program quits.</p>
Parameters	<p>ubChannelV CAN channel of the hardware</p>
Return value	<p>Error code defined in the "cpconst.h" headerfile. If no error occurred, the function will return CpErr_OK.</p>

4.3 CpUserBaudrate

Syntax

```
_U08 CpUserBaudrate(  
    _U08 ubChannelV,  
    _U08 ubBaudV)
```

Function

Set baudrate

This function sets the baudrate of the selected CAN channel. Possible values for the baudrate are:

Baudrate	Parameter "ubBaudV"
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

Parameters

ubChannelV CAN channel of the hardware
ubBaudV Constant for the baudrate

Return value

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

Example

```
main()  
{  
    //....  
    CpUserAppInit(CP_CHANNEL_1, 2, 2, 1);  
    CpUserBaudrate(CP_CHANNEL_1, CP_BAUD_125K);  
    //....  
}
```

Example 2: Set baudrate to 125 kBit/sec

User-Functions

CpUserFifoClear

4.4 CpUserFifoClear

Syntax

```
_U08 CpUserFifoClear(  
    _U08 ubChannelV,  
    _U08 ubBufferV)
```

Function

Clear FIFO buffer

Deletes all messages of the specified FIFO buffer.

4

Parameters

ubChannelV CAN channel of the hardware

buffer **CP_FIFO_RCV** for the Receive FIFO or
 CP_FIFO_TRM for the Transmit FIFO

Return value

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

4.5 CpUserFilterAll

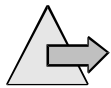
Syntax

```
_U08 CpUserFilterAll(  
    _U08 ubChannelV,  
    _BIT btEnableV)
```

Function

Set global filter

This function enables or disables the reception of CAN messages. If the flag '**btEnableV**' is set to TRUE, all incoming CAN messages are accepted. If the flag '**btEnableV**' is set to FALSE, all CAN messages are rejected. Please notice that this function only works for standard messages (2.0A, 11 bit identifier).



Note

This function is only available, if the member '**SupportSWFilter**' of the HDI structure (**CpStruct_HDI**) is set to TRUE. For more details about the HDI structure refer to "Hardware Description Interface" on page 10 of this document.

Parameters

ubChannelV	CAN channel of the hardware
btEnableV	TRUE for accepting all identifiers, FALSE for rejecting all identifiers

Return value

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

4.6 CpUserFilterMsg

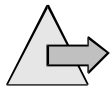
Syntax

```
_U08 CpUserFilterMsg(  
    _U08 ubChannelV  
    _U16 uwStdIdV,  
    _BIT btEnableV)
```

Function

Set filter for receiving messages

This function sets a filter for CAN messages. If the flag `btEnableV` is set to TRUE, the CAN identifier `'uwStdIdV'` is accepted. If the flag `'btEnableV'` is set to FALSE, the reception of messages with the identifier `'uwStdIdV'` is not possible. Please notice that this function only works for standard messages (2.0A, 11 bit identifier).



Note

This function is only available, if the member `'supportSWFilter'` of the HDI structure (`CpStruct_HDI`) is set to TRUE. For more details about the HDI structure refer to "Hardware Description Interface" on page 10 of this document.

Parameters

<code>ubChannelV</code>	CAN channel of the hardware
<code>uwStdIdV</code>	value of the identifier
<code>btEnableV</code>	TRUE for accepting an identifier, FALSE for rejecting an identifier

Return value

Error code defined in the `"cpconst.h"` headerfile. If no error occurred, the function will return `CpErr_OK`.

4.7 CpUserIntFunctions

Syntax

```
_U08 CpUserIntFunctions(  
    _U08 ubChannelV,  
    _U08 (* pfnRcvHandler) (_TsCpCanMsg *),  
    _U08 (* pfnTrmHandler) (_TsCpCanMsg *),  
    _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 or disable the callback, the parameter must be set to NULL.

Parameters

- ubChannelV CAN channel of the hardware
- 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 "**cpconst.h**" headerfile.

```
_U08 MyReceiveFunc(_TsCpCanMsg * ptsCanMsgV)  
{  
    switch( CpMacGetStdId(ptsCanMsgV) )  
    {  
        case 0x022:  
            // do something with ID 0x022  
            break;  
    }  
}  
  
main()  
{  
    //....  
    CpUserAppInit(CP_CHANNEL_1, 2, 2, 1);  
    CpUserIntFunctions(CP_CHANNEL_1, MyReceiveFunc, 0L, 0L);  
    //....  
}
```

Example 3: Install a Callback

4.8 CpUserMsgRead

Syntax	<pre>_U08 CpUserMsgRead(_U08 ubChannelV, _TsCpCanMsg *ptsCanMsgV)</pre>
Function	Read CAN message from Receive FIFO
Parameters	<p>ubChannelV CAN channel of the hardware</p> <p>ptsCanMsgV Pointer to a CAN message structure</p>
Return value	Error code defined in the " cpconst.h " headerfile. If no error occurred, the function will return CpErr_OK .

4

```
void MyReadFunction(void)
{
    _U08  ubStatusT;
    _U08  ubRunPollingT = 1;

    while(ubRunPollingT)
    {
        ubStatus = CpUserMsgRead(CP_CHANNEL_1, &tsCanMsgT);
        if(ubStatusT != CpErr_OK) ubRunPollingT = 0;

        //.. do something with the CAN message
    }
}
```

Example 4: Polling CAN messages

4.9 CpUserMsgWrite

Syntax

```
_U08 CpUserMsgWrite(  
    _U08      ubChannelV,  
    _TsCpCanMsg *ptsCanMsgV)
```

Function

Write CAN message to Transmit FIFO

The CAN message pointed to by ptsCanMsgV is written to the FIFO (Transmit). The appropriate core functions care for the transmission of the message.

Parameters

ubChannelV CAN channel of the hardware

ptsCanMsgV Pointer to a CAN message structure

Return value

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

```
void MyCanWriteFunction(void)  
{  
    _TsCpCanMsg  tsCanMsgT; // CAN message structure  
  
    //-----  
    // setup message with ID = 0x010, DLC = 2  
    //  
    CpMacSetStdId(&tsCanMsgT, 0x010);  
    CpMacSetDlc(&tsCanMsgT, 2);  
    CpMacSetData(&tsCanMsgT, 0, 0x11);  
    CpMacSetData(&tsCanMsgT, 1, 0x22);  
  
    CpUserMsgWrite(CP_CHANNEL_1, &tsCanMsgT);  
}
```

Example 5: Transmit a CAN message

5. 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 unavailable functions will always return the error code 'CpErrr_NOT_SUPPORTED'.

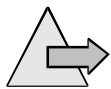
For "FullCAN" controller (i.e. CAN controller that have several message buffers) an extended set of powerful functions is provided.

Function	Description
CpCoreBaudrate()	Set the baudrate of the CAN controller via the bit timing registers
CpCoreCanMode	Set the mode of CAN controller
CpCoreCanState()	Retrieve the mode of CAN controller
CpCoreDeInitDriver()	De-Initialize the CAN driver
CpCoreFilterAll()	Hardware filter function (all messages)
CpCoreFilterMsg()	Hardware filter function
CpCoreHDI()	Read the Hardware Description Information (HDI structure)
CpCoreInitDriver()	Initialize the CAN driver (enable interrupt, set CAN register,...)
CpCoreIntHandler()	Checks the interrupt requests of the CAN controller
CpCoreMsgReceive()	Get a received message out of the CAN controller and put it into the Read FIFO
CpCoreMsgTransmit()	Get a message from the Write FIFO and put it into the CAN controller (transmit)
CpCoreRegRead()	Read the contents of a CAN controller register
CpCoreRegWrite()	Write to the register of a CAN controller

Table 4: Basic Core-Functions

Function	Description
CpCoreAllocBuffer()	Allocate a message buffer in a FullCAN controller
CpCoreBufferGetData()	Get message data from buffer
CpCoreBufferGetDlc()	Get data length code from buffer
CpCoreBufferSetData()	Set message data
CpCoreBufferSetDlc()	Set data length code
CpCoreBufferSend()	Send message out of specified buffer (previously configured)
CpCoreBufferTransmit()	Send message out of specified buffer (configuration inside the function)
CpCoreDeAllocBuffer()	Deallocate a message buffer in a FullCAN controller

Table 5: Core-Functions for buffer manipulation



Note

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.

5.1 CpCoreAllocBuffer

Syntax

```
_U08 CpCoreAllocBuffer(
    _U08          ubChannelV
    _TsCpCanMsg * ptsCanMsgV,
    _U08          ubDirectionV)
```

Function

Allocate the message buffer in a FullCAN controller

This function allocates the message buffer in a FullCAN controller (e.g. AN82527). The number of the message buffer inside the FullCAN controller is coded via the field `v_MsgFlags` in the CAN message structure. For message direction "Receive" an interrupt handler has to be configured via a function call to `CpUserIntFunctions()`. For message direction "Transmit" the functions `CpCoreBufferSend()` as well as `CpCoreBufferTransmit()` can be used.

Parameters

- `ubChannelV` CAN channel of the hardware
- `ptsCanMsgV` Pointer to a CAN message structure, the field `v_MsgFlags` holds the number of the message buffer inside the CAN controller
- `ubDirectionV` Direction of message (receive or transmit)
 `CP_BUFFER_DIR_RX`: receive
 `CP_BUFFER_DIR_TX`: transmit

Return value

Error code defined in the "`cpconst.h`" headerfile. If no error occurred, the function will return `CpErr_OK`.

Example

```
void MyAllocationFunction(_U16 uwStdIdV)
{
    _TsCpCanMsg tsMyCanMsgT;    // temporary CAN message

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

    CpMacSetStdId(&tsMyCanMsgT, uwStdIdV);
    CpMacSetDlc(&tsMyCanMsgT, 2);
    CpMacSetBufNum(&tsCanMsgT, CP_BUFFER_1);

    CpCoreAllocBuffer(CP_CHANNEL_1, &tsCanMsgT,
                     CP_BUFFER_DIR_TX);
}
```

Example 6: Allocation of a message buffer

5.2 CpCoreBaudrate

Syntax

```
_U08 CpCoreBaudrate(  
    _U08 ubChannelV  
    _U08 ubBtr0V, _U08 ubBtr1V,  
    _U08 ubBaudSelV)
```

Function

Set Baudrate of CAN controller

This function directly writes to the bit timing registers of the CAN controller. It is called by [CpUserBaudrate\(\)](#) with pre-defined values for common baudrates.

Parameters

ubChannelV	CAN channel of the hardware
ubBtr0V	Value for Bit Timing Register 0
ubBtr1V	Value for Bit Timing Register 1
ubBaudSelV	Baudrate selection

Return value

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

The parameter 'ubBaudSelV' can take the same constant values as defined for [CpUserBaudrate\(\)](#). In order to use the values given by 'ubBtr0V' and 'ubBtr1V' the parameter 'ubBaudSelV' must be set to 'CP_BAUD_BTR'.

Example

```
void SetCustomBaudrate(void)  
{  
    //-----  
    // setup Btr0 and Btr1 with user defined values  
    //  
    CpCoreBaudrate(CP_CHANNEL_1, 0x12, 0x34, CP_BAUD_BTR);  
  
    //.. now we have a new baudrate setting  
}
```

Example 7: Setup of user-defined baudrate

5.3 CpCoreBufferGetData

Syntax

```
_U08 CpCoreBufferGetData(  
    _U08    ubChannelV,  
    _U08    ubBufferV,  
    _U08 *  pubDataV)
```

Function

Get data from message buffer

This function is the fastest method to get the data bytes of a CAN message. It can be used in combination with a receive callback handler (see page 25). The function copies 8 data bytes from the buffer defined by 'ubBufferV'. The destination buffer must have space for 8 bytes. The buffer has to be configured by [CpCoreAllocBuffer\(\)](#) in advance.

Parameters

ubChannelV	CAN channel of the hardware
ubBufferV	Number of message buffer
pubDataV	Pointer to destination buffer

Return value

Error code defined in the "[cpconst.h](#)" headerfile. If no error occurred, the function will return 'CpErr_OK'.

5.4 CpCoreBufferGetDlc

Syntax

```
_U08 CpCoreBufferGetDlc(  
    _U08    ubChannelV,  
    _U08    ubBufferV,  
    _U08 *  pubDlcV)
```

Function

Get DLC of specified buffer

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

5**Parameters**

ubChannelV	CAN channel of the hardware
ubBufferV	Number of message buffer
pubDlcV	Pointer to destination buffer for DLC

Return value

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

5.5 CpCoreBufferSetData

Syntax

```
_U08 CpCoreBufferSetData(  
    _U08    ubChannelV,  
    _U08    ubBufferV,  
    _U08 *  pubDataV)
```

Function

Set data in message buffer

This function is the fastest method to set the data bytes of a CAN message. It can be used in combination with `CpCoreBufferSend()`. The function copies 8 data bytes into the buffer defined by 'ubBufferV'. The buffer has to be configured by `CpCoreAllocBuffer()` in advance.

5**Parameters**

ubChannelV CAN channel of the hardware
ubBufferV Number of message buffer
pubDataV Pointer to destination buffer

Return value

Error code defined in the "cpconst.h" headerfile. If no error occurred, 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(CP_CHANNEL_1, CP_BUFFER_1, &aubDataT);  
  
//--- send this message -----  
CpCoreBufferSend(CP_CHANNEL_1, CP_BUFFER_1);
```

Example 8: Manipulation of data in message buffer

5.6 CpCoreBufferSetDlc

Syntax

```
_U08 CpCoreBufferSetDlc(  
    _U08    ubChannelV,  
    _U08    ubBufferV,  
    _U08    ubDlcV)
```

Function

Set DLC of specified buffer

This function sets the Data Length Code (DLC) of the specified buffer 'ubBufferV'. The DLC must be in the range from 0 to 8.

5

Parameters

ubChannelV	CAN channel of the hardware
ubBufferV	Number of message buffer
ubDlcV	New DLC value

Return value

Error code defined in the "cpconst.h" headerfile. If no error occurred, the function will return 'CpErr_OK'.

5.7 CpCoreBufferSend

Syntax

```
_U08 CpCoreBufferSend(  
        _U08    ubChannelV,  
        _U08    ubBufferV)
```

Function

Send message from FullCAN buffer

This function will transmit a message from a FullCAN buffer. The buffer has to be configured by a call to [CpCoreAllocBuffer\(\)](#) in advance. In contrast to the function [CpCoreBufferTransmit\(\)](#) no data is copied, which makes this function quite fast.

Parameters

ubChannelV CAN channel of the hardware
ubBufferV Number of message buffer

Return value

Error code defined in the "[cpconst.h](#)" headerfile. If no error occurred, the function will return '[CpErr_OK](#)'.

5.8 CpCoreBufferTransmit

Syntax

```
_U08 CpCoreBufferTransmit(  
    _U08          ubChannelV,  
    _TsCpCanMsg *ptsCanMsgV)
```

Function

Transmit message in a FullCAN buffer

This function will transmit a message from a FullCAN buffer. The buffer has to be configured by a call to [CpCoreAllocBuffer\(\)](#) in advance. Only the data length code and the data fields are copied to the message buffer by this function call. The identifier value is not copied.

Parameters

ubChannelV	CAN channel of the hardware
ptsCanMsgV	Pointer to a CAN message structure, the field v_MsgFlags holds the number of the message buffer inside the CAN controller

Return value

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

5.9 CpCoreCanMode

Syntax

```
_U08 CpCoreCanMode(  
    _U08 ubChannelV,  
    _U08 ubModeV)
```

Function

Set state of CAN controller

This function changes the operating mode of the CAN controller. Possible values for mode are defined in the CP MODE enumeration.

Parameters

ubChannelV CAN channel of the hardware

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_AUTO_BAUD	start auto baudrate detection
CP_MODE_SLEEP	set controller into sleep mode

Return value

Error code defined in the "cpconst.h" headerfile. If no error occurred, the function will return 'CpErr_OK'.

5.10 CpCoreCanState

Syntax

```
_U08 CpCoreCanState(  
    _U08    ubChannelV,  
    _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

ubChannelV CAN channel of the hardware
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 "cpconst.h" headerfile. If no error occurred, the function will return 'CpErr_OK'.

5.11 CpCoreDeAllocBuffer

Syntax

```
_U08 CpCoreDeAllocBuffer(  
    _U08 ubChannelV,  
    _U08 ubBufferV)
```

Function

Free message buffer of FullCAN controller

The function frees the resources of the specified buffer '**ubBufferV**'.

Parameters

ubChannelV CAN channel of the hardware

ubBufferV Number of message buffer

Return value

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

5.12 CpCoreDeInitDriver

Syntax	<code>_U08 CpCoreDeInitDriver(_U08 ubChannelV)</code>
Function	<p>De-Initialize the CAN driver</p> <p>The implementation of this function is dependant on the operating system. Typical tasks might be:</p> <ul style="list-style-type: none">- clear the interrupt vector / routine- close all open paths to the hardware
Parameters	<code>ubChannelV</code> CAN channel of the hardware
Return value	Error code defined in the " cpconst.h " headerfile. If no error occurred, the function will return ' CpErr_OK '.

5.13 CpCoreFilterAll

Syntax

```
_U08 CpCoreFilterAll(  
                                _U08 ubChannelV,  
                                _BIT btEnableV)
```

Function

Set a filter for CAN messages

This function sets up a hardware CAN identifier filter. The filter might be a firmware on an intelligent PC card.

Parameters

ubChannelV CAN channel of the hardware

Return value

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

- **CpErr_NOT_SUPPORTED**
Function is not supported by the driver
- **CpErr_OK**
Function returned without error condition

5.14 CpCoreFilterMsg

Syntax

```
_U08 CpCoreFilterMsg(  
    _U08 ubChannelV,  
    _U16 uwStdIdV,  
    _BIT btEnableV)
```

Function

Hardware Filter for CAN messages

This function sets a filter for CAN messages. If the flag `btEnableV` is set to TRUE, the CAN identifier `'uwStdIdV'` is accepted. If the flag `'btEnableV'` is set to FALSE, the reception of messages with the identifier `'uwStdIdV'` is not possible. Please notice that this function only works for standard messages (2.0A, 11 bit identifier).

Parameters

<code>ubChannelV</code>	CAN channel of the hardware
<code>uwStdIdV</code>	value of the identifier
<code>btEnableV</code>	TRUE for accepting an identifier, FALSE for rejecting an identifier

Return value

Error code defined in the "`cpconst.h`" headerfile. Possible return values are:

- **CpErr_NOT_SUPPORTED**
Function is not supported by the driver
- **CpErr_OK**
Function returned without error condition

5.15 CpCoreHDI

Syntax

```
_U08 CpCoreHDI(_U08 ubChannelV, CpStruct_HDI * hdi)
```

Function

Get Hardware Description Information

This function retrieves information about the used hardware.

Parameters

ubChannelV CAN channel of the hardware

hdi Pointer to the Hardware Description Interface Structure. For more details about CpStruct_HDI, please refer to "Hardware Description Interface" on page 10.

Return value

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

5.16 CpCoreInitDriver

Syntax	<code>_U08 CpCoreInitDriver(_U08 ubChannelV)</code>
Function	<p>Initialize the CAN driver</p> <p>The implementation of this function is dependant on the operating system. Typical tasks might be:</p> <ul style="list-style-type: none">- open a path/file to the hardware- install an interrupt vector
Parameters	<code>ubChannelV</code> CAN channel of the hardware
Return value	<p>Error code defined in the "cpconst.h" headerfile. Possible return values are:</p> <ul style="list-style-type: none">• 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

5.17 CpCoreIntHandler

Syntax

```
void CpCoreIntHandler(void)
```

Function

CAN Interrupt Request (IRQ) Handler

The CAN IRQ Handler is responsible for calling core functions and sending signals (events) to the application whenever an interrupt signal of the CAN controller occurs. This function is **not intended** to be used by the application program.

**Attention !**

Do not call this function by the application program. The implementation of this function is system dependend. For further information please refer to "Interrupt Handler" on page 7 of this document.

Return value

none

5.18 CpCoreMsgReceive

Syntax

```
_U08 CpCoreMsgReceive(_U08 ubChannelV)
```

Function

Read a CAN message from controller

This function reads the receive buffer of a CAN controller and builds a CAN message from that information (refer to “Structure of a CAN message” on page 12). The function is also responsible to generate the time stamp for the CAN message. The complete CAN message is then transferred to the Receive FIFO.

**Attention !**

Do not call this function by the application program. The implementation of this function is system dependend.

Parameters

ubChannelV CAN channel of the hardware

Return value

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

5.19 CpCoreMsgTransmit

Syntax

```
_U08 CpCoreMsgTransmit(_U08 ubChannelV)
```

Function

Transmit a CAN message

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

**Attention !**

Do not call this function by the application program. The implementation of this function is system dependend.

Parameters

channel CAN channel of the hardware

Return value

Error code defined in the "**cpconst.h**" headerfile. If no error occurred, the function will return '**CpErrr_OK**'.

5.20 CpCoreRegRead

Syntax

```
_U08 CpCoreRegRead(  
    _U08    ubChannelV,  
    _U16    uwRegAdrV,  
    _U08 *  pubValueV)
```

Function

Read from a CAN controller register

This function will read a value from a CAN controller register. The first register starts at address 0.

Parameters

ubChannelV CAN channel of the hardware

uwRegAdrV Register address relative to the base address of the CAN controller.

pubValueV Pointer to a variable of type BYTE, which will hold the result of the read process

Return value

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

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

5.21 CpCoreRegWrite

Syntax

```
_U08 CpCoreRegWrite(  
    _U08 ubChannelV,  
    _U16 uwRegAdrV,  
    _U08 ubValueV)
```

Function

Write to a CAN controller register

This function will write a value to a CAN controller register. The first register starts at address 0.

Parameters

ubChannelV	CAN channel of the hardware
uwRegAdrV	Register address relative to the base address of the CAN controller
ubValueV	value to write

Return value

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

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