CC Pilot™ XS

CAN interface description

CC SYSTEMS

Table of Contents

Introduction

Purpose

This document describes the software interface to the CAN Communication Services.

References

-

History

Rev	Date	Author	Remarks
1.0	98-05-25	Göran Nordin	First preliminary version.
1.1	98-06-15	Göran Nordin	Function CanOpenEx added. Added parameter bRtr to CanSend. It is now possible to specify messages you do not want to receive to CanReceive. Type names changed according to coding rules
1.2	98-09-10	Göran Nordin	Functions CanGetDeviceHandle and CanGetLastTimeStamp added.
1.3	98-12-02	Göran Nordin	Parameter pTimeStamp returned from CanGetLastTimeStamp changed.
1.4	2000-05-15	Göran Nordin	Functions CanSendEx, CanReceiveEx and CanGetStatistics added. Configuration added. Windows NT and Windows CE specific restrictions added.
1.5	2000-05-23	Göran Nordin	Description of parameter frameTypeSel to CanReceiveEx corrected. Description of when CanSendEx and CanReceiveEx sets last error to ERROR_NOT_SUPPORTED more detailed.
1.6	2000-08-21	Göran Nordin	Functions CanAddRemoteReply and CanRemoveRemoteReply added.
1.7	2005-03-10	Anders Sipuri	Functions CanAddRemoteReply and CanRemoveRemoteReply removed. Functions CanEnableRemoteFrame, CanDisableRemoteFrame and description for CanEnumRemoteFrame added.
1.8	2005-10-14	Göran Nordin	Description of configuration changed.
1.9	2006-05-14	Göran Nordin	Description of configuration for Windows CE extended.
2.0	2008-04-09	Fredrik Lans	Revision

CAN Communication Services

The CAN Communication Services enables the caller to receive messages from or send messages to CAN interfaces. The caller may receive messages from an interface without "stealing" messages from other users that have opened the same interface.

Summary of interface functions.

CanOpenEx	Opens the CAN interface with Super User privilege.
CanOpen	Opens the CAN interface with Normal User privilege.
CanClose	Closes the CAN interface.
CanSend	Sends a message on the CAN interface.
CanSendEx	Sends a standard or extended frame on the CAN interface.
CanReceive	Receives a message from the CAN interface.
CanReceiveEx	Receives a standard or extended frame from the CAN interface.
CanEnableRemoteFrame	Enables reception of a specified remote frame.
CanDisableRemoteFreme	Disables reception of the specified remote frame.
CanGetStatistics	Returns performance and error counters.
CanGetLastTimeStamp	Returns the time stamp for the last received message.
CanGetDeviceHandle	Returns the handle to the CAN device.
CanEnumRemoteFrame	Enumerates the enabled remote frames.

CanOpenEx

Description

Opens the CAN interface with Super User privilege. Only one user in the system may have Super User privilege for a specific interface. The caller may send messages with any CAN id.

Include files

#include "can.h"

Syntax

```
CanHandle CanOpenEx(

   LPCTSTR pNetName

)
```

Parameters

pNetName	The CAN net name, CAN1-CANn. Either an ANSI or UNICODE string, depending on if _UNICODE and UNICODE is defined or not

Return value

A handle to the opened interface or NULL if operation failed. If operation failed then "GetLastError" can be used to get more information of the error.

If the operation failed because of that another user already has called CanOpenEx then GetLastError will return ERROR_SHARING_VIOLATION.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example

-

CanOpen

Description

Opens the CAN interface with Normal User privilege. There is no restriction on the number of users with Normal User privilege for a specific interface. The caller may only send messages with CAN id's within the configured Normal User range.

Include files

#include "can.h"

Syntax

```
CanHandle CanOpen(

   LPCTSTR pNetName

)
```

Parameters

pNetName	The CAN net name, CAN1-CANn. Either an ANSI or UNICODE string, depending on if _UNICODE and UNICODE is defined or not

Return value

A handle to the opened interface or NULL if operation failed. If operation failed then "GetLastError" can be used to get more information of the error.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example

See example of CanSend or CanReceive.

CanClose

Description

Closes the CAN interface. Disables the caller from receiving messages from or sending messages to the CAN interface.

Include files

#include "can.h"

Syntax

```
BOOL CanClose(

   CanHandle hInterface

)
```

Parameters

hInterface	A handle to an opened interface.

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

Restrictions (Windows NT)

-

Restrictions (Windows CE)

-

Example

See example of CanSend or CanReceive.

CanSend

Description

Sends a message on the CAN interface.

Include files

#include "can.h"

Syntax

```
BOOL CanSend(

   CanHandle hInterface,
   CanMsg *pCanMsg,
   DWORD dataLength,
   BOOL bRtr

)
```

Parameters

hInterface	A handle to an opened interface.
pCanMsg	A pointer to the message to send. The structure of the message is as follows: typedef struct _CanMsg { CanMsgId id; UCHAR data[CAN_MAX_MSG_LENGTH]; } CanMsg; *id* is the CAN message identifier. *data* is the data bytes in the CAN message. *CAN_MAX_MSG_LENGTH* is equal to 8.
dataLength	The number of data bytes to send.
bRtr	Should be set to TRUE if message is to be sent as a remote frame.

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

If the operation failed because the interface was opened with CanOpen and the CAN id of the message is outside the Normal User range then GetLastError will return ERROR_ACCESS_DENIED.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example

This example shows how to send a CAN message on the first interface, i e “CAN1”. Of course opening and closing of an interface should be performed at start-up and termination and not before every sending as in the example.

```
CanHandle hInterface;
CanMsg myCanMsg = {1, {10, 20, 30, 40, 50, 60, 70, 80}};

if ((hInterface = CanOpen(TEXT("CAN1"))) == NULL)
{
  printf(
    "!!ERROR, %lu when calling \"CanOpen\"\n",
    GetLastError());
}

if (!CanSend(hInterface, &myCanMsg, 8 , FALSE))
{
  printf(
    "!!ERROR, %lu when calling \"CanSend\"\n",
    GetLastError());
}

if (!CanClose(hInterface))
{
  printf(
    "!!ERROR, %lu when calling \"CanClose\"\n",
    GetLastError());
}
```

CanSendEx

Description

Sends a standard or extended frame on the CAN interface.

Include files

#include "can.h"

Syntax

```
BOOL CanSendEx(

   CanHandle hInterface,
   CanMsg *pCanMsg,
   DWORD dataLength,
   BOOL bRtr,
   CanFrameType frameType

)
```

Parameters

hInterface	See "CanSend"
pCanMsg	See "CanSend"
dataLength	See "CanSend"
bRtr	See "CanSend"
frameType	Should be set to: CAN_FRAME_STANDARD if message is to be sent as a standard frame

CAN_FRAME_EXTENDED if message is to be sent as an extended frame

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

If the operation failed because the interface was opened with CanOpen and the CAN id of the message is outside the Normal User range then GetLastError will return ERROR_ACCESS_DENIED. If the operation failed because the frame type selected by frameType is not supported by the CAN controller or not configured for the driver then "GetLastError" will return ERROR_NOT_SUPPORTED.

Restrictions (Windows 2000/XP)

Sending extended frames requires that the CAN controller supports extended frames, which is the case for Intel 82527 but not for Philips 82200.

Restrictions (Windows CE)

Same as the "Restrictions (Windows 2000/XP)"

Example

This example shows how to send a standard and extended CAN frame on the first interface, i e “CAN1”. Of course opening and closing of an interface should be performed at start-up and termination and not before every sending as in the example.

```
CanHandle hInterface;
CanMsg myCanMsgStd = {1, {10, 20, 30, 40, 50, 60, 70, 80}};
CanMsg myCanMsgExt = {0x800, {10, 20, 30, 40, 50, 60, 70, 80}};

if (hInterface = CanOpen(TEXT("CAN1")) == NULL)
{
  printf(
    "!!ERROR, %lu when calling \"CanOpen\"\n",
    GetLastError());
}

if (!CanSendEx(
  hInterface,
  &myCanMsgStd,
  8,
  FALSE,
  CAN_FRAME_STANDARD))
{
  printf(
    "!!ERROR, %lu when calling \"CanSendEx\"\n",
    GetLastError());
}

if (!CanSendEx(
  hInterface,
  &myCanMsgExt,
  8,
  FALSE,
  CAN_FRAME_EXTENDED))
{
  printf(
```

```
      "!!ERROR, %lu when calling \"CanSendEx\"\n",
      GetLastError());

  if (!CanClose(hInterface))
  {
    printf(
      "!!ERROR, %lu when calling \"CanClose\"\n",
      GetLastError());
  }
```

CanReceive

Description

Receives a message from the CAN interface.

The caller may receive messages from the interface without "stealing" messages from other users that have called CanOpen.

Include files

#include "can.h"

Syntax

```
BOOL CanReceive(

   CanHandle hInterface,
   CanMsg *pCanMsg,
   LPDWORD pDataLength,
   CanMsgId *pCanMsgSel,
   DWORD milliseconds

)
```

Parameters

hInterface	A handle to an opened interface.
pCanMsg	A pointer to the buffer where the received message should be stored. The structure of the message is as follows: typedef struct _CanMsg { CanMsgId id; UCHAR data[CAN_MAX_MSG_LENGTH]; } CanMsg; *id* is the CAN message identifier. *data* is the data bytes in the CAN message. *CAN_MAX_MSG_LENGTH* is equal to 8.
pDataLength	A pointer to the number of data bytes in the received message.
pCanMsgSel	A pointer to an array specifying which messages that should be received. The first element, (pCanMsgSel[0]), should specify the number of CAN message ID's in the array If pCanMsgSel[0] is positive then any of IDs in the array will be received. If pCanMsgSel[0] is negative then any of IDs that is not in the array will be received. If any message is requested then NULL should be supplied.

milliseconds	Specifies the time-out interval, in milliseconds. The function returns if the interval elapses, even if no messages are received. If milliseconds is zero, the function checks if there are any messages and returns immediately. If milliseconds is INFINITE, the function does not return until a message is received.

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

If the operation failed because of that the time-out interval expired then GetLastError will return ERROR_TIMEOUT.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example 1

This example shows how to receive any message with an infinite time out period on the first interface, i e “CAN1”. Of course opening and closing of an interface should be performed at start-up and termination and not before every reception as in the examples.

```
DWORD dataLength;
DWORD i;
CanHandle hInterface;
CanMsg myCanMsg;
CanTimeStamp timeStamp;

if ((hInterface = CanOpen(TEXT("CAN1"))) == NULL)
{
  printf(
    "!!ERROR, %lu when calling \"CanOpen\"\n",
    GetLastError());
}

if (CanReceive(hInterface,
  &myCanMsg,
  &dataLength,
  NULL,
  INFINITE))
{
  CanGetLastTimeStamp(hInterface, &timeStamp);
  printf("CAN message received\n");
  printf("\tTime stamp (ns): %lu%lu\n",
    timeStamp.high,
    timeStamp.low);
  printf("\tId: %#x\n", myCanMsg.id);
  printf("\tData:");
  for (i = 0; i < dataLength; i++)
  {
    printf(" %#2x", myCanMsg.data[i]);
  }
}
else
{
```

```
    printf(
      "!!ERROR, %lu when calling \"CanReceive\"\n",
      GetLastError());
  }

  if (!CanClose(hInterface))
  {
    printf(
      "!!ERROR, %lu when calling \"CanClose\"\n",
      GetLastError());
  }
```

Example 2

This example shows how to receive messages with id 3 and 5 with a time out period of 500 ms on the first interface, i e “CAN1”.

```
DWORD lastError;
DWORD dataLength;
DWORD i;
CanHandle hInterface;
CanMsg myCanMsg;
CanMsgId CanMsgSel[] = {2, 3, 5};

if ((hInterface = CanOpen(TEXT("CAN1"))) == NULL)
{
  printf(
    "!!ERROR, %lu when calling \"CanOpen\"\n",
    GetLastError());
}

if (CanReceive(
  hInterface,
  &myCanMsg,
  &dataLength,
  CanMsgSel,
  500))
{
  printf("CAN message received\n");
  printf("\tId: %#x\n", myCanMsg.id);
               printf("\tData:");
  for (i = 0; i < dataLength; i++)
  {
    printf(" %#2x", myCanMsg.data[i]);
  }
  printf("\n");
}
else
{
  lastError = GetLastError();

  if (lastError == ERROR_TIMEOUT)
  {
    printf("Time out occurred when receiving CAN messages\n");
  }
  else
  {
    printf(
      "!!ERROR, %lu when calling \"CanReceive\"\n",
      GetLastError());
  }
}

if (!CanClose(hInterface))
{
  printf(
    "!!ERROR, %lu when calling \"CanClose\"\n",
    GetLastError());
}
```

CanReceiveEx

Description

Receives a standard or extended frame from the CAN interface. The caller may receive messages from the interface without "stealing" messages from other users that have called CanOpen.

Include files

#include "can.h"

Syntax

```
BOOL CanReceiveEx(

   CanHandle hInterface,
   CanMsg *pCanMsg,
   LPDWORD pDataLength,
   CanMsgId *pCanMsgSel,
   CanFrameType frameTypeSel,
   CanFrameType *pFrameType,
   DWORD milliseconds

)
```

Parameters

hInterface	See "CanReceive"
pCanMsg	See "CanReceive"
pDataLength	See "CanReceive"
pCanMsgSel	See "CanReceive"
frameTypeSel	Specifies which frame types that should be received. If frameTypeSel is set to CAN_FRAME_STANDARD then only standard frames matching pCanMsgSel is received. If frameTypeSel is set to CAN_FRAME_EXTENDED then only extended frames matching pCanMsgSel is received. If frameTypeSel is set CAN_FRAME_STANDARD | CAN_FRAME_EXTENDED then either standard or extended frames matching pCanMsgSel is received.
pFrameType	A pointer to the frame type of the received message. CAN_FRAME_STANDARD or CAN_FRAME_EXTENDED if a standard or an extended frame is received. CAN_FRAME_REMOTE if the frame is remote.
milliseconds	See "CanReceive".

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

If the operation failed because of that the time-out interval expired then GetLastError will return ERROR_TIMEOUT.
If the operation failed because the frame type selected by frameTypeSel is not supported by the CAN controller or not configured for the driver then "GetLastError" will return ERROR_NOT_SUPPORTED.

Restrictions (Windows 2000/XP)

Receiving extended frames requires that the CAN controller supports extended frames, which is the case for Intel 82527 but not for Philips 82200.

Restrictions (Windows CE)

Same as the "Restrictions (Windows 2000/XP)"

Example

This example shows how to receive any message of either standard or extended frame type with an infinite time out period on the first interface, i e “CAN1”. Of course opening and closing of an interface should be performed at start-up and termination and not before every reception as in the examples.

```
DWORD dataLength;
CanHandle hInterface;
CanMsg myCanMsg;
CanFrameType frameType;

if ((hInterface = CanOpen(TEXT("CAN1"))) == NULL)
{
  printf(
    "!!ERROR, %lu when calling \"CanOpen\"\n",
    GetLastError());
}

if (!CanReceiveEx(
  hInterface,
  &myCanMsg,
  &dataLength,
  NULL,
  CAN_FRAME_STANDARD | CAN_FRAME_EXTENDED,
  &frameType,
  INFINITE))
{
  printf(
    "!!ERROR, %lu when calling \"CanReceiveEx\"\n",
    GetLastError());
}

if (!CanClose(hInterface))
{
  printf(
    "!!ERROR, %lu when calling \"CanClose\"\n",
    GetLastError());
}
```

CanEnableRemoteFrame

Description

Enables reception of the specified remote frame. This function is only needed/supported by hardware that uses 82527 alike message objects. For other hardware this function will fail.

Include files

#include "can.h"

Syntax

```
BOOL CanEnableRemoteFrame(

   CanHandle hInterface,
   CanMsgId id

)
```

Parameters

hInterface	A handle to an opened interface.
id	*id* is the CAN message identifier of the remote frame to enable.

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

If the operation failed because the interface was opened with CanOpen and the CAN id of the message is outside the Normal User range then GetLastError will return ERROR_ACCESS_DENIED.

If the operation failed because there is no more room for more remote replies then GetLastError will return ERROR_NO_MORE_ITEMS, see restrictions for the Intel 82527 Controller. If the operation failed because the hardware doesn't need/support the function then GetLastError will return ERROR_NOT_SUPPORTED.

Restrictions (Windows 2000/XP)

The restrictions for remote frame handling are hardware dependent.

For the Intel 82527 Controller:

It is not possible to enable extended remote frames if reception of both standard and extended frames is enabled, see configuration of Mode.

The number of remote replies is limited to 13.

Restrictions (Windows CE)

Same as the "Restrictions (Windows 2000/XP)".

Example

This example shows how to enable a reply to a remote frame with id 1.

```
CanHandle hInterface;
CanMsgId myRemoteFrameId =  1;

if ((hInterface = CanOpen(TEXT("CAN1"))) == NULL)
{
  printf(
    "!!ERROR, %lu when calling \"CanOpen\"\n",
    GetLastError());
}

if (!CanEnableRemoteFrame(hInterface, myRemoteFrameId))
{
```

```
    printf(
      "!!ERROR, %lu when calling \"CanEnableRemoteFrame\"\n",
      GetLastError());
  }

  .
  .
  .

  if (!CanDisableRemoteFrame(hInterface, myRemoteFrameId))
  {
    printf(
      "!!ERROR, %lu when calling \"CanRemoveRemoteReply\"\n",
      GetLastError());
  }

  if (!CanClose(hInterface))
  {
    printf(
      "!!ERROR, %lu when calling \"CanClose\"\n",
      GetLastError());
  }
```

CanDisableRemoteFrame

Description

Disables reception of the specified remote frame. This function is only needed/supported by hardware that uses 82527 alike message objects. For other hardware this function will fail.

Include files

#include "can.h"

Syntax

```
BOOL CanDisableRemoteFrame(

   CanHandle hInterface,
   CanMsgId id

)
```

Parameters

hInterface	A handle to an opened interface.
id	The CAN message identifier of the remote frame to disable.

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

If the operation failed because the remote frame reply specified by id has never been added, then GetLastError will return ERROR_NOT_FOUND. . If the operation failed because the hardware doesn't need/support the function then GetLastError will return ERROR_NOT_SUPPORTED.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example

See example of CanEnableRemoteFrame.

CanEnumRemoteFrame

Description

Enumerates the enabled remote frames. This function is only needed or supported by hardware that uses 82527 alike message objects. For other hardware this function will fail.

Include files

#include "can.h"

Syntax

```
BOOL CanEnumRemoteFrame(

   CanHandle hInterface,
   ULONG index,
   CanMsgId *pId,
   CanFrameType *pFrameType

)
```

Parameters

hInterface	A handle to the opened interface.
index	Specifies the index of the subkey to retrieve. This parameter should be zero for the first call to the CanEnumRemoteReply function and then, as long as ERROR_SUCCESS is returned, incremented for subsequent calls until ERROR_NO_MORE_ITEMS are returned.
pCanMsgId	A pointer to the message id of the remote frame .
pFrameType	A pointer to the frame type of the remote frame.

Return value

ERROR_SUCCESS or some error code. If operation failed because there are no more remote frame replies then ERROR_NO_MORE_ITEMS is returned.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example

```
CanHandle hInterface;
DWORD lastError;
ULONG i;
CanMsg canMsg;
CanFrameType frameType;

// Other code... (need hInterface for example)

i = 0;
while ((lastError = CanEnumRemoteFrame(
  hInterface,
  i,
  &canMsg.id,
  &frameType)) == ERROR_SUCCESS)
{
  // Doing stuff...

  i++;
}
```

CanGetStatistics

Description

Returns performance and error counters.

Include files

#include "can.h"

Syntax

```
BOOL CanGetStatistics(

   CanHandle hInterface,
   CanStatistics *pStatistics

)
```

Parameters

hInterface	A handle to an opened interface.
pStatistics	A pointer to the performance and error counters. The structure of the performance and error counters is as follows typedef struct _CanStatistics { ULONG rxMsgCntr; ULONG rxDataCntr; ULONG txMsgCntr; ULONG txDataCntr; ULONG hwOvrnCntr;

```
    ULONG busWarnCntr;
    ULONG busOffCntr;
    ULONG appOvrnCntr;
    ULONG rxFifoOvrnCntr;
    ULONG rxFifoMax;
    ULONG txFailCntr;
} CanStatistics;
```

rxMsgCntr is the number CAN messages received so far.
rxDataCntr is the number of data bytes received so far.
txMsgCntr is the number CAN messages transmitted so far.
txDataCntr is the number of data bytes transmitted so far.
hwOvrnCntr is the number of times that the CAN controller has detected a receive overrun.
busWarnCntr is the number of times that the CAN controller has experienced a bus warning condition.
busOffCntr is the number of times that the CAN controller has entered the bus off state.
appOvrnCntr is the maximum number of overruns that have occurred for a client application. If this occurs then the client application needs to be optimized.
rxFifoOvrnCntr is the number of times that a overrun has occurred in the type-ahead receive fifo. If this occurs then the size of the type-ahead receive fifo needs to be increased, see configuration value "RxFifoSize".
rxFifoMax is the maximum number of messages that the type-ahead receive fifo have contained.
txFailCntr is the number of times that a transmit operation has failed.

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

Restrictions (Windows 2000/XP)

Element *txFailCntr* is not implemented

Restrictions (Windows CE)

Element *rxFifoOvrnCntr* and *rxFifoMax* are not implemented.

Example

This example shows how the statistics are retrieved.

```
CanHandle hInterface;
CanStatistics statistics;

if ((hInterface = CanOpen(TEXT("CAN1"))) == NULL)
{
  printf(
    "!!ERROR, %lu when calling \"CanOpen\"\n",
    GetLastError());
}

if (!CanGetStatistics(hInterface, &statistics))
{
  printf(
    "!!ERROR, %lu when calling \"CanGetStatistics\"\n",
```

```
      GetLastError());
  }

  if (!CanClose(hInterface))
  {
    printf(
      "!!ERROR, %lu when calling \"CanClose\"\n",
      GetLastError());
  }
```

CanGetLastTimeStamp

Description

Returns the time stamp for the last sent or received message.

Include files

#include "can.h"

Syntax

```
BOOL CanGetLastTimeStamp(

   CanHandle hInterface,
   CanTimeStamp *pTimeStamp

)
```

Parameters

hInterface	A handle to an opened interface.
pTimeStamp	A pointer to the time stamp. The structure of the time stamp is as follows typedef struct _CanTimeStamp { ULONG low; ULONG high; } CanTimeStamp; *low* is the lower 32 bits of the time stamp in 100-nanoseconds. *high* is the upper 32 bits of the time stamp in 100-nanoseconds.

Return value

TRUE if operation succeeded otherwise FALSE. If operation failed then "GetLastError" can be used to get more information of the error.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example

See example of CanReceive.

CanGetDeviceHandle

Description

Returns the handle to the CAN device. The handle can be used in DeviceIoControl calls

Include files

#include "can.h"

Syntax

```
HANDLE CanGetDeviceHandle(

   CanHandle hInterface

)
```

Parameters

hInterface A handle to an opened interface.

Return value

A handle to the CAN device or INVALID_HANDLE_VALUE if operation failed. If operation failed then "GetLastError" can be used to get more information of the error.

Restrictions (Windows 2000/XP)

-

Restrictions (Windows CE)

-

Example

-

Configuration common

Summary of common configuration values

Name: [1)]	Type:	Description:
BaudRate	String	Can either be set to an explicit baud-rate or to initialization values of the CAN controller's registers. The latter is hardware dependent. For example for an explicit baud-rate of 125 kBit/s the field should be set to "125000". For an 82527 Controller it can also be set to: "DSC=<value>, BTR0=<value>, BTR1=< value >".
BusOffRecoverDelay	DWORD	Should be set to the delay in milliseconds to wait before an attempt should be made to recover from a bus off state. If this field is omitted then the delay is set to 0.
EnableTimeStamp	DWORD	Should be set to "1" if incoming messages should be time stamped. If this field is omitted then no time stamping is made.
Mode	DWORD	Should be set to: "Standard" if standard frames should be sent/received. "Extended" if extended frames should be sent/received. "Standard, Extended" if both standard and extended frames should be sent/received. If this field is omitted then only standard frames are sent/received.
RxFifoSize	DWORD	Should be set to the size of the type-ahead receive fifo. If this field is omitted then the size is set to 8.
TxUserIdRange	String	Should be set to the normal user range. For example if a normal user is allowed to send CAN messages greater than 0, then the field should be set to "1-0xffffffff". If this field is omitted then a normal user is allowed to send any messages.

[1] Values in bold are required.

Configuration (Windows 2000/XP)

The configuration parameters are stored in the registry key:
HKEY_LOCAL_MACHINE\SYSTEM\System\CurrentControlSet\Services\<Driver name>\Parameters\Device<0-n>

Driver name is a string that varies depending on the hardware. Examples are: Can8252CCS, Can82200IpME, Can82527IpME.

The device number, *0-n*, is a number that identifies the interface. Configuration for the interface "CAN1" is stored in *Device0*, configuration for the interface "CAN2" is stored in *Device1* and so on.

Summary of Windows 2000/XP configuration values

Name:	Type:	Description:
RxProcPrio	DWORD	Should be set to: 0x00000040 if the priority for the CAN receive process should be low. 0x00004000 if the priority for the CAN receive process should be below normal. 0x00000020 if the priority for the CAN receive process should be normal. 0x00008000 if the priority for the CAN receive process should be above normal. 0x00000080 if the priority for the CAN receive process should be high. 0x00000100 if the priority for the CAN receive process should be realtime. If this field is omitted then the priority for the CAN receive process is set to normal.

Configuration (Windows CE)

For the ISA/plain driver the configuration parameters are stored in the registry key: HKEY_LOCAL_MACHINE\SYSTEM\System Drivers\BuiltIn\Can for the first CAN device and HKEY_LOCAL_MACHINE\SYSTEM\System Drivers\BuiltIn\Can<2-n> for the following CAN devices.

For the PCI driver the configuration parameters are stored in the registry key: HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PCI\Instance\CANMULTI1\CAN1 for the first CAN device and HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PCI\Instance\CANMULTI1\CAN2 for the second CAN device.

Summary of Windows CE configuration values

Name:[2)]	Type:	Description:
Prefix	String	The prefix of the device name.
Index	DWORD	The device number. If Prefix is “CAN” and index is 1 then the device can be accessed via the name “CAN1:”
RxBufferSize	DWORD	Should be set to the size of the receive buffer. If this field is omitted then the size is set to 256.
TxFailTimeout	DWORD	Should be set to the timeout in milliseconds to wait before a pending transmission should be considered to have failed. If this field is omitted then the timeout is set to 5000.

[2] Values in bold are required