coolgear

CAN Programming Guide

Coolgear Inc. | 01/24/2017
Support: coolgear.com/support

Contents

1. Installation

1.1. DLL Installation

There is no specific DLL installer. The DLL, LIB and Header files are typically copied to an application project directory. Different programming languages and compiler configurations require different locations. Consult your programming environment documentation for the correct locations to place the files contained within the distribution.

2. Application Programming Interface

2.1. Functions

2.1.1. CAN_Open

Summary

Open a channel to the CAN device.

Definition

TCAN_HANDLE CAN_Open(CHAR *ComPort, CHAR *szBitrate, CHAR *acceptance_code, CHAR *acceptance_mask, VOID *flags , DWORD Mode);

Parameters

ComPort	CAN device Serial port name
szBitrate	10 = 10Kbps 20 = 20Kbps 50 = 50Kbps 100 = 100Kbps 125 = 125Kbps 250 = 250Kbps 500 = 500Kbps 800 = 800Kbps 1000 = 1000Kbps aaabbccdd = custom bit rate aaa = BRP bb = TSeg1 cc = TSeg2 dd = SJW
acceptance_code	11-bit (0x000 to 0x7FF) or 29-bit (0x00000000 to 0x1FFFFFFF) code used in filtering specific or ranges of CAN messages. Default is to pass all frames (Acceptance Filter = 0x7FF for standard messages and 0x1FFFFFFF for extended messages).
acceptance_mask	11-bit (0x000 to 0x7FF) or 29-bit (0x00000000 to 0x1FFFFFFF) code used in filtering specific or ranges of CAN messages.

	A value of “0” in a bit location indicates that the bit location ID value is to be ignored when filtering messages. Default is to pass all frames (Acceptance Mask = 0x000 for standard messages and 0x00000000 for extended messages)
flags	CAN_TIMESTAMP_ON = The timestamp function will be enabled by the CAN device. CAN_TIMESTAMP_OFF = The timestamp function will be disabled.
Mode	Normal = the normal operation mode. ListenOnly = the listen only mode. LoopBack = the loop back mode, the CAN device will return messages you send.

Return value

CAN_HANDLE	Handle to the device if successful.
<= 0	Error as noted in 2.2.2 CAN_ERRORS

Example

```
CHAR *ComPort = "COM11";
CHAR *szBitrate = "800";
CHAR *acceptance_code= "1FFFFFFF"; // Default is to pass all frames
CHAR * acceptance_mask= "00000000"; // Default is to pass all frames
VOID *flags=CAN_TIMESTAMP_OFF;
DWORD Mode = Normal;

Handle=CAN_Open(ComPort,szBitrate,acceptance_code,acceptance_mask,flag,Mode);
```

2.1.2. CAN_Close

Summary

Close the CAN channel with Handle

Definition

TCAN_STATUS CAN_Close(TCAN_HANDLE Handle);

Parameters

Handle	the handle of the CAN channel which will be closed

Return value

=1	CAN channel is closed successfully.
<0	Error as noted in 2.2.2 CAN_ERRORS

Example

```
Status = CAN_Close(Handle);
```

2.1.3. CAN_Write

Summary

Write a message to the open channel with handle. Message is contained within the message structure.

Definition

TCAN_STATUS CAN_Write(TCAN_HANDLE Handle, CAN_MSG *Buf);

Parameters

Handle	handle returned by CAN_Open
Buf	transmitted message frame structure

Remarks

```
// Message flags
    #define CAN_FLAGS_STANDARD    (1 << 0)   // Stand CAN ID
    #define CAN_FLAGS_EXTENDED    (1 << 1)   // Extended CAN ID
    #define CAN_FLAGS_REMOTE      (1 << 2)   // Remote frame
// CAN Frame
    typedef struct
    {
        UINT32 Id;
        UINT8 Size;
        UINT8 Data[8];
        UINT8 Flags;
        UINT16 Timestamp;
    } CAN_MSG;
```

Return value

=1	CAN channel is work successfully.
<0	Error as noted in 2.2.2 CAN_ERRORS

Example

```
CAN_MSG *msg[1];
msg[0].Flags = CAN_FLAGS_STANDARD;

msg[0].Id = 0x111;
msg[0]. Size = 5
```

```
msg[0].Data[0] = 0x11;
msg[0].Data[1] = 0x11;
msg[0].Data[2] = 0x11;
msg[0].Data[3] = 0x11;
msg[0].Data[4] = 0x11;
msg[0].Data[5] = 0x11;
msg[0].Data[6] = 0x11;
msg[0].Data[7] = 0x11;
Status=CAN_Write(Handle, msg);
```

2.1.4. CAN_Read

Summary

Read a message from the open channel with handle. Message is contained within the message structure.

Definition

TCAN_STATUS CAN_Read(TCAN_HANDLE Handle, CAN_MSG *Buf);

Parameters

Handle	handle returned by CAN_Open
buf	received message frame structure

Remarks

```
// Message flags
    #define CAN_FLAGS_STANDARD    (1 << 0)   // Stand CAN ID
    #define CAN_FLAGS_EXTENDED    (1 << 1)   // Extended CAN ID
    #define CAN_FLAGS_REMOTE      (1 << 2)   // Remote frame
// CAN Frame
    typedef struct
    {
        UINT32 Id;
        UINT8 Size;
        UINT8 Data[8];
        UINT8 Flags;
        UINT16 Timestamp;
    } CAN_MSG;
```

Return value

=1	CAN channel is work successfully.
<0	Error as noted in 2.2.2 CAN_ERRORS

Example

```
CAN_MSG msg;

Status=CAN_Read( Handle, &msg );
```

2.1.5. CAN_Status

Summary

Retrieve adaptor status for channel with handle.

Definition

TCAN_STATUS CAN_Status(TCAN_HANDLE Handle);

Parameters

Handle	handle returned by CAN_Open

Return value

>0	Status value with bit values in 2.2.3 CAN_STATUS

Example

```
Status = CAN_ Status ( Handle );
```

2.1.6. CAN_Version

Summary

Retrieve device firmware version with handle.

Definition

TCAN_STATUS CAN_Version(TCAN_HANDLE Handle, CHAR *Buf);

Parameters

Handle	handle returned by CAN_Open
Buf	device firmware version

Return value

=1	CAN channel is work successfully.
<0	Error as noted in 2.2.2 CAN_ERRORS

Example

```
CHAR Versioninfo[10];

Status=CAN_Version( Handle, Versioninfo );
```

2.1.7. CAN_Flush

Summary

Clears the buffers for device with handle.

Definition

TCAN_STATUS CAN_Flush(TCAN_HANDLE Handle);

Parameters

Handle	handle returned by CAN_Open

Return value

=1	CAN channel is work successfully.
<0	Error as noted in 2.2.2 CAN_ERRORS

Example

```
Status = CAN_Flush( Handle );
```

2.2. Types and Structures

2.2.1. CAN_HANDLE

```
typedef int TCAN_HANDLE;
```

2.2.2. CAN_ERRORS

```
CAN_ERR_OK
CAN_ERR_ERR
CAN_ERR_OPEN_CHANNEL
CAN_ERR_PARAMETER
CAN_ERR_NOT_OPEN
CAN_ERR_READ_NO_MSG
```

2.2.3. CAN_STATUS

```
typedef int TCAN_STATUS;

CAN_STATUS_BOFF
CAN_STATUS_EWARN
CAN_STATUS_EPASS
CAN_STATUS_LEC_NO
CAN_STATUS_LEC_STUFF_ERR
CAN_STATUS_LEC_FORM_ERR
CAN_STATUS_LEC_ACK_ERR
CAN_STATUS_LEC_BIT1_ERR
CAN_STATUS_LEC_BIT0_ERR
CAN_STATUS_LEC_CRC_ERR
```

2.2.4. CAN_MSG

```
typedef struct
{
	UINT32 Id;
	UINT8 Size;
	UINT8 Data[8];
	UINT8 Flags;
	UINT16 Timestamp;
} CAN_MSG;
```

2.3. Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include "CAN_API.h"

int main() {
   TCAN_HANDLE Handle;
   TCAN_STATUS Status;
   CHAR *ComPort = "COM23";
   CHAR *szBitrate = "800";
   CHAR *acceptance_code = "1FFFFFFF";
   CHAR *acceptance_mask = "00000000";
   VOID *flags = CAN_TIMESTAMP_OFF;
   DWORD Mode = LoopBack;

   char version[10];
   CAN_MSG SendMSG;
   CAN_MSG RecvMSG;
   Handle = -1;
   Status = 0;

   SendMSG.Flags = CAN_FLAGS_EXTENDED;
   SendMSG.Id = 0x12345678;
   SendMSG.Size = 8;
   SendMSG.Data[0] = 0x11;
   SendMSG.Data[1] = 0x22;
   SendMSG.Data[2] = 0x33;
   SendMSG.Data[3] = 0x44;
   SendMSG.Data[4] = 0x55;
   SendMSG.Data[5] = 0x66;
   SendMSG.Data[6] = 0x77;
   SendMSG.Data[7] = 0x88;

   Handle = CAN_Open ( ComPort, szBitrate, acceptance_code, acceptance_mask, flags,
Mode );

   printf ( "handle= %d\n", Handle );
   if ( Handle < 0 ) {
      return 0;
   }

   memset ( version, 0, sizeof ( char ) * 10 );
   Status = CAN_Flush ( Handle );
   Status = CAN_Version ( Handle, version );

   if ( Status == CAN_ERR_OK ) {
      printf ( "Version : %s\n", version );
   }

   Status = CAN_Write ( Handle, &SendMSG );
   if ( Status == CAN_ERR_OK ) {
      printf ( "Write Success\n" );
   }

   while ( 1 ) {
      Status = CAN_Read ( Handle, &RecvMSG );
      if ( Status == CAN_ERR_OK ) {
         printf ( "Read ID=0x%X, Type=%s, DLC=%d, FrameType=%s, Data=",
         RecvMSG.Id,( RecvMSG.Flags & CAN_FLAGS_STANDARD ) ? "STD" : "EXT",
         RecvMSG.Size,( RecvMSG.Flags & CAN_FLAGS_REMOTE ) ? "REMOTE" : "DATA"  );

            for ( int i = 0; i < RecvMSG.Size; i++ ) {
            printf ( "%X,", RecvMSG.Data[i] );
         }
         break;
```

```
        }
    }

    Status = CAN_Close ( Handle );
    printf ( "Test finish\n" );
    return 0;
}
```