Applieddata.net
Embedded Computer Systems

ADS Windows CE PeliCAN Driver

SJA1000T CAN Controller

Specification Version 1.4

Introduction

The ADS Windows CE PeliCAN driver provides a standard stream interface API to the Philips SJA1000T CAN controller. This document gives a specification for this API and describes the basic operation of the driver.

Theory of Operation

This section describes how the ADS driver interacts with the Philips SJA1000T CAN Controller. Details about the settings and APIs are listed in the following sections, ***Using the Driver*** and ***API Reference***.

PeliCAN Mode

The SJA1000T controller can operate in two distinct modes: BasicCAN mode and PeliCAN mode. BasicCAN supports only standard, 11-bit message identifiers. PeliCAN mode provides the BasicCAN functionality, but additionally supports extended 29-bit CAN identifiers and is compliant with the CAN2.0B standard. The ADS Windows CE PeliCAN driver operates the SJA1000T in PeliCAN mode only.

Receiving CAN Messages

When the CAN controller receives a message, the ADS PeliCAN driver fetches it from the chip and adds it to a message queue (5000 message maximum) resident in the driver. The driver then notifies applications that a new message has arrived by pulsing the "message event" and setting the "data ready event" (The names of these events are configurable in the registry, see ***Using the Driver: Registry Settings***. Also, further information about the message and data ready events is provided in ***Using the Driver: Driver Events).*** Applications can use the **`ReadFile`** function to retrieve CAN messages from the driver receive queue.

The SJA1000T provides an acceptance filtering feature. Acceptance filtering allows only messages with identification fields that meet the filter requirements to be received by the CAN controller. The ADS PeliCAN driver API allows the acceptance filter to be set via the **`DeviceIoControl`** function (see ***API Reference: I/O Controls***). By default the driver accepts all messages it reads from the CAN bus.

Sending CAN Messages

The CAN driver packages and sends CAN messages via the **`WriteFile`** function. The call is blocking only if another message is in the process of being sent.

If the CAN bus is disconnected, or another error condition exists that prevents sending the data, the **`WriteFile`** function returns FALSE and the cause of the error can be obtained using the **`GetLastError`** function (See ***API Reference: Error Codes***).

Driver Priority

You can set the driver priority in the CE registry (see ***Using the Driver: Registry Settings***). Driver priority can become important in systems that have multiple communication or I/O channels in simultaneous operation. Set the driver priority based on your application's architecture. Please be advised that modifying the default priority of the driver can have a negative impact on system performance. Systems should be thoroughly tested with the new priority setting.

Using the Driver

Driver Name

The CAN driver is referenced with the filename **`CAN1:`**. If an ADS product is built with additional CAN controllers, they are referenced as **`CAN2:`**, **`CAN3:`**, and so forth.

Files

The following files are important for developers of CAN applications:

PeliCanSja1000.dll	*CAN driver for 29-bit PeliCAN mode*
CANapp.h	*Header file for CAN driver constants*
ADSerror.h	*Header file for ADS error codes*

Registry Settings

Windows CE uses the following system registry keys to configure the driver:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN]
       "Dll"="PeliCanSja1000.dll"    ; use ADS PeliCAN SJA1000T driver
       "Priority256"=dword:62        ; CAN driver priority = 0x62
       "MessageEvent"="EV_APP0"      ; pulsed when message is received
       "ErrorEvent"="EV_ERR0"        ; pulsed when a CAN error occurs
       "OverrunEvent"="EV_OVRERR0"   ; pulsed when an overrun occurs
       "DataReadyEvent"="EV_DATA_RDY0" ; set when RX queue is not empty
```

These settings can be modified by changing the ADSLOAD.REG file or by changing the registry and persisting it using either ADS tools or hive-based images.

Driver Events

The CAN driver uses events to notify applications that messages are available, or that an error has occurred. The names of the events can be modified in the CE registry. The default name and triggering condition of each event is listed in the table below (***Table 1: PeliCAN Driver Events)***.

Table 1: PeliCAN Driver Events		
Registry Entry	***Default Event Name****	***Condition***
"MessageEvent"	*"EV_APP0"*	Pulsed when each new CAN message arrives in the queue.
"ErrorEvent"	*"EV_ERR0"*	Pulsed when the SJA1000T input buffer has overflowed (DOI bit is set in the SJA1000T interrupt register). You may need to increase the CAN thread priority so the driver can service incoming messages more quickly.
"OverrunEvent"	*"EV_OVRERR0"*	Pulsed when EI (Error Warning Interrupt) or BEI (Bus Error Interrupt) bits in the SJA1000T interrupt register have been set.
"DataReadyEvent"	*"EV_DATA_RDY0"*	Set when there are messages in the receive queue. Reset when the queue is empty.
**Default event name for first CAN controller (CAN1:). For additional controllers the default event names are appended with 1, 2,… etc.*		

Note that the message, error, and overrun events are all *pulsed* by the driver, while the data ready event is set and reset. A pulsed event is automatically unset when all threads waiting on it have been signaled. This means that a thread must be waiting on the event at the moment it is pulsed in order to receive it, otherwise it will be missed. For example, if a thread begins waiting for the message event after a message has been received, the event for that message will be missed. The PeliCAN driver also provides a data ready event that is set when messages are ready in the receive queue, and reset when the queue is empty. The decision to use either the message event or data ready event to detect new CAN messages should depend on the application architecture.

API Reference

Windows CE applications access the ADS PeliCAN driver using the stream interface functions (**`CreateFile`**, **`ReadFile`**, **`WriteFile`** and **`DeviceIoControl`**). Usage of these functions with examples is provided here.

HANDLE **`CreateFile(`** *lpFileName*, *dwDesiredAccess*, *dwShareMode*, *lpSecurityAttributes*, *dwCreationDispostion*, *dwFlagsAndAttributes*, *hTemplateFile*)

Opens the CAN port and returns a handle to access it with. Returns NULL if an error occurred. **`GetLastError`** can be used to retrieve a specific error code. ADS error codes are listed in the file ADSerror.h.

Example:

```
HANDLE hCanPort;
hCanPort = CreateFile( _T("CAN1:"),
                       GENERIC_WRITE | GENERIC_READ,
                       0,
                       NULL,
                       OPEN_EXISTING,
                       FILE_ATTRIBUTE_NORMAL,
                       NULL);
```

BOOL **`ReadFile(`** *hFile*, *lpBuffer*, *nNumberOfBytesToRead*, *lpNumberOfBytesRead*, *lpOverlapped* **)**

Reads one or more messages from the CAN receive queue and stores them in the CAN_MSG structure array referenced by *lpBuffer*. The number of messages to read is determined by the *nNumberOfBytesToRead* parameter. *nNumberOfBytesToRead* should be the number of messages to read, multiplied by `sizeof(CAN_MSG)`. **ReadFile** returns TRUE if successful or FALSE if there was an error. **ReadFile** will return successfully even if there are fewer messages in the queue than were requested. When **ReadFile** returns successfully, *lpNumberOfBytesRead* will reference a value equal to the number of messages actually read multiplied by `sizeof(CAN_MSG)`.

Example:

```
CAN_MSG RxCanMsg[10];
DWORD dwBytesRead;

// Read one CAN message into RxCanMsg[0]
ReadFile(hCanPort, &RxCanMsg[0], sizeof(CAN_MSG), &dwBytesRead, NULL);
printf("Read %d CAN messages.\r\n", dwBytesRead / sizeof(CAN_MSG));

// Read nine CAN messages into RxCanMsg[1]-RxCanMsg[9]
ReadFile(hCanPort, &RxCanMsg[1], 9 * sizeof(CAN_MSG), &dwBytesRead, NULL);
printf("Read %d CAN messages.\r\n", dwBytesRead / sizeof(CAN_MSG));
```

BOOL **`WriteFile(`** *hFile*, *lpBuffer*, *nNumberOfBytesToWrite*, *lpNumberOfBytesWritten*, *lpOverlapped*)

Writes one or more CAN messages from the CAN_MSG structure array referenced by *lpBuffer* to the CAN bus. The number of messages to write is determined by the *nNumberOfBytesToWrite* parameter. *nNumberOfBytesToWrite* should be the number of messages to write, multiplied by `sizeof(CAN_MSG)`. Returns TRUE if successful or FALSE if there was an error. If the write completes successfully, *lpNumberOfBytesRead* will reference a value equal to the number of messages

written multiplied by `sizeof(CAN_MSG)`. If the SJA1000T is in the BUS OFF state, **WriteFile** will fail and **GetLastError** will return the CAN_ERROR_BUS_OFF error code.

Example:

```
CAN_MSG TxCanMsg[10];
DWORD dwBytesWritten;

... (initialize TxCanMsg array) ...

// Write one CAN message (TxCanMsg[0]) to the bus.
WriteFile(hCanPort, &TxCanMsg[0], sizeof(CAN_MSG), &dwBytesWritten, NULL);
printf("Wrote %d CAN messages.\r\n", dwBytesWritten / sizeof(CAN_MSG));

// Write nine CAN messages (TxCanMsg[1] - TxCanMsg[9]) to the bus.
WriteFile(hCanPort, &TxCanMsg[1], 9 * sizeof(CAN_MSG), &dwBytesWritten, NULL);
printf("Wrote %d CAN messages.\r\n", dwBytesWritten / sizeof(CAN_MSG));
```

BOOL **DeviceIoControl(** *hDevice*, *dwIoControlCode*, *lpInBuffer*, *nInBufferSize*, *lpOutBuffer*, *nOutBufferSize*, *lpBytesReturned*, *lpOverlapped* **)**

Provides an additional set of CAN functions. Returns TRUE if successful or FALSE if there is an error. The available I/O control functions are listed in the next section. Input parameters are passed in through *lpInBuffer*, and output data is returned via *lpOutBuffer*. *nInBufferSize* and *nOutBufferSize* are required to specify the size of the input and output buffers. The *lpBytesReturned* parameter is not used.

Example:

```
UINT nNumMessages = 0;

DeviceIoControl( hCanPort,
                 IOCTL_CAN_GET_NUM_MSGS,
                 NULL,
                 0,
                 &nNumMessages,
                 sizeof(UINT),
                 NULL,
                 NULL);
```

BOOL **CloseHandle(** *hObject* **)**

Closes the CAN port referenced by *hObject*. Returns TRUE if successful or FALSE if there is an error.

Example:

```
CloseHandle(hCanPort);
```

DWORD **Seek(** *hOpenContext*, *Amount*, *Type* **)**

Calls to the **Seek** function have no effect and always return 0xFFFFFFFF.

I/O Controls

The I/O control codes listed below provide access to additional functionality in the ADS PeliCAN driver. Usage of **`DeviceIoControl`** was described in the previous section.

IOCTL_CAN_SET_ACCEPTANCE_FILTER (0x01): Sets the acceptance filter. The *`lpInBuffer`* parameter to **`DeviceIoControl`** must reference the CAN_MSG_FILTER structure that contains the new filter settings. Set the *`nInBufferSize`* parameter to `sizeof(CAN_MSG_FILTER)`.

IOCTL_CAN_GET_ACCEPTANCE_FILTER (0x02): Retrieves the current acceptance filter settings. The *`lpOutBuffer`* parameter to **`DeviceIoControl`** must reference the CAN_MSG_FILTER structure that will receive the filter settings. Set the *`nOutBufferSize`* parameter to `sizeof(CAN_MSG_FILTER)`.

IOCTL_CAN_SET_BAUDRATE (0x03): Sets the CAN baudrate to the closest possible rate to the value provided. IOCTL_CAN_GET_BAUDRATE can be used to read the actual value from the CAN chip. The *`lpInBuffer`* parameter to **`DeviceIoControl`** must reference a ULONG typed variable that contains the new baudrate in units of kilobits per second. Set the *`nInBufferSize`* parameter to `sizeof(ULONG)`.

IOCTL_CAN_GET_BAUDRATE (0x04): Returns the current CAN baudrate as read from the CAN chip. The *`lpOutBuffer`* parameter to **`DeviceIoControl`** must reference the ULONG typed variable that will receive the current baudrate. Set the *`nOutBufferSize`* parameter to `sizeof(ULONG)`.

IOCTL_RESET_CHIP (0x05): Resets the SJA1000T CAN controller. Does not require any I/O parameters.

IOCTL_CAN_GET_STATUS_REG (0x07): Returns the current state of the SJA1000T status register (SJASR). The *`lpOutBuffer`* parameter to **`DeviceIoControl`** must reference the BYTE typed variable that will receive the status register state. Set the *`nOutBufferSize`* parameter to `sizeof(BYTE)`.

IOCTL_CAN_CLEAR_QUEUE (0x08): Flushes the driver's receive queue. Does not require any I/O parameters.

IOCTL_CAN_GET_NUM_MSGS (0x09): Returns the number of CAN messages currently stored in the driver receive queue. The *`lpOutBuffer`* parameter to **`DeviceIoControl`** must reference the UINT typed variable that will receive the number of messages. The *`nOutBufferSize`* parameter must be set to `sizeof(UINT)`.

IOCTL_CAN_GET_SAMPLE_POINT (0x0A): Returns the current sample point as a percentage (i.e. a returned value of 75 should be interpreted as 75%). The *lpOutBuffer* parameter to **DeviceIoControl** must reference the ULONG typed variable that will receive the sample point value. Set the *nOutBufferSize* parameter to sizeof(ULONG).

IOCTL_CAN_SET_SAMPLE_POINT (0x0B): Sets the CAN sample point as the close as possible to the value provided. The input value will be interpreted as a percentage (i.e. an integer value of 75 will result in a sample point at 75%) and must be within the range of 0 to 100. IOCTL_CAN_GET_SAMPLE_POINT can be used to read the actual value from the CAN chip. The *lpInBuffer* parameter to **DeviceIoControl** must reference a ULONG typed variable that contains the new sample point. Set the *nInBufferSize* parameter to sizeof(ULONG).

IOCTL_GET_DRIVER_VERSION (0xA0): Retrieves the specification version that the current driver adheres to in null-terminated string format (i.e. "1.2\0"). The `lpOutBuffer` parameter to **`DeviceIoControl`** must be a pointer to a wchar_t buffer of sufficient size to accept the string or a failure will occur.

Error Codes

If a driver function call fails, calling GetLastError may return one of the following error codes defined in *ADSError.h*:

CAN_ERROR_OUTPUT_BUFFER_TOO_SMALL

The output buffer provided is insufficient to contain the data required.

CAN_ERROR_INPUT_BUFFER_WRONG_SIZE

The input buffer provided does not match the size expected.

CAN_ERROR_INVALID_HANDLE

The CAN port handle is invalid.

CAN_ERROR_CANNOT_OPEN_DEVICE

The CAN port cannot be opened.

CAN_ERROR_CANNOT_ALLOC_MEMORY

There was an error while attempting to allocate memory.

CAN_ERROR_BUS_OFF

The SJA1000T CAN controller is currently in the BUS_OFF state. When this error occurs call DeviceIoControl with the **IOCTL_RESET_CHIP** constant.

CAN_ERROR_INPUT_OUT_OF_RANGE

The value of the input provided was outside the valid range.

CAN Application Header File: CANApp.h

```
// CANapp.h
// 720020-11851
//
// Applied Data Systems
//
// Description
// -----------
// This header file is to be used by applications interfacing
// with the ADS PeliCAN v1.1 Driver.  It contains all necessary device
// IOCTL, flag, and message definitions.

// CAN message structure
typedef struct _CAN_MSg
{
        SHORT length;
        ULONG  id;
        SHORT  flags;
        union
        {
                BYTE           data[8];
                WORD           wData[4];
                DWORD          dwData[2];
                LONGLONG       lData;
        };
}CAN_MSG;

// Definitions to use for CAN_MSG flags
#define MSG_RTR (1<<0)        // Remote Transmission Request flag
#define MSG_EXT (1<<1)        // Extended identifier format flag

/* Acceptance filter message structure */
typedef struct __CAN_MSG_FILTER
{
  BOOL mode  ;         // set to 0 for dual filter mode, or 1 for single
  BYTE code0 ;
  BYTE code1 ;
  BYTE code2 ;
  BYTE code3 ;
  BYTE mask0 ;
  BYTE mask1 ;
  BYTE mask2 ;
  BYTE mask3 ;
} CAN_MSG_FILTER, *PCAN_MSG_FILTER;


#define IOCTL_CAN_SET_ACCEPTANCE_FILTER      0x01
#define IOCTL_CAN_GET_ACCEPTANCE_FILTER      0x02
#define IOCTL_CAN_SET_BAUDRATE               0x03
#define IOCTL_CAN_GET_BAUDRATE               0x04
#define IOCTL_RESET_CHIP                     0x05
#define IOCTL_SEND_COMMAND                   0x06
#define IOCTL_CAN_GET_STATUS_REG             0x07
#define IOCTL_CAN_CLEAR_QUEUE                0x08
#define IOCTL_CAN_GET_NUM_MSGS               0x09
#define IOCTL_CAN_GET_SAMPLE_POINT           0x0A
#define IOCTL_CAN_SET_SAMPLE_POINT           0x0B
#define IOCTL_GET_DRIVER_VERSION             0xA0
```

Implementation Matrix

The following table illustrates the ADS run-time image in which each version of the CAN driver was included. For example, version 1.0 of the driver was first included in AGX image 4.20.09.

ADS Product	CAN Specification					
	v 0.1	v1.0	v1.1	v1.2	v1.3	v1.4
AGX		4.20.09		4.20.10		
VGX	4.20.12					4.20.26
GCX			4.20.05		4.20.07	4.20.09

Document History

The following list summarizes the changes made between releases of this document.

REV	DESCRIPTION	BY
0	First version of document template	9/16/04 ak
1	Initial release.	11/15/04 jc
2	• Changed CAN_MSG struct format in CANapp.h • Updated Implementation Matrix	11/24/04 jc
3	• Added Specification History section • Added documentation for IOCTL_GET_DRIVER_VERSION • Updated Implementation Matrix • Minor formatting and text changes	11/30/04 ct
4	• Minor wording changes, fixed header and footer	1/14/05 ct
5	• Modified description of reads and writes to include multiple message support. • Added documentation for DataReadyEvent, with more detailed description of how the driver sets events. • Added documentation for IOCTL_GET_SAMPLE_POINT and IOCTL_SET_SAMPLE_POINT. • Modified description of IOCTL_SET_BAUDRATE and IOCTL_GET_BAUDRATE to point out that the actual baudrate set is a best fit for the value provided. • Added documentation for CAN_ERROR_INPUT_OUT _OF_RANGE error code.	5/17/05 jc

Specification History

The following list summarizes the changes made between versions of the specification.

REV	DESCRIPTION	BY
1.0	Initial release.	11/15/04 jc
1.1	Changed CAN_MSG struct format	11/24/04 jc
1.2	• Added IOCTL_GET_DRIVER_VERSION • IOCTL_CAN_READ_ACCEPTANCE_FILTER changed to IOCTL_CAN_GET_ACCEPTANCE_FILTER	11/30/04 ctacke
1.3	• ReadFile and WriteFile support multiple messages • Added DataReadyEvent	3/11/05 jc
1.4	• Added IOCTL_CAN_GET_SAMPLE_POINT and IOCTL_CAN_SET_SAMPLE_POINT • Modified IOCTL_CAN_GET_BAUDRATE to read the actual baudrate from the CAN chip. • Added CAN_ERROR_INPUT_OUT_OF_RANGE error code.	5/16/05 jc