Applieddata.net
Embedded Computer Systems

ADS Windows CE CAN Driver

for the SJA1000T CAN Controller

Specification Version 1.5

ADS document # 110025-10036

10260 Old Columbia Road, Columbia, MD 21046 USA

www.applieddata.net

301.490.4007

Table of Contents

Document History

The following list summarizes the changes made between releases of this document. Changes to the driver specification are listed in the following section.

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
6	• Reformatted document and document styles • Added table of contents • Added Background and Features sections • Added sections for re-entrancy, defaults, known issues • Added section about Sleep/Wake behavior • Rewrote Driver Priority section • Added documentation for IOCTL_CAN_FLUSH_MSGS • Added documentation for events EV_CAN_DATAQ_FULL and EV_CAN_DATAQ_THRESHOLD • Added overview diagram • Added MSG_EXT message detail • Removed Implementation Matrix	10/2/06 jj

Driver Revision 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 ct
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 baud rate from the CAN controller. • Added CAN_ERROR_INPUT_OUT_OF_RANGE error code.	5/16/05 jc
1.5	• Added IOCTL_CAN_FLUSH_MSGS • Added events EV_CAN_DATAQ_FULL and EV_CAN_DATAQ_THRESHOLD	9/21/06 cb

Known Issues

- If a CAN Controller hardware error is encountered during a CreateFile() operation, the driver may hang and GetLastError() will not show the error.
- If CloseHandle() is given an invalid handle, it will return FALSE without GetLastError() showing the error.
- If KernelIoControl() is given an invalid handle, it will return FALSE without GetLastError() showing the error.

1 Introduction

This document describes the Windows CE driver provided by Applied Data Systems for interfacing to the Philips SJA1000T CAN controller found on many ADS products. This document specifies and describes basic operation of the driver.

1.1 Features

The ADS CAN driver features the following functionality:

- Automatic buffering of incoming messages
- Transfer single or multiple messages with each call
- Configurable baud rate and sample point
- Hardware acceptance filter
- Windows CE events that indicate messages received, buffer status and error conditions
- Ability to read status register and reset CAN controller
- Get status of receive queue
- Discard messages from receive queue

1.2 Background

Controller Automation Network, known as CAN or CAN bus, is a communications standard initially developed for the automotive industry.

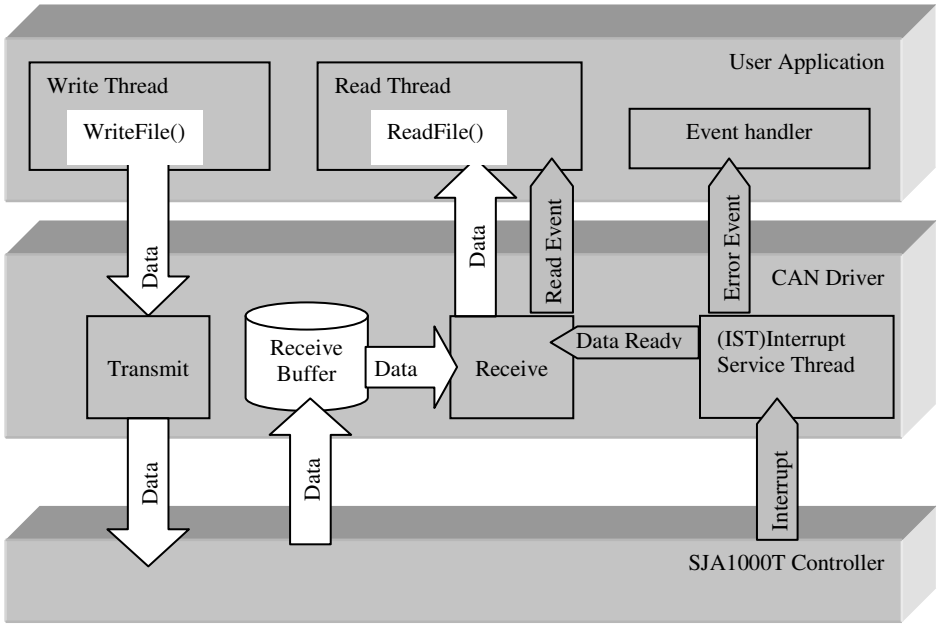
A "message" is defined as a fixed 8 bytes of data with another 8 bytes of header information. This definition is in the CANApp.h file under the name CAN_MSG.

2 Driver Theory of Operation

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

The ADS CAN driver provides a standard Windows CE stream interface driver to access CAN functionality on ADS products.

The following diagram illustrates the relationship between the application, driver and controller. For a write operation, the application calls `WriteFile()` which directs the CAN driver to send the output buffer to the hardware transmit circuitry. For read operations, the Interrupt Service Thread of the CAN driver receives interrupts from the hardware that indicate there is a message to read. The CAN driver reads and adds this message to the receive buffer, then generates a read event to the application. The application calls `ReadFile()` to read the message from the driver's receive buffer. Any error interrupts from the CAN controller are sent via the Interrupt Service Thread to the application as an error event.



2.1 PeliCAN Mode

The SJA1000T controller supports two distinct modes of operation: BasicCAN and PeliCAN. BasicCAN mode supports standard, 11-bit message identifiers. PeliCAN mode provides all BasicCAN functionality and further supports extended, 29-bit CAN identifiers and full CAN2.0B compliance. To transmit 11-bit message identifiers, clear the `MSG_EXT` bit as defined in the `CANApp.h` header file (see **Creating a CAN Application:5.1 Header Files**). The ADS Windows CE CAN driver operates the SJA1000T in PeliCAN mode only.

2.2 Receiving CAN Messages

A CAN message is a fixed length of 8 bytes and 8 bytes of formatting. When the CAN controller receives a message from the remote system, the ADS CAN driver fetches it from the chip and adds it to a message queue 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." Applications then use the `ReadFile()` function to retrieve CAN messages from the driver receive queue. The receive queue has a 5000 message maximum.

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. Set the acceptance filter with the `DeviceIoControl()` function (see ***Driver Reference:4.6 I/O Controls***). The default message filter accepts all messages from the CAN bus.

2.3 Sending CAN Messages

The CAN driver packages and sends CAN messages with 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 ***Driver Reference:4.8 Error Codes***).

2.4 Driver Priority

Driver priority is important in multi-threaded applications, especially those with multiple communication or I/O channels in simultaneous operation. Set the driver priority relatively high to ensure that the CAN controller is serviced promptly and its buffer does not overflow. The priority of the application thread that reads from the CAN buffer should also be high, but set slightly lower than that of the driver. Set the driver priority in the CE registry (see ***Driver Reference:4.8 Error Codes***).

Note that heavy CAN bus traffic can slow down performance of a system. Use message filtering where possible and pay careful attention to the architecture of message processing loops. Test systems thoroughly after any changes in message priority.

2.5 Sleep/Wake Behavior

The ADS CAN driver supports Windows CE Sleep mode. When the system goes to sleep, the driver turns off the CAN controller and transceiver. Upon wakeup the driver powers up the CAN circuits and restores all driver settings and filters.

If a transmission is in progress when the system goes to sleep, the transmission will be interrupted but no data will be lost. Transmission resumes when the system wakes up.

Data stored in the receive buffer remains intact during sleep. The driver does not receive any messages and the CAN controller does not respond to other devices while the system is in Sleep mode. Sleep mode must be entered with care because data may be lost if the system enters sleep mode during a receive operation. The system will not wake upon data receipt.

2.6 Reentrancy and Multi-threaded Behavior

The CAN driver can be opened once. For multi-threaded operation, open the CAN driver using `createfile()`, then pass the new handle to message processing threads.

All driver operations are atomic, so the driver can support different threads accessing the driver asynchronously.

3 Application Architectures and Driver Usage

This section describes some common application architectures that work well with the CAN driver.

3.1 Structure of a CAN Application

A typical application that uses the CAN driver will consists of a main processing loop (or equivalent) with a spawned thread to process incoming messages.

The following pseudo code illustrates such an application:

```
Main()
    InitializeCanDriver()
    spawn MessageProcessingThread() for incoming messages
    ...
    transmit CAN messages as needed
    ...
    stop processing thread
    close CAN driver

InitializeCanDriver()
    Open CAN driver
    Set sample point
    Set baud rate
    ...

MessageProcessingThread()
    while(application is running)
        wait for CAN receive event [blocking call]
            while (messages in queue)
                read message
                process message
```

3.2 Adding Error Handling

The CAN driver generates several events to indicate error conditions. The following pseudo code is for an error processing thread.

```
CanErrorProcessingThread()
    while(TRUE)
        wait for CAN error event
            if (error is buffer full)
                increase MessageProcessingThread() priority
                or delete some number of older messages in the queue
            if (error is CAN controller)
                reset CAN controller
                indicate error in service log and/or to user
```

4 Driver Reference

This section provides the details needed to write C/C++ application code to access the driver.

4.1 Driver Name

The CAN driver is referenced with the driver name `CAN1:`. Additional CAN controllers available on some ADS products are referenced as `CAN2:`, `CAN3:`, and so forth.

4.2 Default Settings

The CAN driver starts up with the following default settings:

Baud rate: 0 kbps

Sample point: 75 %

ADS recommends that your application set all key parameters for the driver, and that it not rely upon driver defaults for any settings.

4.3 Files

The following files are needed to develop CAN applications:

`PeliCanSja1000.dll`
CAN driver described in this specification. This file is automatically loaded at boot by WindowsCE.

`CANapp.h`
Header file for CAN driver constants and data structures. Include this file in all applications that use this CAN driver.

`ADSerror.h`
Header file for ADS error codes. Include this file for error handling.

4.4 Registry Settings

The CAN driver reads the registry for initialization settings at boot time. The first CAN driver's settings are in the "CAN" key. Additional drivers use the key, "CANn" where n is the driver number.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN]
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN2]
```

The driver reads the following registry keys during initialization. The examples indicate the default values that the driver uses if the registry entries are not present.

Dll

Sets the name of the CAN driver . Changing this to a non-existent file name prevents the driver from loading.

```
"Dll"="PeliCanSja1000.dll"; use ADS PeliCAN SJA1000T driver
```

Priority256

Sets the CE driver priority. Values are in hexadecimal. The 256 at the end of the name is a reminder that the lowest priority is 256 or hexadecimal 0x100.

```
"Priority256"=dword:62    ; CAN driver priority = 0x62
```

MessageEvent

Sets the event to be pulsed when a message is received. A new event name may be substituted but if this key does not exist the event name shown will remain as the default.

```
"MessageEvent"="EV_APP0"  ; pulsed when message is received
```

ErrorEvent

Sets the event to be pulsed when an error is encountered. A new event name may be substituted but if this key does not exist the event name shown will remain as the default.

```
"ErrorEvent"="EV_ERR0"    ; pulsed when a CAN error occurs
```

DataReadyEvent

Sets the event to be set when the queue contains one or more messages. A new event name may be substituted but if this key does not exist the event name shown will remain as the default.

```
"DataReadyEvent"="EV_DATA_RDY0" ; set when RX queue is not empty
```

Deprecated Registry Keys

No deprecated keys at this time.

4.5 API Reference

The CAN driver follows the Windows CE stream driver interface standard. Stream interface functions used to access the ADS CAN driver include `CreateFile()`, `ReadFile()`, `WriteFile()`, `CloseHandle()` and `DeviceIoControl()`.

4.5.1 CreateFile()

Description: Opens the CAN driver for the specified port.

Prototype: Types shown are defined in `winbase.h`.

```
HANDLE CreateFile(
                LPCTSTR lpFileName,
                DWORD dwDesiredAccess,
                DWORD dwShareMode,
                LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                DWORD dwCreationDisposition,
                DWORD dwFlagsAndAttributes,
                HANDLE hTemplateFile
              );
```

Parameters: Values for parameters not shown should be NULL for pointer types and zero for numeric types.

`LPCTSTR lpFileName`
[in] Name of CAN driver to open (e.g. "`CAN1:`", "`CAN2:`")

Returns: NULL if an error occurs. Use `GetLastError()` to determine the specific reason for failure. ADS error codes are listed in the file `ADSerror.h`.

Availability: CAN driver 1.0 and later.

Blocking: Yes

Remarks: If the driver was not successfully initialized or has already been opened, this sets the last error to CAN_ERROR_CANNOT_OPEN_DEVICE. A memory allocation error sets the error to CAN_ERR_R_CANNOT_ALLOC_MEMORY.

Example:

```
HANDLE hCanPort;
hCanPort = CreateFile( _T("CAN1:"),
                  0,
                  0,
                  NULL,
                  0,
                  0,
                  NULL);
```

4.5.2 CloseHandle()

Description: Closes the CAN port referenced by *hObject*.

Prototype: Types shown are defined in `winbase.h`.

```
BOOL CloseHandle(
  HANDLE hObject
);
```

Parameters:

`HANDLE hObject`
[in] Handle to close

Returns: TRUE if successful or FALSE if there is an error.

Availability: CAN driver 1.0 and later.

Blocking: Yes

Remarks: The `GetLastError()` values are not updated.

Example:

```
CloseHandle(hCanPort);
```

4.5.3 Seek()

Description: Calls to the `Seek()` function have no effect and always return `0xFFFFFFFF`.

Prototype: `DWORD Seek( hOpenContext, Amount, Type )`

Returns: `0xFFFFFFFF` or -1 in decimal.

Availability: CAN driver v 1.0 and later.

Blocking: No

Remarks: This function included for Windows API compatibility.

4.5.4 ReadFile()

Description: Reads one or more messages from the CAN receive queue.

Prototype: Types shown are defined in `winbase.h`.

```
BOOL ReadFile(
               HANDLE hFile,
               LPVOID lpBuffer,
               DWORD nNumberOfBytesToRead,
               LPDWORD lpNumberOfBytesRead,
               LPOVERLAPPED lpOverlapped
               );
```

Parameters: Values for parameters not shown should be NULL for pointer types and zero for numeric types.

`HANDLE hFile,`
[in] CAN driver handle from `CreateFile()`

`LPVOID lpBuffer,`
[in] CAN_MSG structure array for retrieved messages

`DWORD nNumberOfBytesToRead,`
[in] Number of messages to read multiplied by `sizeof(CAN_MSG)`

`LPDWORD lpNumberOfBytesRead,`
[out] References a value equal to the number of messages actually read multiplied by `sizeof(CAN_MSG).`

Returns: TRUE if successful and FALSE if there was an error. Call `GetLastError()` for error information.

Availability: CAN driver v 1.0 and later.
v 1.1 and later use an updated message structure
v 1.3 and later support reading multiple messages.

Blocking: Yes

Remarks: While the `ReadFile()` function can polled, it is best used in conjunction with the `DataReadyEvent`. If the handle or any of the pointer values are invalid the ERROR_INVALID_PARAMETER value is generated.

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));
```

4.5.5 WriteFile()

Description: Places one or more messages into the outgoing queue for transmission on the CAN bus.

Prototype: Types shown are defined in `winbase.h`.

```
BOOL WriteFile(
               HANDLE hFile,
               LPCVOID lpBuffer,
               DWORD nNumberOfBytesToWrite,
               LPDWORD lpNumberOfBytesWritten,
               LPOVERLAPPED lpOverlapped
               );
```

Parameters: Values for parameters not shown should be NULL for pointer types and zero for numeric types.

`HANDLE hFile`
[in] CAN handle from `CreateFile()`

`LPCVOID lpBuffer`
[in] CAN_MSG structure array containing messages to be sent

`DWORD nNumberOfBytesToWrite`
[in] Number of messages to write, multiplied by `sizeof(CAN_MSG)`

`LPDWORD lpNumberOfBytesWritten`
[out] References a value equal to the number of messages actually written multiplied by `sizeof(CAN_MSG)`.

Returns: Non zero if successful and FALSE if there was an error. Call `GetLastError()` for error information

Availability: CAN driver v 1.0 and later.
v 1.1 and later use an updated message structure
v 1.3 and later support reading multiple messages.

Blocking: Yes

Remarks: If the CAN controller is in the BUS OFF state, `WriteFile()` will fail and `GetLastError()` will return the `CAN_ERROR_BUS_OFF` error code. An invalid pointer value for lpBuffer or lpNumberOfBytesWritten will generate an ERROR_INVALID_PARAMETER error and an un-initialized handle generates a CAN_ERROR_INVALID_HANDLE error.

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));
```

4.5.6 DeviceIoControl()

Description: Provides functions to control operation of the CAN controller and queue.

Prototype: Types shown are defined in `winbase.h`. The use of parameters `dwIoControlCode, lpInBuffer, nInBufferSize, lpOutBuffer,` and `nOutBufferSize` are function specific and are described in the ***Driver Reference:4.6 I/O Controls*** section.

```
BOOL DeviceIoControl(
                HANDLE hDevice,
                DWORD dwIoControlCode,
                LPVOID lpInBuffer,
                DWORD nInBufferSize,
                LPVOID lpOutBuffer,
                DWORD nOutBufferSize,
                LPDWORD lpBytesReturned,
                LPOVERLAPPED lpOverlapped
                );
```

Returns: TRUE if successful and FALSE if there was an error. Call `GetLastError()` for error information. If the `dwIoControlCode` value is not recognized, `GetLastError()` returns ERROR_BAD_COMMAND.

Availability: CAN driver 1.0 and later.

Blocking: Yes

Example:

```
UINT nNumMessages = 0;

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

4.6 I/O Controls

The I/O control codes listed below provide access to additional functionality in the ADS CAN driver. Parameters and execution of `DeviceIoControl()` was described in the previous section. The following are a summary of the control codes detailed in this section.

IOCTL_CAN_SET_ACCEPTANCE_FILTER (0x01)

IOCTL_CAN_GET_ACCEPTANCE_FILTER (0x02)

IOCTL_CAN_SET_BAUDRATE (0x03)

IOCTL_CAN_GET_BAUDRATE (0x04)

IOCTL_CAN_RESET_CHIP (0x05)

IOCTL_CAN_GET_STATUS_REG (0x07)

IOCTL_CAN_CLEAR_QUEUE (0x08)

IOCTL_CAN_GET_NUM_MSGS (0x09)

IOCTL_CAN_GET_SAMPLE_POINT (0x0A)

IOCTL_CAN_SET_SAMPLE_POINT (0x0B)

IOCTL_CAN_FLUSH_MSGS (0x0C)

IOCTL_CAN_GET_DRIVER_VERSION (0xA0)

4.6.1 IOCTL_CAN_SET_ACCEPTANCE_FILTER (0x01)

Description: Sets the message acceptance filter.

Parameters: Constants shown are defined in `CANApp.h`

`dwIoControlCode`
[in] Set to `IOCTL_CAN_SET_ACCEPTANCE_FILTER`.

`lpInBuffer`
[in] Pointer to a CAN_MSG_FILTER structure that contains the new filter settings.

`nInBufferSize`
[in] Set to `sizeof(CAN_MSG_FILTER)`.

Availability: CAN driver v 1.2 and later.

Remarks: If `nInBufferSize` does not match `sizeof(CAN_MSG_FILTER)`, `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_INPUT_BUFFER_WRONG_SIZE`.

4.6.2 IOCTL_CAN_GET_ACCEPTANCE_FILTER (0x02)

Description: Retrieves the current acceptance filter settings.

Parameters: Constants shown are defined in `CANApp.h`

`dwIoControlCode`
[in] Set to `IOCTL_CAN_GET_ACCEPTANCE_FILTER`.

`lpOutBuffer`
[in] Pointer to a `CAN_MSG_FILTER` structure that contains the new filter settings.

`nOutBufferSize`
[in] Set to `sizeof(CAN_MSG_FILTER)`.

Availability: CAN driver v 1.2 and later.

Remarks: If `nOutBufferSize` is less than `sizeof(CAN_MSG_FILTER)`, then `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_OUTPUT_BUFFER_TOO_SMALL`.

4.6.3 IOCTL_CAN_SET_BAUDRATE (0x03)

Description: Uses provided baud rate and the CAN sample point to generate and set the CAN baud rate.

Parameters: Constants shown are defined in `CANApp.h`

`dwIoControlCode`
[in] Set to `IOCTL_CAN_SET_BAUDRATE`.

`lpInBuffer`
[in] References a ULONG typed variable that contains the new baudrate in units of kilobits per second.

`nInBufferSize`
[in] Set to `sizeof(ULONG)`.

Remarks: Use `IOCTL_CAN_GET_BAUDRATE` to determine the actual baud rate set in the CAN controller.

Availability: CAN driver v 1.0 and later.

Remarks: If `nInBufferSize` does not match `sizeof(ULONG)`, `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_INPUT_BUFFER_WRONG_SIZE`.

4.6.4 IOCTL_CAN_GET_BAUDRATE (0x04)

Description: Returns the current CAN baudrate as read from the CAN chip.

Parameters: Constants shown are defined in `CANApp.h`

`dwIoControlCode`
[in] Set to `IOCTL_CAN_GET_BAUDRATE`.

`lpOutBuffer`
[in] References the ULONG typed variable that will receive the current baudrate

`nOutBufferSize`
[in] Set to `sizeof(ULONG)`.

Availability: CAN driver v 1.0 to 1.3 return baud rate stored in driver.
v 1.4 and later read baud rate from CAN controller.

Remarks: If `nOutBufferSize` is less than `sizeof(ULONG)`, then `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_OUTPUT_BUFFER_TOO_SMALL`.

4.6.5 IOCTL_CAN_RESET_CHIP (0x05)

Description: Resets the SJA1000T CAN controller.

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to `IOCTL_CAN_RESET_CHIP`.

Availability: CAN driver v 1.0 and later.

4.6.6 IOCTL_CAN_GET_STATUS_REG (0x07)

Description: Returns the current state of the SJA1000T status register (SJASR).

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to `IOCTL_CAN_GET_STATUS_REG`.

`lpOutBuffer`
[in] References the BYTE typed variable that will receive the status register state

`nOutBufferSize`
[in] Set to `sizeof(BYTE)`.

Availability: CAN driver v 1.0 and later.

Remarks: The status register is defined in SJA1000.h . If `nOutBufferSize` is less than `sizeof(BYTE)`, then `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_OUTPUT_BUFFER_TOO_SMALL`.

4.6.7 IOCTL_CAN_CLEAR_QUEUE (0x08)

Description: Flushes the driver's receive queue.

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to `IOCTL_CAN_CLEAR_QUEUE`.

Availability: CAN driver v 1.0 and later.

4.6.8 IOCTL_CAN_GET_NUM_MSGS (0x09)

Description: Returns the number of CAN messages currently stored in the driver receive queue.

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to IOCTL_CAN_GET_NUM_MSGS.

`lpOutBuffer`
[in] References the UINT typed variable that will receive the number of messages in the receive queue.

`nOutBufferSize`
[in] Set to `sizeof(UINT)`.

Availability: CAN driver v 1.0 and later.

Remarks: If `nOutBufferSize` is less than `sizeof(UINT)`, then `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_OUTPUT_BUFFER_TOO_SMALL`.

4.6.9 IOCTL_CAN_GET_SAMPLE_POINT (0x0A)

Description: Returns the current sample point as a percentage (i.e. interpret a returned value of 75 as "75%").

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to `IOCTL_CAN_GET_SAMPLE_POINT`.

`lpOutBuffer`
[in] References the `ULONG` typed variable that will receive the sample point value.

`nOutBufferSize`
[in] Set to `sizeof(ULONG)`.

Availability: CAN driver v 1.4 and later.

Remarks: If `nOutBufferSize` is less than `sizeof(ULONG)`, then `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_OUTPUT_BUFFER_TOO_SMALL`.

4.6.10 IOCTL_CAN_SET_SAMPLE_POINT (0x0B)

Description: Sets the CAN sample point as the close as possible to the value provided.

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to `IOCTL_CAN_SET_SAMPLE_POINT`.

`lpInBuffer`
[in] References a ULONG typed variable that contains the new sample point as a percentage value.

`nInBufferSize`
[in] Set to `sizeof(ULONG)`.

Availability: CAN driver v 1.4 and later.

Remarks: 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. Use IOCTL_CAN_GET_SAMPLE_POINT to read the actual value from the CAN controller. If `nInBufferSize` does not match `sizeof(ULONG)`, `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_INPUT_BUFFER_WRONG_SIZE`. If `nInBuffer` is greater than 100 the error generated is CAN_ERROR_INPUT_OUT_OF_RANGE.

4.6.11 IOCTL_CAN_FLUSH_MSGS (0x0C)

Description: Removes a specified number of messages from the CAN receive queue.

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to `IOCTL_CAN_FLUSH_MSGS`.

`lpInBuffer`
[in] References a ULONG typed variable that contains the number of messages to remove from the receive queue.

`nInBufferSize`
[in] Set to `sizeof(ULONG)`.

`lpOutBuffer`
[in] References the `ULONG` typed variable that will receive the number of messages deleted.

`nOutBufferSize`
[in] Set to `sizeof(ULONG)`.

Availability: CAN driver v 1.5 and later.

Remarks: The number of messages to flush is passed as integer in the "`lpInBuffer`" parameter of "`DeviceIoControl`". In case that number is greater than or equal to the actual number of messages existing in the queue, the queue content is cleared. If the number passed is less than the queue message count, then messages will be disregarded starting with the oldest ones in the queue.

4.6.12 IOCTL_CAN_GET_DRIVER_VERSION (0xA0)

Description: Returns the specification version number.

Parameters: Constants shown are defined in `CANApp.h` .

`dwIoControlCode`
[in] Set to `IOCTL_CAN_GET_DRIVER_VERSION`.

`lpOutBuffer`
[in] References the `wchar_t` typed array that will receive the version string. The returned value is a null terminated Unicode string.

`nOutBufferSize`
[in] Set to the size of supplied buffer, e.g. 128 bytes.

Availability: CAN driver v 1.2 and later.

Remarks: 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. If `nOutBufferSize` is less than the version string + 1, then `DeviceIoControl()` returns FALSE and `GetLastError()` will return `CAN_ERROR_OUTPUT_BUFFER_TOO_SMALL`.

4.7 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.

The message, error, and overrun events are all *pulsed* by the driver, while the `DataReadyEvent` is set and reset. A pulsed event is automatically reset when at least one thread waiting on it has been released.. This means that a thread must be waiting on the event at the moment it is pulsed in order to receive it, or 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 CAN driver provides a data ready event that is set when messages are ready in the receive queue, and that is 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.

The event name shown in the following listing is for the first CAN driver (CAN1:). Increment the last digit if additional drivers are loaded.

The Registry field is the name of the registry key that can set a different event name.

4.7.1 Data in Receive Queue

Description: Set when there are messages in the receive queue. Reset when the queue is empty.

Default Name: `EV_DATA_RDY0`

Data: None

Registry Key: `DataReadyEvent`

Availability: CAN driver v 1.3 and later.

Remarks: In most applications, use this event for processing messages in the event queue.

4.7.2 New Message Received

Description: Pulsed each time a CAN message arrives in the queue.

Default Name: `EV_APP0`

Data: None

Registry Key: `MessageEvent`

Remarks: This event can be used to determine message frequency, but is not used in most applications. To ensure that no messages are missed, perform very little processing after the event triggers.

4.7.3 CAN Controller Buffer Overflow

Description: Pulsed when the SJA1000T input buffer has overflowed (DOI bit is set in the SJA1000T interrupt register).

Default Name: `EV_OVRERR0`

Data: None

Registry Key: `OverrunEvent`

Remarks: You may need to increase the CAN thread priority so the driver can service incoming messages more quickly. Another option is to dump some number of older messages in the queue to leave space for newer messages.

4.7.4 CAN Controller Error

Description: Pulsed when EI (Error Warning Interrupt) or BEI (Bus Error Interrupt) bits in the SJA1000T interrupt register have been set.

Default Name: `EV_ERR0`

Data: None

Registry Key: `ErrorEvent`

4.7.5 CAN Data Queue Full

Description: Set when a message is lost due to a full data queue. Reset when queue is no longer full.

Default Name: `EV_CAN_DATAQ_FULL`

Data: None

Registry Key: `CanDataQFullEvent`

Remarks: The queue full condition can be cleared by a Readfile() or `IOCTL_FLUSH_MSGS` command. When the data queue is full, each new message overwrites the oldest message in the queue. In this condition the queue always contains the newest messages.

4.7.6 CAN Queue Threshold Reached

Description: Set as long as message queue is over half full.

Default Name: `EV_CAN_DATAQ_THRESHOLD`

Data: None

Registry Key: `CanDataQThresholdEvent`

Remarks: The driver resets this event when the queue is less than half full. The `Readfile()`, the `IOCTL_CAN_FLUSH_MSGS`, and the `IOCTL_CAN_CLEAR_QUEUE` command can bring the queue size smaller than the threshold.

4.8 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.

5 Creating a CAN Application

This section lists the CAN header file and provides sample code and references as a starting point for creating your own CAN applications.

5.1 Header Files

CANApp.h

CANApp.h is the header file for the CAN driver. It defines the constants required to use the CAN driver and is typically included in applications.

This header file is available in source code form from the ADS support web site and is provided here as a reference.

```
// CANapp.h
//
// 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.

#ifndef __CANAPP_H
#define __CANAPP_H

// 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_CAN_RESET_CHIP                    0x05
#define IOCTL_CAN_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_CAN_GET_DRIVER_VERSION    0xA0

#define IOCTL_CAN_FLUSH_MSGS            0x0C

// Deprecated
#define IOCTL_RESET_CHIP                        0x05
#define IOCTL_SEND_COMMAND                      0x06
#define IOCTL_GET_DRIVER_VERSION        0xA0

#endif __CANAPP_H
```

5.2 Sample Applications

For a detailed sample application, see the ADS Support Forums topic 1739, eVC CAN Sample Application. This application spawns either the sender or receiver thread under keyboard control, and then processes user input until user chooses to quit. The user is also prompted for the baud rate for data transmission.