Intel® Platform Controller Hub
EG20T
Controller Area Network (CAN) Driver for Windows® Programmer’s Guide

February 2011
Contents

1.0 Introduction ..............................................................................................................6
2.0 Operating System (OS) Support ................................................................................7
3.0 Dependencies ............................................................................................................8
4.0 CAN Driver API Details ..............................................................................................9
   4.1 Features ............................................................................................................9
   4.2 Interface Details ................................................................................................9
   4.3 IOCTL Usage Details .......................................................................................10
   4.4 Structures and Enumerations ............................................................................11
      4.4.1 Structures.............................................................................................11
         4.4.1.1 ioh_can_msg_t.........................................................................11
         4.4.1.2 ioh_can_timing_t......................................................................11
         4.4.1.3 ioh_can_error_t........................................................................11
         4.4.1.4 ioh_can_acc_filter_t.........................................................12
         4.4.1.5 ioh_can_rx_filter_t....................................................................12
      4.4.2 Enumerations ........................................................................................12
         4.4.2.1 ioh_can_listen_mode_t..............................................................12
         4.4.2.2 ioh_can_run_mode_t ................................................................12
         4.4.2.3 ioh_can_arbiter_mode_t............................................................13
         4.4.2.4 ioh_can_restart_mode_t............................................................13
         4.4.2.5 ioh_can_baud_t........................................................................13
         4.4.2.6 ioh_can_interrupt_t ..................................................................13
   4.5 Error Handling ..................................................................................................14
   4.6 Inter-IOCTL dependencies ..................................................................................14
5.0 Programming Guide .................................................................................................15
   5.1 Basic Flow........................................................................................................15
   5.2 Opening the Device ...........................................................................................15
      5.2.1 Using GUID Interface Exposed by the Driver ..............................................16
   5.3 Device Functionality...........................................................................................16
      5.3.1 CAN Device Configuration Options ......................................................16
      5.3.2 CAN Filter Configuration ......................................................................20
      5.3.3 CAN Clock Configuration .......................................................................21
      5.3.4 Getting CAN Error Status .....................................................................22
      5.3.5 Reading and Writing CAN Message .....................................................22
      5.3.6 Configuring CAN Receive/Transmit Message Object ...............................23
      5.3.7 CAN Device Reset ...............................................................................26
   5.4 Closing the Device.............................................................................................26

Tables

1   CAN Driver IOCTLs................................................................................................9
2   ioh_can_msg_t structure .......................................................................................11
3   ioh_can_timing_t structure..................................................................................11
4   ioh_can_error_t structure ..................................................................................12
5   ioh_can_acc_filter_t structure ..........................................................................12
6   ioh_can_rx_filter_t structure ............................................................................12
7   ioh_can_listen_mode_t enumeration ..................................................................12
8   ioh_can_run_mode_t enumeration ....................................................................13
9   ioh_can_arbiter_mode_t enumeration ................................................................13
10  ioh_can_restart_mode_t enumeration ................................................................13
11  ioh_can_baud_t enumeration .............................................................................13
12 ioh_can_interrupt_t enumeration.................................................................14
# Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 2011</td>
<td>002</td>
<td>Updated Section 2.0, &quot;Operating System (OS) Support&quot; on page 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Section 5.2.1, &quot;Using GUID Interface Exposed by the Driver&quot; on page 16</td>
</tr>
<tr>
<td>September 2010</td>
<td>001</td>
<td>Initial release</td>
</tr>
</tbody>
</table>
1.0 Introduction

Controller Area Network (CAN) is a multi-master broadcast serial bus standard for connecting electronic control units (ECUs). Each node is able to send and receive messages, but not simultaneously. A message consists primarily of an ID — usually chosen to identify the message-type or sender — and up to eight data bytes. It is transmitted serially onto the bus. This signal pattern is encoded in NRZ and is sensed by all nodes. Devices that are connected by a CAN network are typically sensors, actuators, and other control devices. These devices are not connected directly to the bus, but through a host processor and a CAN controller.

If the bus is free, then any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with more dominant ID (which has more dominant bits, i.e., zeroes) will overwrite other nodes’ less dominant IDs, so that eventually (after this arbitration on the ID) only the dominant message remains and is received by all nodes.

This document describes the CAN driver interfaces exposed to the user mode applications and how to use those interfaces to drive the CAN hardware to achieve effective communication.
### 2.0 Operating System (OS) Support

The CAN driver is supported by the following operating systems:

<table>
<thead>
<tr>
<th>No</th>
<th>OS</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Microsoft Windows XP*</td>
<td>Service Pack 3</td>
</tr>
<tr>
<td>2</td>
<td>Windows Embedded Standard*</td>
<td>2009</td>
</tr>
<tr>
<td>3</td>
<td>Windows Embedded POSReady*</td>
<td>2009</td>
</tr>
<tr>
<td>4</td>
<td>Microsoft Windows 7*</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Windows Embedded Standard7</td>
<td></td>
</tr>
</tbody>
</table>
3.0 Dependencies

The Intel® Platform Controller Hub EG20T CAN Hardware Assist driver is dependent upon the Intel® Platform Controller Hub EG20T Packet Hub driver to set up the clock frequency before the CAN operation can be started.
4.0 **CAN Driver API Details**

The CAN driver exposes the interfaces through Input/Output Controls (IOCTLs), which can be called from the user mode applications. The following sections provide information about the IOCTLs of the driver and how to use them to drive the CAN hardware successfully.

4.1 **Features**

The CAN driver supports:

- 32 message objects
- Setting bit rate up to 1 Mbits/sec
- Enabling the interruption to CAN hardware and setting the interrupt mask and mode
- Disabling the interruption to CAN hardware
- Set or clear selected registers of CAN message object
- Getting the status of selected registers of CAN message object
- Reading the data of selected CAN message object, when the other CAN device responds / event (interrupt) occurs
- Reading CAN hardware and bus status items
- FIFO mode select, programmable FIFO mode (concatenation of message objects) – using FIFO mode or not
- CAN bus byte/multi-byte read transactions
- CAN bus byte/multi-byte write transactions
- Notification that can be sent back to the user mode by using a system wide event object

4.2 **Interface Details**

Table 1 lists the IOCTLs supported by the CAN driver.

<table>
<thead>
<tr>
<th>No</th>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IOCTL_CAN_RESET</td>
<td>Reset the CAN device. Issuing this IOCTL causes the device to stop and reset. After the device has been reset, the device must be reconfigured and explicitly set to run.</td>
</tr>
<tr>
<td>2</td>
<td>IOCTL_CAN_RUN</td>
<td>Set the device to run. The device must be configured (e.g., timing (baud rate), active/listen mode, etc.) before using this command.</td>
</tr>
<tr>
<td>3</td>
<td>IOCTL_CAN_STOP</td>
<td>Stop all operations and the device. The device no longer transmits or receives packets. However, interrupts are still active.</td>
</tr>
<tr>
<td>4</td>
<td>IOCTL_CAN_RUN_GET</td>
<td>Get the current run state. This returns whether the device is currently running or stopped.</td>
</tr>
<tr>
<td>5</td>
<td>IOCTL_CAN_FILTER</td>
<td>Set the receive filter for a particular receive buffer.</td>
</tr>
<tr>
<td>6</td>
<td>IOCTL_CAN_FILTER_GET</td>
<td>Get the receive filter configuration for a particular receive buffer.</td>
</tr>
</tbody>
</table>
### Table 1. CAN Driver IOCTLs (Sheet 2 of 2)

<table>
<thead>
<tr>
<th>No</th>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>IOCTL_CAN_CUSTOM</td>
<td>Set the timing (i.e., baud rate) via custom timing values. Every aspect of the CAN timing must be specified. This should be used when the recommended timings cannot be used.</td>
</tr>
<tr>
<td>8</td>
<td>IOCTL_CAN_SIMPLE</td>
<td>Set the timing (baud rate) using one of the pre-defined recommended timings.</td>
</tr>
<tr>
<td>9</td>
<td>IOCTL_CAN_TIMING_GET</td>
<td>Get the current timing values.</td>
</tr>
<tr>
<td>10</td>
<td>IOCTL_CAN_BLOCK</td>
<td>Set file read and write operation to block. The read operation will wait until a buffer is received before returning. The write operation will not return until the message has been transmitted.</td>
</tr>
<tr>
<td>11</td>
<td>IOCTL_CAN_NON_BLOCK</td>
<td>Set file read and write operation to not block. The read operation returns immediately. If a message is waiting, the read operation is returned with that message. Otherwise, the operation returns without any data. The write operation returns immediately, i.e., does not wait until the message is fully transmitted.</td>
</tr>
<tr>
<td>12</td>
<td>IOCTL_CAN_BLOCK_GET</td>
<td>Get the blocking state of the device.</td>
</tr>
<tr>
<td>13</td>
<td>IOCTL_CAN_LISTEN</td>
<td>Set the device to listen. This allows the device to receive messages, based on the receive buffer filters, but not send any message. This is used for debugging.</td>
</tr>
<tr>
<td>14</td>
<td>IOCTL_CAN_ACTIVE</td>
<td>Set the device to active. This is the normal read/write operation of the device.</td>
</tr>
<tr>
<td>15</td>
<td>IOCTL_CAN_LISTEN_GET</td>
<td>Get the listen state of the device.</td>
</tr>
<tr>
<td>16</td>
<td>IOCTL_CAN_ARBITER_ROUND_ROBIN</td>
<td>Set the transmit buffer arbitration mode to round robin. This means that messages are placed in the next available slot using a round robin scheme (0, 1, ..., 7, 0, 1, ...).</td>
</tr>
<tr>
<td>17</td>
<td>IOCTL_CAN_ARBITER_FIXED_PRIORITY</td>
<td>Set the transmit buffer arbitration to fixed priority. This means that a message placed in transmit buffer 0 gets the highest priority. Transmit buffer 7 gets the lowest priority.</td>
</tr>
<tr>
<td>18</td>
<td>IOCTL_CAN_ARBITER_GET</td>
<td>Get the transmit buffer arbitration mode.</td>
</tr>
<tr>
<td>19</td>
<td>IOCTL_CAN_ERROR_STATS_GET</td>
<td>Get the error statistics of the CAN device.</td>
</tr>
<tr>
<td>20</td>
<td>IOCTL_CAN_BUFFER_LINK_SET</td>
<td>Set buffer linking for a particular receive buffer.</td>
</tr>
<tr>
<td>21</td>
<td>IOCTL_CAN_BUFFER_LINK_CLEAR</td>
<td>Clear the buffer linking for a particular receive buffer.</td>
</tr>
<tr>
<td>22</td>
<td>IOCTL_CAN_BUFFER_LINK_GET</td>
<td>Get the receive buffer linking status for a receive buffer.</td>
</tr>
<tr>
<td>23</td>
<td>IOCTL_CAN_RX_ENABLE_SET</td>
<td>Enable a receive buffer.</td>
</tr>
<tr>
<td>24</td>
<td>IOCTL_CAN_RX_ENABLE_CLEAR</td>
<td>Clear (disable) a receive buffer.</td>
</tr>
<tr>
<td>25</td>
<td>IOCTL_CAN_RX_ENABLE_GET</td>
<td>Get the receive buffer enable status.</td>
</tr>
<tr>
<td>26</td>
<td>IOCTL_CAN_TX_ENABLE_SET</td>
<td>Enable a transmit buffer.</td>
</tr>
<tr>
<td>27</td>
<td>IOCTL_CAN_TX_ENABLE_CLEAR</td>
<td>Clear (disable) a transmit buffer.</td>
</tr>
<tr>
<td>28</td>
<td>IOCTL_CAN_TX_ENABLE_GET</td>
<td>Get the transmit buffer enable status.</td>
</tr>
<tr>
<td>29</td>
<td>IOCTL_CAN_READ</td>
<td>Read and copy CAN messages to user space.</td>
</tr>
<tr>
<td>30</td>
<td>IOCTL_CAN_WRITE</td>
<td>Copy message from user space and transmit.</td>
</tr>
</tbody>
</table>

### 4.3 IOCTL Usage Details

This section provides the details for configuring the CAN interface and initiating CAN operations. The following files contain the details of the IOCTLs and data structures used for the configuration.
• ioh_can_ioctl – contains IOCTL definitions
• ioh_can_common.h – data structures and other variables used by the IOCTLs

Refer to Section 5.0 for the programming details.

4.4 Structures and Enumerations

This section provides the structures and enumerations used by interfaces exposed by the CAN driver. All the structures and enumerations used by the interfaces are defined in ioh_can_common.h.

4.4.1 Structures

4.4.1.1 ioh_can_msg_t

Structure for sending the message data.

Table 2. ioh_can_msg_t structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short ide</td>
<td>Standard/extended message</td>
</tr>
<tr>
<td>unsigned int id</td>
<td>11 or 29 bit msg id</td>
</tr>
<tr>
<td>unsigned short dlc</td>
<td>Size of data</td>
</tr>
<tr>
<td>unsigned char data? [IOH_CAN_MSG_DATA_LEN]</td>
<td>Message payload</td>
</tr>
<tr>
<td>unsigned short rtr</td>
<td>RTR message</td>
</tr>
</tbody>
</table>

4.4.1.2 ioh_can_timing_t

Structure for setting CAN timing values.

Table 3. ioh_can_timing_t structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int bitrate</td>
<td>Bitrate (kbps)</td>
</tr>
<tr>
<td>unsigned int cfg_bitrate</td>
<td>Bitrate value for Baud rate prescaler</td>
</tr>
<tr>
<td>unsigned int cfg_tseg1</td>
<td>Timing segment 1</td>
</tr>
<tr>
<td>unsigned int cfg_tseg2</td>
<td>Timing segment 2</td>
</tr>
<tr>
<td>unsigned int cfg_sjw</td>
<td>Sync jump width</td>
</tr>
<tr>
<td>unsigned int smpl_mode</td>
<td>Sampling mode</td>
</tr>
<tr>
<td>unsigned int edge_mode</td>
<td>Edge R / D</td>
</tr>
</tbody>
</table>

4.4.1.3 ioh_can_error_t

Structure for getting the CAN error status.
4.4.1.4 `ioh_can_acc_filter_t`
Structure contains filter information.

### Table 4. `ioh_can_error_t` structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int rxgte96</td>
<td>Rx error count &gt;=96</td>
</tr>
<tr>
<td>unsigned int txgte96</td>
<td>Tx error count &gt;=96</td>
</tr>
</tbody>
</table>
| unsigned int error_stat | Error state of CAN node:  
                          00=error active (normal)  
                          01=error passive  
                          1x=bus off |
| unsigned int rx_err_cnt | Rx counter                                               |
| unsigned int tx_err_cnt | Tx counter                                               |

4.4.2 Enumerations

This section lists the enumerations exposed by the interface.

4.4.2.1 `ioh_can_listen_mode_t`
CAN listen mode.

### Table 7. `ioh_can_listen_mode_t` enumeration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOH_CAN_ACTIVE</td>
<td>R/w to/from the CAN</td>
</tr>
<tr>
<td>IOH_CAN_LISTEN</td>
<td>Only read from the CAN</td>
</tr>
</tbody>
</table>

4.4.2.2 `ioh_can_run_mode_t`
CAN run mode.
4.4.2.3 **ioh_can_arbiter_mode_t**

Identifies valid values for the arbitration mode.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOH_CAN_ROUND_ROBIN</td>
<td>Equal priority</td>
</tr>
<tr>
<td>IOH_CAN_FIXED_PRIORITY</td>
<td>Buffer num priority</td>
</tr>
</tbody>
</table>

4.4.2.4 **ioh_can_restart_mode_t**

Identifies valid values for the auto-restart mode.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_MANUAL</td>
<td>Manual restart</td>
</tr>
<tr>
<td>CAN_AUTO</td>
<td>Automatic restart</td>
</tr>
</tbody>
</table>

4.4.2.5 **ioh_can_baud_t**

Identifies common baudrates.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOH_CAN_BAUD_10</td>
<td>10 Kbps</td>
</tr>
<tr>
<td>IOH_CAN_BAUD_20</td>
<td>20 Kbps</td>
</tr>
<tr>
<td>IOH_CAN_BAUD_50</td>
<td>50 Kbps</td>
</tr>
<tr>
<td>IOH_CAN_BAUD_125</td>
<td>125 Kbps</td>
</tr>
<tr>
<td>IOH_CAN_BAUD_250</td>
<td>250 Kbps</td>
</tr>
<tr>
<td>IOH_CAN_BAUD_500</td>
<td>500 Kbps</td>
</tr>
<tr>
<td>IOH_CAN_BAUD_800</td>
<td>800 Kbps</td>
</tr>
<tr>
<td>IOH_CAN_BAUD_1000</td>
<td>1000 Kbps</td>
</tr>
</tbody>
</table>

4.4.2.6 **ioh_can_interrupt_t**

Identifies interrupt enable/disable.
Table 12.  ioh_can_interrupt_t enumeration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_ENABLE</td>
<td>Enable bit only</td>
</tr>
<tr>
<td>CAN_DISABLE</td>
<td>Disable bit only</td>
</tr>
<tr>
<td>CAN_ALL</td>
<td>All interrupts</td>
</tr>
<tr>
<td>CAN_NONE</td>
<td>No interrupt</td>
</tr>
</tbody>
</table>

4.5 Error Handling

Since the IOCTL command is implemented using the Windows* API, the return value of the call is dependent on and defined by the OS. On Windows XP*, the return value is a non-zero value. If the error is detected within or outside the driver, an appropriate system defined value is returned by the driver.

4.6 Inter-IOCTL dependencies

There are no inter-IOCTL dependencies. Once the driver has been loaded successfully, the IOCTLs above can be used in any order.
5.0 Programming Guide

This section explains the basic procedure for using the CAN driver from a user mode application. All operations are through the IOCTLs exposed by the CAN driver. Refer to Section 4.2 for details on the IOCTLs. The steps involved in accessing the CAN driver from the user mode application are described below:

- Opening the Device
- Configure the device for different modes of operations
- Closing the Device

5.1 Basic Flow

The basic flow to use this driver from a user mode application is to open the device, configure the device, transmit/receive messages and then close the device. The following describes the flow using pseudo-code.

```c
main()
{
    /* Open the CAN device driver */
    hDevice = CreateFile(DriverName,
                         GENERIC_READ|GENERIC_WRITE,
                         0, NULL, OPEN_EXISTING, 0,NULL);

    /* Configure the CAN device driver */
    DeviceIoControl(hDevice, IOCTL_CAN_SIMPLE,
                    &baud,sizeof(baud),NULL,0,&dwBytesReturned,NULL);

    /* Receive/transmit messages. */
    DeviceIoControl(hDevice, IOCTL_CAN_READ,
                    NULL,0,&msg,sizeof(msg),&dwBytesReturned,NULL);
    DeviceIoControl(hDevice, IOCTL_CAN_WRITE,
                    &msg,sizeof(msg),NULL,0,&dwBytesReturned,NULL);

    /* Close the CAN device driver */
    CloseHandle(hDevice);
}
```

5.2 Opening the Device

CAN Device is opened using CreateFile Win32 API. To get the device name, refer to Section 5.2.1.
5.2.1 Using GUID Interface Exposed by the Driver

A device interface class is a way of exporting device and driver functionality to other system components, including other drivers, as well as user-mode applications. A driver can register a device interface class, and then enable an instance of the class for each device object to which user-mode I/O requests might be sent. The Intel® PCH EG20T CAN driver registers the following interface.

<table>
<thead>
<tr>
<th>No</th>
<th>Interface Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GUID_DEVINTERFACE_IOHCAN</td>
</tr>
</tbody>
</table>

This is defined in ioh_can_common.h.

Device interfaces are available to both kernel-mode components and user-mode applications. User-mode code can use SetupDiXxx functions to find out the registered, enabled device interfaces.

Please refer the following site to get the details about SetupDiXxx functions.


5.3 Device Functionality

This section describes how to configure the device to achieve time synchronization on Ethernet and CAN.

5.3.1 CAN Device Configuration Options

This section describes the IOCTLs that are used to configure the CAN device and to get different device operation mode statuses.

- IOCTL_CAN_RESET
  This ioctl is called to reset the device to a known state.

```c
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_RESET,
    NULL,
    0,
    NULL,
    0,
    &dwRet,
    NULL);
```

- IOCTL_CAN_RUN
  This ioctl is called to set the device to run mode.

```c
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_RUN,
    NULL,
    0,
```
- **IOCTL_CAN_RUN_GET**
  This ioctl is called to get the current run mode of the device.

  ```
  ULONG runmode;
  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_RUN_GET,
    NULL,
    0,
    &runmode,
    sizeof(runmode),
    &dwRet,
    NULL);
  ```

- **IOCTL_CAN_BLOCK**
  This ioctl is called to put the device to Block mode.

  ```
  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_BLOCK,
    NULL,
    0,
    NULL,
    0,
    &dwRet,
    NULL);
  ```

- **IOCTL_CAN_NON_BLOCK**
  This ioctl is called to put the device to non-block mode.

  ```
  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_NON_BLOCK,
    NULL,
    0,
    NULL,
    0,
    &dwRet,
    NULL);
  ```
• **IOCTL_CAN_BLOCK_GET**  
  This ioctl is called to get the current block mode of the device.

  ULONG ulBlockGet;
  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_BLOCK_GET,
    NULL,
    0,
    &ulBlockGet,
    sizeof(ulBlockGet),
    &dwRet,
    NULL);

• **IOCTL_CAN_STOP**  
  This ioctl is called to put the device to stop mode.

  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_STOP,
    NULL,
    0,
    NULL,
    0,
    &dwRet,
    NULL);

• **IOCTL_CAN_ACTIVE**  
  This ioctl is called to put the device to active mode.

  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_ACTIVE,
    NULL,
    0,
    NULL,
    0,
    &dwRet,
    NULL);

• **IOCTL_CAN_LISTEN**  
  This ioctl is called to put the device to listen mode.

  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_LISTEN,
- **IOCTL_CAN_LISTEN_GET**
  This ioctl is called to get the current listen mode of the device.

  ```c
  ULONG ulListenGet;
  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_LISTEN_GET,
    NULL,
    0,
    &ulListenGet,
    sizeof(ULONG),
    &dwRet,
    NULL);
  ```

- **IOCTL_CAN_ARBITER_ROUND_ROBIN**
  This ioctl is called to put the device priority to round robin.

  ```c
  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_ARBITER_ROUND_ROBIN,
    NULL,
    0,
    NULL,
    0,
    &dwRet,
    NULL);
  ```

- **IOCTL_CAN_ARBITER_FIXED_PRIORITY**
  This ioctl is called to put the device priority to fixed priority.

  ```c
  bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_ARBITER_FIXED_PRIORITY,
    NULL,
    0,
    NULL,
    0,
    NULL,
    NULL);
  ```
• IOCTL_CAN_ARBITER_GET
  This ioctl is called to get the current priority of the device.

  ULONG ulArbiterGet;
  bRet = DeviceIoControl( hDevice,
                          IOCTL_CAN_ARBITER_GET,
                          NULL,
                          0,
                          &ulArbiterGet,
                          sizeof(ULONG),
                          &dwRet,
                          NULL);

5.3.2 CAN Filter Configuration

• IOCTL_CAN_FILTER
  This ioctl is called to set the receive filter for a receive buffer.

  accFilter.id = 0x01;
  accFilter.id_ext = 0x00;
  accFilter.rtr = 0;
  filter.num = 1;
  filter.umask = 0xffffffff;
  filter.amr = accFilter;
  filter.aidr = accFilter;

  bRet = DeviceIoControl( hDevice,
                          IOCTL_CAN_FILTER,
                          &filter,
                          sizeof(ioh_can_rx_filter_t),
                          NULL,
                          0,
                          &dwRet,
                          NULL);

• IOCTL_CAN_FILTER_GET
  This ioctl is called to get the current filter.

  ULONG ulArbiterGet;
  bRet = DeviceIoControl( hDevice,
                          IOCTL_CAN_FILTER_GET,
accFilter.id = 0x01;
accFilter.id_ext = 0x00;
accFilter.rtr=0;

filter.num = 1;
filter.umask = 0xffff;
filter.amr = accFilter;
filter.aidr = accFilter;

bRet = DeviceIoControl(hDevice,
                      IOCTL_CAN_FILTER_GET,
                      NULL,
                      0,
                      &filter,
                      sizeof(ioh_can_rx_filter_t),
                      &dwRet,
                      NULL);

5.3.3 CAN Clock Configuration

- IOCTL_CAN_CUSTOM
  This ioctl is called to set the custom clock rate.

  bRet = DeviceIoControl(hDevice,
                         IOCTL_CAN_CUSTOM,
                         &NULL,
                         0,
                         &NULL,
                         NULL,
                         0,
                         &dwRet,
                         NULL);

- IOCTL_CAN_SIMPLE
  This ioctl is called to set a different baud rate for the device.

  ioh_can_baud_t baudrate = IOH_CAN_BAUD_20;

  bRet = DeviceIoControl(hDevice,
                         IOCTL_CAN_SIMPLE,
                         &baudrate,
Sizeof(baudrate),
NULL,
0,
&dwRet,
NULL);

• IOCTL_CAN_TIMING_GET
  This ioctl is called to get the current clock setting of the device.

ioh_can_timing_t timing;
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_TIMING_GET,
    NULL,
    0,
    &timing,
    Sizeof(ioh_can_timing_t)
    &dwRet,
    NULL);

5.3.4 Getting CAN Error Status

• IOCTL_CAN_ERROR_STATS_GET
  This ioctl is called to get the error status of the device.

ioh_can_error_t errorStat;
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_ERROR_STATS_GET,
    &NULL,
    0,
    &errorStat,
    Sizeof(ioh_can_error_t),
    &dwRet,
    NULL);

5.3.5 Reading and Writing CAN Message

Following IOCTLs are used for getting receive and transmit CAN messages.

• IOCTL_CAN_WRITE
  This ioctl is called to write the message.

ioh_can_msg_t msg;
msg.ide=0;
msg.id=(0x7ff);
msg.dlc=1;
msg.data[0]=10;
msg.rtr=0;

bRet = DeviceIoControl(hDevice,
    IOCTL_CAN_WRITE,
    &msg,
    sizeof(msg),
    NULL,
    0,
    &dwRet,
    NULL);

• IOCTL_CAN_READ
This ioctl is called to read the message.

ioh_can_msg_t msg;
bRet = DeviceIoControl(hDevice,
    IOCTL_CAN_READ,
    NULL,
    0,
    &msg,
    sizeof(msg),
    &dwRet,
    NULL)

5.3.6 Configuring CAN Receive/Transmit Message Object

• IOCTL_CAN_RX_ENABLE_SET
This ioctl is called to enable the receive buffer.

unsigned int uiReceiveBuffNo=1;
bRet = DeviceIoControl(hDevice,
    IOCTL_CAN_RX_ENABLE_SET,
    &uiReceiveBuffNo,
    sizeof(uiReceiveBuffNo),
    NULL,
    0,
&dwRet,
NULL);

- **IOCTL_CAN_RX_ENABLE_CLEAR**
  This ioctl is called to clear the receive buffer.

```c
unsigned int uiReceiveBuffNo=1;
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_RX_ENABLE_CLEAR,
    &uiReceiveBuffNo,
    sizeof(uiReceiveBuffNo),
    NULL,
    0,
    &dwRet,
    NULL);
```

- **IOCTL_CAN_RX_ENABLE_GET**
  This ioctl is called to get the receive buffer status.

```c
unsigned int uiReceiveBuffNo=1;
unsigned int uiStatus;
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_RX_ENABLE_GET,
    &uiReceiveBuffNo,
    sizeof(uiReceiveBuffNo),
    &uiStatus,
    sizeof(uiStatus),
    &dwRet,
    NULL);
```

- **IOCTL_CAN_TX_ENABLE_SET**
  This ioctl is called to enable the transmit buffer.

```c
unsigned int uiTransmitBuffNo=1;
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_TX_ENABLE_SET,
    &uiTransmitBuffNo,
    sizeof(uiTransmitBuffNo),
    NULL,
    0,
    &dwRet,
    NULL);
```
NULL);

- **IOCTL_CAN_TX_ENABLE_CLEAR**
  This ioctl is called to clear the transmit buffer.

```c
unsigned int uiTransmitBuffNo = 1;
bRet = DeviceIoControl( hDevice,
                        IOCTL_CAN_TX_ENABLE_CLEAR,
                        &uiTransmitBuffNo,
                        sizeof(uiTransmitBuffNo),
                        NULL,
                        0,
                        &dwRet,
                        NULL);
```

- **IOCTL_CAN_TX_ENABLE_GET**
  This ioctl is called to get the transmit buffer status.

```c
unsigned int uiTransmitBuffNo = 1;
unsigned int uiStatus;
bRet = DeviceIoControl( hDevice,
                        IOCTL_CAN_TX_ENABLE_GET,
                        &uiTransmitBuffNo,
                        sizeof(uiTransmitBuffNo),
                        &uiStatus,
                        sizeof(uiStatus),
                        &dwRet,
                        NULL);
```

- **IOCTL_CAN_BUFFER_LINK_SET**
  This ioctl is called to set the buffer link.

```c
unsigned int uiReceiveBuffNo = 1;
bRet = DeviceIoControl( hDevice,
                        IOCTL_CAN_BUFFER_LINK_SET,
                        &uiReceiveBuffNo,
                        sizeof(uiReceiveBuffNo),
                        NULL,
                        0,
                        &dwRet,
                        NULL);
```
• **IOCTL_CAN_BUFFER_LINK_CLEAR**
  This ioctl is called to clear the buffer link.

```c
unsigned int uiReceiveBuffNo=1;
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_BUFFER_LINK_CLEAR,
    &uiReceiveBuffNo,
    sizeof(uiReceiveBuffNo),
    NULL,
    0,
    &dwRet,
    NULL);
```

• **IOCTL_CAN_BUFFER_LINK_GET**
  This ioctl is called to get the buffer link status.

```c
unsigned int uiReceiveBuffNo=1;
unsigned int uiStatus;
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_BUFFER_LINK_GET,
    &uiReceiveBuffNo,
    sizeof(uiReceiveBuffNo),
    &uiStatus,
    sizeof(uiStatus),
    &dwRet,
    NULL);
```

### 5.3.7 CAN Device Reset

The following IOCTL is used for resetting the device.

- **IOCTL_CAN_RESET**

```c
bRet = DeviceIoControl( hDevice,
    IOCTL_CAN_RESET, NULL, 0, NULL, 0, &dwRet, NULL);
```

### 5.4 Closing the Device

Once all the operations related to the CAN driver are finished, the device handle must free the application by calling the Win32 API `CloseHandle`.

`CloseHandle(hHandle);`