Welcome to use Advantech series products. This manual describes how to operate Advantech data communication devices through Advantech CAN Windows WDM&CE Driver. This manual supplies information about driver interfaces of Advantech CAN device, including calling procedure of operating data communication and descriptions of functions of each function, parameter and data structure. With the help of Advantech CAN Windows WDM&CE Driver, users can develop applications with tools like VC, VB, VB.net, C#.NET, VC.NET, eVC in different Windows system platforms (Windows 2K/XP/Vista/CE). This manual also provides examples for Advantech device driver, explaining how to use Advantech with series of real examples and offering reference for users to develop their own applications.

Click the following links to enter different section in this chapter. In this document, you can also use table of contents, search and scan the content by index.

Sections include:

- **Copyright**
- **Technical Support**
- **Limited Warranty**
Copyright

This document is copyrighted by Advantech. Co., Ltd. All rights are reserved. The original manufacturer reserves the right to make improvements to the products described in this manual at any time without notice. No part of this manual may be reproduced, copied, translated or transmitted in any form or by any means without the prior written permission of the original manufacturer. Information provided in this manual is intended to be accurate and reliable. However, the original manufacturer assumes no responsibility for its use, nor for any infringements upon the rights of third parties that may result from such use.
Advantech customer services

Each and every Advantech product is built to the most exacting specifications to ensure reliable performance in the harsh and demanding conditions typical of industrial environments. Whether your new Advantech equipment is destined for the laboratory or the factory floor, you can be assured that your product will provide the reliability and ease of operation for which the name Advantech has come to be known. Your satisfaction is our primary concern. Here is a guide to Advantech’s customer services. To ensure you get the full benefit of our services, please follow the instructions below carefully.
Technical Support

We want you to get the maximum performance from your products. So if you run into technical difficulties, we are here to help. For the most frequently asked questions, you can easily find answers in your product documentation. These answers are normally a lot more detailed than the ones we can give over the phone. So please consult this manual first. If you still cannot find the answer, gather all the information or questions that apply to your problem, and with the product close at hand, call your dealer. Our dealers are well trained and ready to give you the support you need to get the most from your Advantech products. In fact, most problems reported are minor and are able to be easily solved over the phone. In addition, free technical support is available from Advantech engineers every business day. We are always ready to give advice on application requirements or specific information on the installation and operation of any of our products.
Technical Support Offices

USA
USA Advantech
Advantech Corporation, Industrial Automation Group
1320 Kemper Meadow Dr., Ste. 500
Cincinnati, OH 45240
Tel: (877) 451-6868
Fax: (513) 742-4451
E-mail: info@advantech.com

Asia
Advantech Co.,Ltd 4th Floor, 108-3 Min-Chuan Road
Shing-Tien City, Taipei County, Taiwan ROC
Tel: (+886-2) 2218-4567
Fax: (+886-2) 2218-1989
E-mail: IASupport@advantech.com.tw

Europe
Germany Advantech
Karlsruherstr. 11/1
D-70771 Leinf.-Echterdingen
Germany
Tel: +49 (0) 711 797 333 60
Fax: +49 (0) 711 797 333 85

Italy
Italy Advantech
Via Don Verderio
4/B-20060 Cassina de, Pecchi (MI), Italy
Tel: +39-2-95343054
Fax: +39-2-95343067

China Mainland
Beijing Advantech
No.7 6th Street Shangdi Zone, Haidian District, Beijing, China
Post code: 100085
Tel: +86-10-62984345~47, 62986314~17
Fax: +86-10-62984341~42

Shanghai Advantech
Room 701, 7F, Longfeng Building, No. 1566, Yan'an West Road, Shanghai, China
Post Code: 200232
Tel: +86-21-64696831, 64697910
Fax: +86-21-64696834
Limited warranty

Advantech Co., Ltd does not warrant that Advantech CAN Driver can work normally in any hardware/software environment. Advantech Co., Ltd does not warrant the information described in this manual, especially no warranty is made for any special purpose. Advantech Co., Ltd assumes no responsibility for its use, nor for any infringements upon the rights of third parties that may result from such use. Advantech Co., Ltd reserves the right to make improvements to the products described in this manual at any time without notice.
Overview

Advantech CAN Windows WDM&CE Driver is composed of bus drivers, ports drivers, tools and examples. Bus drivers and ports drivers, which run in PeliCAN mode, are used to drive SJA1000 chip on Advantech CAN device. Bus drivers and ports drivers are compliant with PCI and ISA bus, and provide the users with coherent operation interfaces. Users can directly write applications with Windows API. Examples of VC, VB, VB.NET, C#.NET, VC.NET, eVC are supplied in the package, providing a reference for users to develop applications. When developing work is completed, users can use test tools to verify if functions of the application are correct.
Feature

1. Supports CAN 2.0B Protocol and compatible with CAN 2.0A Protocol, which means both Standard frame and Extended frame can be dealt with.

2. Supports single filter mode and dual filter mode.


4. Supports self reception mode on WDM platform.

5. Provides API interface similar to windows standard serial port, easy to develop.

6. Provides tools like CANTest, CANDemo to test the basic functions.

7. Provides CANMonitor to monitor CAN/CANopen network. A maximum of 1000000 frame data can be reserved a time and meanwhile CAN/CNOpen messages can be dealt with.

8. Provides interface COTI.DLL of CANopen Conformance Test Tool. The COTI DLL allows users to use the CANopen Conformance Test Tool of CiA (CAN in Automation) with Advantech CAN WDM Driver.
CAN 2.0A and CAN 2.0B

CAN 2.0 Spec. includes CAN 2.0A and CAN 2.0B. CAN 2.0A supports standard 11 bit Identifier. CAN 2.0B supports both 11 bit Identifier and extended 29 bit Identifier. So CAN 2.0B are compatible with CAN 2.0A.
SJA1000 supports BasicCAN mode and PeliCAN mode. BasicCAN mode supports CAN 2.0A, while PeliCAN mode supports 2.0B. Advantech CAN Windows WDM&CE Driver runs in PeliCAN mode, thus it can support both Standard frame and Extended frame.
System Requirements:
The usage of WDM is different from that of CE in the following aspects:

**Driver Installation**

**WDM:**
The user should refer to Install WDM Driver to install the driver.

**CE:**
The driver has been pre-built in Platform image, so the user doesn't need to install the driver.

**Development Kit Installation**

**WDM:** none

**CE:** Advantech platform SDK

**Function**

**WDM:** Support all the functions listed in the manual.

**CE:** Not support self reception function.

**Interface**

The interfaces that WDM and CE support are almost the same. The user should pay attention to the following differences:

1. The device name is different.
   - Here CAN1 is used as an example:
     - **WDM:** \\\.\CAN1
     - **CE:** CAN1:
   - Functions involved: CreateFile

2. Since OVERLAPPED type is not supported by CE, NULL pointers will be loaded instead of parameters using OVERLAPPED structure.
   - Functions involved: DeviceIoControl, ReadFile, WriteFile, WaitCommEvent.

3. OVERLAPPED type is not supported by CE, so does not support GetOverLappedResult function.

**Hardware Support:**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Description</th>
<th>WDM</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCM-3680</td>
<td>2 port Isolated ISA CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PCL-841</td>
<td>2 port Isolated ISA CAN bus</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Model</td>
<td>Description</td>
<td>Support 1</td>
<td>Support 2</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>TPC-662G</td>
<td>1 port Isolated ISA CAN bus Device on TPC-662G.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PCI-1680</td>
<td>2 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>UNO-2052(E)</td>
<td>2 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>EAMB-PH07</td>
<td>1 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TPC-68T</td>
<td>1 port Isolated ISA CAN bus Device on TPC-68T.</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TPC-120H</td>
<td>1 port Isolated ISA CAN bus Device on TPC-120H.</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TPC-32T</td>
<td>1 port Isolated ISA CAN bus Device on TPC-32T.</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AMAX-2050</td>
<td>1 port Isolated PCI CAN bus Device on AMAX-2050.</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ADAM-5095</td>
<td>2 port Isolated PCI CAN bus Card.</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C001 CAN card (1 PORT)</td>
<td>1 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C002 CAN card (2 PORT)</td>
<td>2 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C004 CAN card (4 PORT)</td>
<td>4 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C101 CAN card (1 PORT, support CANopen)</td>
<td>1 port Isolated PCI CAN bus Card and support CANopen.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C102 CAN card (2 PORT, support CANopen)</td>
<td>2 port Isolated PCI CAN bus Card and support CANopen.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C104 CAN card (4 PORT, support CANopen)</td>
<td>4 port Isolated PCI CAN bus Card and support CANopen.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C201 CAN card (1 PORT)</td>
<td>1 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C202 CAN card (2 PORT)</td>
<td>2 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Model</td>
<td>Description</td>
<td>Support</td>
<td>Linux</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>ADVANTECH C204 CAN card (4 PORT)</td>
<td>4 port Isolated PCI CAN bus Card.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C301 CAN card (1 PORT, support CANopen)</td>
<td>1 port Isolated PCI CAN bus Card and support CANopen.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C302 CAN card (2 PORT, support CANopen)</td>
<td>2 port Isolated PCI CAN bus Card and support CANopen.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ADVANTECH C304 CAN card (4 PORT, support CANopen)</td>
<td>4 port Isolated PCI CAN bus Card and support CANopen.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Users of Windows CE can refer to [Guide for developing applications](#)
References:
"SJA1000 Standard-alone CAN controller"
Begin to use Advantech CAN device driver

The following chart shows procedures of developing applications with Advantech CAN Driver.

1. Insert the card and turn on the computer
2. Install the driver according to instruction in the manual
3. Restart the computer
4. Use matching Utility to test hardware
5. Read Manual
6. Write Application

There are different installation procedures for PCI device driver and ISA device driver, please respectively refer to:

- [PCI device installation procedures](#)
- [ISA device installation procedures](#)
Note:
Default installation path for Advantech CAN WDM Driver is **C:\Program Files\Advantech\AdvCAN**.
PCI device setup

The following installation procedures are for PCI devices.

Take PCI-1680 as example:
If you insert the PCI-1680 first and then install the driver, the driver installation will complete the driver installation automatically.
If you install the driver first, then power off and insert the PCI-1680, when you turn on the computer and enter operating system.
You will see the following screen.

• Click "Next" as the following.
- Click "Finish" as the following.
The following installation procedures are for ISA devices.

Take PCM-3680 as an example.

- Follow the instructions in the manual to install the device, then turn on the computer to enter operating system.
- Enter Control Panel, then select "Add Hardware".

Click "Next" as the following.
• Click "Next" as the following.

• Click "Next" as the following.
• Click "Next" as the following.

• Click "Have Disk...".
• Suppose inf file is installed under C:\Program Files\Advantech\AdvCAN\Driver, click "Next" as the following.

• Click "Next" as the following.
When the installation is complete, click "View or change resources for this hardware (Advanced)" to configure resources of the device. Users can also directly click "Finish" and configure it in Device Manager.

- Select "Set Configuration Manually".
Configure according to your hardware.
• Restart the computer to finish the installation.
Port setup

When bus driver is installed, users can install port driver AdvCanPort.sys according to the instructions. The installation file is AdvCanPort.inf. When the installation is finished, users can find the device in Device Manager and set ports in properties page.
Device setup

Users can set the device in "Windows Device Manager".

- When driver is installed, users can open "Windows Device Manager" to set Advantech CAN device;

  "Windows Device Manager" can be opened in the following two ways:

  1. Control Panel->Administrative Tools->Computer Management, then select "Device Manager";
  2. Right click icon of "My Computer" and click "Manage", then select "Device Manager".

- Click the eAutomation CAN port that you want to configure, and select "Properties" to open properties page of the port. Users can set the device in PORT_INFO.
CANMonitor can be used to monitor the messages in CAN network and to show the messages according to different message types. CANMonitor can show messages in the formats of CAN and CANopen.

System Requirements:


Function Introduction
Interface Introduction
## Function Introduction

This section introduces the functions that CANMonitor supports.

### Base function:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Device</td>
<td>Opens a device and make basic configuration.</td>
</tr>
<tr>
<td>Close Device</td>
<td>Closes a device and clear all the data.</td>
</tr>
<tr>
<td>Load from file</td>
<td>Loads data from the file.</td>
</tr>
<tr>
<td>Save to file</td>
<td>Saves data to the specified file.</td>
</tr>
<tr>
<td>Start Communication</td>
<td>Restarts communication based on the configuration.</td>
</tr>
<tr>
<td>Stop Communication</td>
<td>Stops communication, which is used to view data when the refresh is frequent.</td>
</tr>
<tr>
<td>Show Message</td>
<td>Shows messages.</td>
</tr>
<tr>
<td>Hide Message</td>
<td>Hides messages.</td>
</tr>
<tr>
<td>Clear Data</td>
<td>Clears all data.</td>
</tr>
<tr>
<td>Auto Scroll</td>
<td>Supports auto scroll function.</td>
</tr>
<tr>
<td>Goto</td>
<td>Goes to the specified message line.</td>
</tr>
<tr>
<td>Listen only mode</td>
<td>Supports Listen Only mode.</td>
</tr>
<tr>
<td>CAN Message Filter</td>
<td>Configures CAN message filter modes.</td>
</tr>
<tr>
<td>CAN Message View</td>
<td>Shows CAN format message.</td>
</tr>
</tbody>
</table>

### CANopen function:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Node</td>
<td>Lists CANopen nodes detected and the node state.</td>
</tr>
<tr>
<td>Change Node State</td>
<td>Changes the specified CANopen node state.</td>
</tr>
<tr>
<td>Read Data from Node</td>
<td>Reads data from the specified CANopen node.</td>
</tr>
<tr>
<td>Write Data to Node</td>
<td>Writes data to the specified CANopen node.</td>
</tr>
<tr>
<td>Detect All Nodes in CANopen</td>
<td>Detects all the possible CANopen nodes, from 1 to 127.</td>
</tr>
<tr>
<td>CAN Message Filter</td>
<td>Configures CAN message filter modes.</td>
</tr>
</tbody>
</table>
CANopen Message Filter
Configures CANopen message filter modes.

CAN Message View
Shows CAN format message.

CANopen Message View
Shows CANopen format message.
Introduction of Main Items in Menu Screen

Menu Introduction

**File**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Open device</td>
</tr>
<tr>
<td>Close</td>
<td>Close device</td>
</tr>
<tr>
<td>Load</td>
<td>Load data from File</td>
</tr>
<tr>
<td>Save</td>
<td>Save data to File</td>
</tr>
<tr>
<td>Exit</td>
<td>Exit the program</td>
</tr>
</tbody>
</table>

**View**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolbar</td>
<td></td>
</tr>
<tr>
<td>Status Bar</td>
<td></td>
</tr>
<tr>
<td>Auto Scroll</td>
<td></td>
</tr>
<tr>
<td>Clear</td>
<td></td>
</tr>
<tr>
<td>Goto</td>
<td>Ctrl+G</td>
</tr>
<tr>
<td>Data Format</td>
<td></td>
</tr>
</tbody>
</table>

View has the following functions:
Data Format has following setting:

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Hex</th>
<th>Dec</th>
<th>Ascii</th>
</tr>
</thead>
</table>

### Name
- Hex
- Dec
- Ascii

### Function

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hex</td>
<td>The monitor data which is include &quot;ID&quot; and &quot;Data&quot; will be displayed as Hex format.</td>
</tr>
<tr>
<td>Dec</td>
<td>The monitor data which is include &quot;ID&quot; and &quot;Data&quot; will be displayed as Dec format.</td>
</tr>
<tr>
<td>Ascii</td>
<td>The monitor &quot;Data&quot; will be displayed as Dec format but &quot;ID&quot; will be displayed as Hex format</td>
</tr>
</tbody>
</table>

### Function

Function has the following functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Start communication</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop communication</td>
</tr>
<tr>
<td>Show</td>
<td>Show the message</td>
</tr>
<tr>
<td>Hide</td>
<td>Hide the message</td>
</tr>
<tr>
<td>Listen Only</td>
<td>Whether to open Listen Only mode</td>
</tr>
</tbody>
</table>

CAN has a submenu:
<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send CAN Message</td>
<td>Send CAN Message</td>
</tr>
</tbody>
</table>

CANopen has the following functions:

- **List Node**: List all CANopen nodes
- **Change Node State**: Change CANopen node state
- **Read Object**: Read CANopen node data
- **Write Object**: Write CANopen node data
- **Node Guard All**: Get all CANopen nodes

**Filter**

Filter has **CAN Filter** setting

- **CAN**
  - SFF
  - EFF
  - RTR
  - Sel Reception

and **CANopen Filter** setting.

- **CANopen**
  - Identifier
  - Sub Protocol
### Toolbar Introduction

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td><strong>Open device</strong></td>
</tr>
<tr>
<td>Close</td>
<td><strong>Close Device</strong></td>
</tr>
<tr>
<td>Load</td>
<td><strong>Load data from file</strong></td>
</tr>
<tr>
<td>Save</td>
<td><strong>Save data to file</strong></td>
</tr>
<tr>
<td>Run</td>
<td><strong>Start communication</strong></td>
</tr>
<tr>
<td>Stop</td>
<td><strong>Stop communication</strong></td>
</tr>
<tr>
<td>Show</td>
<td><strong>Show the message</strong></td>
</tr>
<tr>
<td>Hide</td>
<td><strong>Hide the message</strong></td>
</tr>
<tr>
<td>Auto Scroll</td>
<td><strong>Whether to scroll automatically</strong></td>
</tr>
<tr>
<td>Clear</td>
<td><strong>Clear all the messages</strong></td>
</tr>
</tbody>
</table>

### Status Bar Introduction

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Show the load; the two formats - Kbit/s and bit/s will be selected automatically</td>
</tr>
<tr>
<td>S2</td>
<td>Show the current status of the device: Normal, Passive, Bus off</td>
</tr>
<tr>
<td>S3</td>
<td>Show the number of the messages sent</td>
</tr>
<tr>
<td>S4</td>
<td>Show the number of the messages received</td>
</tr>
<tr>
<td>S5</td>
<td>Show the number of Overrun</td>
</tr>
<tr>
<td>S6</td>
<td>Show the number of lines of the data; the maximum number will not exceed the number configured when the device is open</td>
</tr>
</tbody>
</table>
Use Utility to test hardware

An Utility is provided for users to test whether hardware is working normally. It's installed under C:\Program Files\Advantech\AdvCAN\Utility directory by default.

This Utility can be divided into four parts:
In Part 1, users can select which port to open in the choice box at top. Set "Baud Rate", "acceptance filter mode", "acceptance code", "acceptance Mask", and choose if you want to receive "Self Reception" or send "RTR". Users can select "Listen Only" if you want use listen only mode. can also set "Cycle" of data transfer by entering the interval time (ms). Users can select "Auto Increase" if you want the data increase automatically. When setting is finished, users can choose to Open/Close the port, Start Tx/Stop Tx and Start Rx/Stop Rx.
In Part 2, detail information of received data will be shown when communication is built.
In Part 3, the following items will be shown: Numbers of sent frames, timeout sent frames, received frames and timeout received frames. Times of entering PASSIVE error mode, times of entering BUS OFF mode, times of hardware OVERRUN and times of software OVERRUN (BOVERRUN). Users can select ASCII mode. As a result, ASCII format of the data will be shown in Part 2.
In Part 4, ID, Flag (SFF: Standard Frame; EFF: Extended Frame), length and data of messages will be configured before being sent.
### CAN Test Configuration

**Baud Rate:** 125k

**Acceptance Filter:**
- **Code:** `ffffffff`
- **Mask:** `ffffffff`

**Self Reception:**
- **Listen Only:**
- **RTR:**
- **Cycle:** 1000 ms
- **Auto Increase:**

**Open**

<table>
<thead>
<tr>
<th>ID (Hex)</th>
<th>Flag</th>
<th>EFF</th>
<th>Length</th>
<th>Data (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

**Transmission Settings:**

<table>
<thead>
<tr>
<th>SendNum</th>
<th>0</th>
<th>PASSIVE</th>
<th>0</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOut(5s)</td>
<td>0</td>
<td>BUSOFF</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ReceiveNum</td>
<td>0</td>
<td>OVERRUN</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>TimeOut(3s)</td>
<td>0</td>
<td>DOVERUN</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
System Requirements:
CAN_DEMO for CE

Users on CE platform can examine whether the device is working normally through CAN_DEMO program. First, this utility supports the basic configuration programs, including setting up baud rate; acceptance code and acceptance mask. Listen only Mode is also available. Moreover, this utility can achieve the basic “Send and Receive”. Users can configure the sending data, as well as displaying the receiving data.
System Requirements:
Windows CE.
How to dispose message

CAN2.0 Spec. includes CAN 2.0A and CAN 2.0B. CAN 2.0A supports standard 11 bit Identifier. CAN 2.0B supports both 11 bit Identifier and extended 29 bit Identifier. So CAN 2.0B are compatible with CAN 2.0A. SJA1000 supports BasicCAN mode and PeliCAN mode. BasicCAN mode supports CAN 2.0A, while PeliCAN mode supports 2.0B. Advantech CAN Windows WDM&CE Driver runs in PeliCAN mode, thus it can support both Standard frame and Extended frame.

While sending and receiving messages, the user can set or tell the message type via canmsg_t.flags. For detailed information, please refer to here.

The following part introduces how to send and receive CAN messages.

How to send Standard frame, compatible with CAN 2.0A.

```c
DWORD    dwWrite=0;
canmsg_t WriteBuffer;
ZeroMemory(&WriteBuffer, sizeof(canmsg_t));
WriteBuffer.flags = 0; //Standard frame
WriteBuffer.id = 0;
WriteBuffer.length = 8;
for(int i=0; i<8; i++)
{
    WriteBuffer.data[i] = i;
}
WriteFile(hDevice, &WriteBuffer, 1, &dwWrite, 0);
```

How to send Extended frame, compatible with CAN 2.0B.

```c
DWORD    dwWrite=0;
canmsg_t WriteBuffer;
ZeroMemory(&WriteBuffer, sizeof(canmsg_t));
WriteBuffer.flags = MSG_EXT; //Extended frame
WriteBuffer.id = 0;
```
WriteBuffer.length = 8;
for(int i=0; i<8; i++)
{
    WriteBuffer.data[i] = i;
}
WriteFile(hDevice, &WriteBuffer, 1, &dwWrite, 0);

How to send RTR frame.

DWORD dwWrite=0;
canmsg_t WriteBuffer;
ZeroMemory(&WriteBuffer, sizeof(canmsg_t));
WriteBuffer.flags = 0; //Standard frame
//WriteBuffer.flags = MSG_EXT; //Extended frame
WriteBuffer.flags |= MSG_RTR; //Remote frame
WriteBuffer.id = 0;
WriteBuffer.length = 0;
WriteFile(hDevice, &WriteBuffer, 1, &dwWrite, 0);

How to dispose received messages.

DWORD dwRead;
canmsg_t ReadBuffer;
ZeroMemory(&ReadBuffer, sizeof(canmsg_t));
BOOL bSuccess = ReadFile(hDevice, &ReadBuffer, 1, &dwRead, 0);
if(bSuccess)
{
    if(dwRead == 1)
    {
        //SUCCESS
        if(ReadBuffer.flags & MSG_EXT)
        {
            //Extended frame
        }
        else{
            //Standard frame
        }
        if(ReadBuffer.flags & MSG_RTR)
        {
            //Remote frame
        }
        if(ReadBuffer.flags & MSG_SELF)
if (ReadBuffer.flags & MSG_BOVR)
{
  //receive buffer overflow
}
if (ReadBuffer.flags & MSG_BUSOFF)
{
  //CAN controller bus off
}
if (ReadBuffer.flags & MSG_OVR)
{
  //CAN controller Msg overflow error
}
if (ReadBuffer.flags & MSG_PASSIVE)
{
  //CAN controller in error passive
}
else{
  //Timeout
}
else{
  DWORD dwError = GetLastError();

  if (dwError == ERROR_IO_PENDING)
  {
    //pending
  }
  else if (dwError == ERROR_INVALID_PARAMETER)
  {
    //parameter error
  }
  else if (dwError == ERROR_OPERATION_ABORTED)
  {
    //cancelled
  }
  else if (dwError == ERROR_GEN_FAILURE)
  {
    //bus off
  }
}
}
canmsg_t
WriteFile
ReadFile
**Acceptance filtering**

Acceptance code corresponds to 4 registers: ACR0, ACR1, ACR2, ACR3. Acceptance mask corresponds to 4 registers: AMR0, AMR1, AMR2, AMR3. ACR works with AMR. Only when the acceptance filtering of the Standard frame or the Extended frame is confirmed, will the filter save the data of the frame into FIFO.

For AMR bits,

- 0: the corresponding bits of ACR and CAN information frames should be the same in order to be accepted.

- 1: the acceptance filtering function of the corresponding ACR bits is disabled and the corresponding bits of the CAN information frame are independent of the acceptance result.

**Single mode**

For Standard frame, 11 identifiers and the RTR bit correspond to ACR0, ACR1, AMR0 and AMR1 with unused lower 4 bits of ACR1 and AMR1. Since the Standard frame has 11 identifiers only, the first 2 bytes of the data field are also included for filtering. Data1 corresponds to ACR2, AMR2; Data2 corresponds to ACR3, AMR3. If the received Standard frame is a Remote frame and the RTR bit is 1, then only the identifier will be used for filtering. If the received Standard frame is a data frame and the data field is less than 2 bytes, then the missed data will not be used for filtering. For Extended frame, 29 identifiers and the RTR bit correspond to 4 ACRs and 4 AMRs. Please note that the lower 2 bits of ACR3 and AMR3 are reserved.

- Example of Standard frame:

<table>
<thead>
<tr>
<th>0</th>
<th>1 (lower 4 bits)</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR</td>
<td>11XXX010</td>
<td>XXXX</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>ACM</td>
<td>00111000</td>
<td>1111</td>
<td>11111111</td>
</tr>
<tr>
<td>accepted ID (ID.10 ` ID.0)</td>
<td>11XXX010</td>
<td>XXX</td>
<td></td>
</tr>
</tbody>
</table>
In this example, Data1, Data2 and RTR are not considered and the accepted IDs include: 0x7D7(11111010111), 0x610(11000010000), etc.

- Example of Extended frame:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR</td>
<td>10010100</td>
<td>1011000X</td>
<td>1100XXXX</td>
<td>00110XXX</td>
</tr>
<tr>
<td>ACM</td>
<td>00000000</td>
<td>00000001</td>
<td>00001111</td>
<td>00000111</td>
</tr>
<tr>
<td>accepted ID(ID.28 ` ID.0)</td>
<td>10010100</td>
<td>1011000X</td>
<td>1100XXXX</td>
<td>00110XXX</td>
</tr>
</tbody>
</table>

In this example, RTR is not considered and the accepted IDs include: 0x12961806(10010100101100001100000000110), 0x129639E6(10010100101100011100111100110), etc.

**Dual Mode**

Dual filtering is more complicated than single filtering. In Dual filtering mode, 4 ACRs and 4 AMRs form two filters; the received information frame could be accepted by either filter. For Standard frame, filter1 is composed of ACR0, ACR1, AMR0, AMR1 and the lower four bits of ACR3 and AMR3. It corresponds to 20 bits, including 11 identifiers, the RTR bit and the first byte; all of 20 bits are used for the filtering. filter2 is composed of ACR2, AMR2 and the higher four bits of ACR3 and AMR3, while only 11 identifiers and the RTR bit are used for the filtering. For Extended frame, filter1 is composed of ACR0, ACR1, AMR0 and AMR1, not including ACR3 and AMR3. What’s more, there are only 16 bits out of 29 bits used for the filtering. Filter2 is composed of ACR2, ACR3, AMR2 and AMR3. The higher 16 bits out of 29 bits are used for the filtering.

- Example of Standard frame:

Filter1:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>3 (lower 4 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR</td>
<td>000000000</td>
<td>001XXXXX</td>
<td>XXXX</td>
</tr>
<tr>
<td>ACM</td>
<td>000000000</td>
<td>00011111</td>
<td>1111</td>
</tr>
</tbody>
</table>
In this example, Data1 and RTR are not considered and the accepted ID: 0x1.

Filter2:

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3 (upper 4 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR0-ACR3</td>
<td>00000000</td>
<td>001X</td>
</tr>
<tr>
<td>ACM0-ACM3</td>
<td>00000000</td>
<td>0001</td>
</tr>
<tr>
<td>accepted</td>
<td>00000000</td>
<td>001</td>
</tr>
</tbody>
</table>

In this example, RTR are not considered and the accepted ID: 0x1.

- Example of Extended frame:

Filter1:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACR</td>
<td>00000000</td>
<td>00000001</td>
</tr>
<tr>
<td>ACM</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>accepted ID(ID.28 ` ID.13)</td>
<td>00000000</td>
<td>00000001</td>
</tr>
</tbody>
</table>

In this example, the accepted IDs include: 0x2000(10000000000000), 0x3FFF(11111111111111), etc.
In this example, the accepted IDs include: \(0x2000(10000000000000)\), \(0x3FFF(11111111111111)\), etc.
The COTI DLL allows users to use the CANopen Conformance Test Tool of 
CiA (CAN in Automation) with Advantech CAN WDM Driver. This file has to be 
copied into the directory of the CANopen Conformance Test. More detailed 
presentation about CANopen Conformance Test Tool, please refer to here. 
The default installation path for COTI.DLL is C:\Program 
Files\Advantech\AdvCAN\COTI.
System Requirements:
Users can directly access drivers with **WINDOWS Native API**. In the following, we will provide an example by opening a CAN port and reading its current status to explain how to write basic applications in VC, VB, VB.NET and C# environment. Necessary files for developing applications are listed below. Suppose installation paths of all header files in the example are C:\Program Files\ADVANTECH\AdvCAN\Include.

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdvCan.h</td>
<td>Header file for VC</td>
</tr>
<tr>
<td>AdvCan.cs</td>
<td>Header file for C#</td>
</tr>
<tr>
<td>AdvCan.vb</td>
<td>Header file for VB.NET</td>
</tr>
<tr>
<td>AdvCan.bas</td>
<td>Header file for VB</td>
</tr>
</tbody>
</table>
Note:
Users who use CAN driver on windows CE can directly refer to Structure list and Introduction to API (GetOverlappedResult is not included).
Structure List

The table below is a list of structures needed by Advantech CAN WDM Driver.
Methods:

<table>
<thead>
<tr>
<th>canmsg_t</th>
<th>Received/Sent message structure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanStatusPar_t</td>
<td>Port status structure.</td>
</tr>
<tr>
<td>Command_par</td>
<td>Command/Configure operation structure.</td>
</tr>
</tbody>
</table>
canmsg_t

When users directly use ReadFile or WriteFile interface of Windows Native API to call drivers, this structure is needed.

```c
typedef struct {
    int flags;
    int cob;
    ULONG id;
    short int length;
    UCHAR data[8];
} canmsg_t;
```

**Member Description**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>Types of messages.</td>
</tr>
<tr>
<td>cob</td>
<td>Reserved.</td>
</tr>
<tr>
<td>id</td>
<td>ID of message.</td>
</tr>
<tr>
<td>length</td>
<td>Length of messages (0~8).</td>
</tr>
<tr>
<td>data</td>
<td>Data transferred (Made up of 0~8 Byte data).</td>
</tr>
</tbody>
</table>

**Remarks**

8-bit flags are used to record types of messages during sending or receiving. The meanings of these bits are:

<table>
<thead>
<tr>
<th>Bit Declaration</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 MSG_RTR</td>
<td>RTR</td>
<td>1 means Remote frame, 0 means data frame.</td>
</tr>
<tr>
<td>1 MSG_OVR</td>
<td>Hardware OVERRUN</td>
<td>1 means receive buffer overrun of hardware.</td>
</tr>
<tr>
<td>2 MSG_EXT</td>
<td>Extension</td>
<td>1 means Extended frame(29 bit identifier), 0 means Standard frame(11 bit identifier).</td>
</tr>
</tbody>
</table>

ID shows type of CAN. When ID is 0xFFFFFFFF, it means error frame, which
If ID does not equal 0xFFFFFFFF, flags may be Remote frame, Extended frame, Standard frame, Self Reception or software overrun.
DeviceIOControl's parameter dwIoControlCode is CAN_IOCTL_STATUS (0x222554). It uses this structure.

```c
typedef struct {
    unsigned int baud;
    unsigned int status;
    unsigned int error_warning_limit;
    unsigned int rx_errors;
    unsigned int tx_errors;
    unsigned int error_code;
    unsigned int rx_buffer_size;
    unsigned int rx_buffer_used;
    unsigned int tx_buffer_size;
    unsigned int tx_buffer_used;
    ULONG            retval;
    unsigned int type;
    unsigned int acceptancecode;
    unsigned int acceptancemask;
    unsigned int acceptancemode;
    unsigned int selfreception;
    unsigned int readtimeout;
    unsigned int writetimeout;
} CanStatusPar_t, *PCcanStatusPar_t;
```

### Member Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baud</td>
<td>Actual bit rate.</td>
</tr>
<tr>
<td>status</td>
<td>The status register (SR); CAN address 2.</td>
</tr>
<tr>
<td>error_warning_limit</td>
<td>The error warning limit register (EWLR); CAN address 13.</td>
</tr>
<tr>
<td>rx_errors</td>
<td>The RX error counter register (RXERR); CAN address 14.</td>
</tr>
<tr>
<td>tx_errors</td>
<td>The TX error counter register (TXERR); CAN address 15.</td>
</tr>
<tr>
<td>error_code</td>
<td>The error code capture register (ECC); CAN address 12.</td>
</tr>
<tr>
<td>rx_buffer_size</td>
<td>Size of rx buffer.</td>
</tr>
<tr>
<td>rx_buffer_used</td>
<td>Number of received messages.</td>
</tr>
<tr>
<td>tx_buffer_size</td>
<td>Size of tx buffer for wince.</td>
</tr>
<tr>
<td>tx_buffer_used</td>
<td>Number of messages of tx buffer for wince.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>retval</td>
<td>Return Value. 0, SUCCESS; 0xFFFFFFFF, FAIL.</td>
</tr>
<tr>
<td>type</td>
<td>CAN controller type. 1, SJA1000; 0 other device.</td>
</tr>
<tr>
<td>acceptancecode</td>
<td>Acceptance code.</td>
</tr>
<tr>
<td>acceptancemask</td>
<td>Acceptance mask.</td>
</tr>
<tr>
<td>acceptancemode</td>
<td>Acceptance Filter Mode: 1:Single 0:Dual.</td>
</tr>
<tr>
<td>selfreception</td>
<td>Self reception.</td>
</tr>
<tr>
<td>ReadTimeout</td>
<td>Read timeout.</td>
</tr>
<tr>
<td>WriteTimeout</td>
<td>Write timeout.</td>
</tr>
</tbody>
</table>
Command_par

When users directly use DeviceIOControl interface of Windows Native API to call drivers, this structure is needed.

If DeviceIOControl's parameter dwIoControlCode is CAN_IOCTL_COMMAND (0x222540), this means command operation. If dwIoControlCode is CAN_IOCTL_CONFIG (0x222544), it means configuration operation. (Please refer to MSDN for related information about DeviceIOControl).

```c
struct Command_par {
    int cmd;
    int target;
    ULONG val1;
    ULONG val2;
    int error;
    ULONG retval;
};
```

Note

1. The two modes of configuring Baud Rate are standard mode and custom mode.

   - Standard mode

     The standard mode includes the recommended Baud Rate value. If the setting value is 10, then the Baud Rate will be 10k.

     | Target value | BTR0  | BTR1  | Setting value |
     |--------------|------|-------|---------------|
     | 10K          | 0x31 | 0x1c  | 10            |
     | 20K          | 0x18 | 0x1c  | 20            |
     | 50K          | 0x09 | 0x1c  | 50            |
     | 100K         | 0x04 | 0x1c  | 100           |
     | 125K         | 0x03 | 0x1c  | 125           |
     | 250K         | 0x01 | 0x1c  | 250           |
     | 500K         | 0x00 | 0x1c  | 500           |
     | 800K         | 0x00 | 0x16  | 800           |
     | 1000K        | 0x00 | 0x14  | 1000          |

   - Custom mode

     If user need the baud rate is not in above table, user can use the custom mode to set the custom baud rate.
The CAN port's baud rate is determined by CAN clock, BTR0 and BTR1.

The Advantech CAN devices use 8MHz clock and oscillator frequency is 16M, the internal clock is divided by two from the oscillator frequency.

BTR0 and BTR1 Timing Registers are used for:
- defining the bit-rate on the bus.
- defining the sample point in a bit period (bit sample point).
- defining the number of samples taken in a bit period.

We provide API function to set BTR0 and BTR1. The function will write the low 8 bits of setting value to BTR0 and high 8 bits of setting value to BTR1 of CAN device register.

The following code is the example to custom baud rate by setting BTR0 and BTR1 device register.

```
AdvCANIO.h Line:269
/*
 * acSetBaudRegister
 *
 * Purpose:
 * Configures baud rate by custom mode.
 *
 * Arguments:
 * hDevice - handle of device
 * Btr0 - BTR0 register value.
 * Btr1 - BTR1 register value.
 * Returns:
 * =0 SUCCESS; or <0 failure
 *
***************************************************************************/
int acSetBaudRegister(HANDLE hDevice, unsigned char Btr0, unsigned char Btr1)
{
    unsigned int BaudRateValue = Btr0 * 256;
    BaudRateValue += Btr1;
    return acSetBaud(hDevice, BaudRateValue);
}
```

```
can_receive.cpp Line:105
nRet = acSetBaudRegister( hDevice, byBtr0, byBtr1 ); //Set baud
if ( nRet < 0 )
{
    SetDlgItemText( hwnd, IDC_SHOW_RESULT, "Failed to set Baud ");
```

User can also refer to receive and send examples for details usage.

How to calculate BTR0 and BTR1, please refer to SJA1000 datasheet for details.

Here we show some value of BTR0 and BTR1 for some custom baud rate.

<table>
<thead>
<tr>
<th>Target value</th>
<th>BTR0</th>
<th>BTR1</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>0xBF</td>
<td>0xFF</td>
<td>0xFFFBF</td>
</tr>
<tr>
<td>40K</td>
<td>0x87</td>
<td>0xFF</td>
<td>0xFF87</td>
</tr>
<tr>
<td>80K</td>
<td>0x83</td>
<td>0xFF</td>
<td>0xFF83</td>
</tr>
<tr>
<td>200K</td>
<td>0x81</td>
<td>0xFA</td>
<td>0xFA81</td>
</tr>
<tr>
<td>400K</td>
<td>0x80</td>
<td>0xFA</td>
<td>0xFA80</td>
</tr>
</tbody>
</table>

2. Please refer to [acceptance filtering](#) for setting acceptance code and acceptance mask.

3. Self reception is not supported on windows CE.

**Member Description**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>Send start or stop command to drivers.</td>
</tr>
<tr>
<td>target</td>
<td>Send configure command to drivers.</td>
</tr>
<tr>
<td>val1</td>
<td>Parameter 1.</td>
</tr>
<tr>
<td>val2</td>
<td>Parameter 2.</td>
</tr>
<tr>
<td>error</td>
<td>Reserved.</td>
</tr>
<tr>
<td>retval</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**Remarks**

1. While configuring cmd, the following commands are supported:

<table>
<thead>
<tr>
<th>Optional commands</th>
<th>Corresponding value of cmd</th>
<th>Description</th>
<th>val1</th>
<th>val2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_START</td>
<td>1</td>
<td>Start chip and enter work mode</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
2. While configuring target, the following commands are supported:

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_STOP</td>
<td>2</td>
<td>Stop chip and enter reset mode.</td>
</tr>
<tr>
<td>CMD_RESET</td>
<td>3</td>
<td>Stop chip by canceling current send/receive operation and enter reset mode.</td>
</tr>
<tr>
<td>CMD_CLEARBUFFERS</td>
<td>4</td>
<td>Clear receive buffer.</td>
</tr>
<tr>
<td>CONF_ACC</td>
<td>0</td>
<td>Acceptance code register and acceptance mask register. Need to enter reset mode.</td>
</tr>
<tr>
<td>CONF_ACCM</td>
<td>1</td>
<td>Acceptance mask only. Need to enter reset mode.</td>
</tr>
<tr>
<td>CONF_ACCC</td>
<td>2</td>
<td>Acceptance code only. Need to enter reset mode.</td>
</tr>
<tr>
<td>CONF_TIMING</td>
<td>3</td>
<td>Bit timing. Need to enter reset mode.</td>
</tr>
<tr>
<td>CONF_LISTEN_ONLY_MODE</td>
<td>8</td>
<td>Listen only mode. Need to enter reset mode.</td>
</tr>
<tr>
<td>Configuration</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CONF_SELF_RECEPTION</td>
<td>9</td>
<td>Self reception.</td>
</tr>
<tr>
<td>CONF_TIMEOUT</td>
<td>13</td>
<td>Configure read and write timeout.</td>
</tr>
<tr>
<td>CONF_ACC_FILTER</td>
<td>20</td>
<td>Acceptance filter mode: 1-Single, 0-Dual. Need to enter reset mode.</td>
</tr>
</tbody>
</table>
Introduction to API

Main API used in current development are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateFile</td>
<td>Open specified Can port.</td>
</tr>
<tr>
<td>CloseHandle</td>
<td>Close specified Can port.</td>
</tr>
<tr>
<td>DeviceIoControl</td>
<td>Send commands to drivers, including configuration, management and getting status, etc.</td>
</tr>
<tr>
<td>ReadFile</td>
<td>Read data.</td>
</tr>
<tr>
<td>WriteFile</td>
<td>Send data.</td>
</tr>
<tr>
<td>GetOverLappedResult</td>
<td>Wait until asynchronous operation is finished.</td>
</tr>
<tr>
<td>SetCommMask</td>
<td>Set mask.</td>
</tr>
<tr>
<td>GetCommMask</td>
<td>Get mask.</td>
</tr>
<tr>
<td>WaitCommEvent</td>
<td>Wait specified event.</td>
</tr>
<tr>
<td>ClearCommError</td>
<td>Get error code with this function when receiving error event.</td>
</tr>
</tbody>
</table>

Only brief introduction is given in this manual regarding detailed usage of each function. Notes are made to notify users of important operation. For more detailed information about the usage, please see MSDN.
CreateFile

Users can use this interface to open CAN port device. Close the port by calling `CloseHandle` when operation is completed.
Note
This driver does not support share open function, so the third parameter must be set to 0.
HANDLE CreateFile(
    LPCTSTR  lpFileName,
    DWORD   dwDesiredAccess,
    DWORD   dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD   dwCreationDisposition,
    DWORD   dwFlagsAndAttributes,
    HANDLE  hTemplateFile
);
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpFileName</td>
<td>Input</td>
<td>Name of device which was opened, such as <code>\\.\\\CAN1</code>. *Note In WINDOWS CE, use CAN1:.</td>
</tr>
<tr>
<td>ldwDesiredAccess</td>
<td>Input</td>
<td>Ways of opening the port, which is usually `GENERIC_READ</td>
</tr>
<tr>
<td>dwShareMode</td>
<td>Input</td>
<td>Does not support share open. It must be set to 0.</td>
</tr>
<tr>
<td>lpSecurityAttributes</td>
<td>Input</td>
<td>NULL.</td>
</tr>
<tr>
<td>dwCreationDisposition</td>
<td>Input</td>
<td>OPEN_EXISTING.</td>
</tr>
<tr>
<td>dwFlagsAndAttributes</td>
<td>Input</td>
<td>Synchronous open: <code>FILE_ATTRIBUTE_NORMAL</code>. Asynchronous open: `FILE_ATTRIBUTE_NORMAL</td>
</tr>
<tr>
<td>hTemplateFile</td>
<td>Input</td>
<td>NULL.</td>
</tr>
</tbody>
</table>
Return Value
Successful, return effective HANDLE. Unsuccessful, return INVALID_HANDLE_VALUE.
Example

Synchronous open CAN1:
HANDLE hDevice = CreateFile(
    "\\.\\CAN1",
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,
    NULL);

Asynchronous open CAN1:
HANDLE hDevice = CreateFile(
    "\\.\\CAN1",
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
    NULL);
CloseHandle

Close the port by calling this function when operation is completed.
BOOL CloseHandle(
    HANDLE hDevice
);

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
</tbody>
</table>
Return Value
Successful: return non-zero values. Unsuccessful: return zero value. Please call GetLastError function.
Example

// close
BOOL bRet = CloseHandle(hDevice);
See Also
CreateFile
DeviceIoControl

Users can use this interface to send commands, configure ports, like stop device, start device, configure Baud Rate, etc. See [Command_par](#), [CanStatusPar_t](#) for detailed configuration.
BOOL DeviceIoControl(
    HANDLE hDevice,
    DWORD dwIoControlCode,
    LPVOID lpInBuffer,
    DWORD nInBufferSize,
    LPVOID lpOutBuffer,
    DWORD nOutBufferSize,
    LPDWORD lpBytesReturned,
    LPOVERLAPPED lpOverlapped
);
<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>dwIoControlCode</td>
<td>Input</td>
<td>Control code, shows the specific operation mode.</td>
</tr>
<tr>
<td>lpInBuffer</td>
<td>Input</td>
<td>Start address sent to data area of driver.</td>
</tr>
<tr>
<td>nInBufferSize</td>
<td>Input</td>
<td>Byte length sent to data area of driver.</td>
</tr>
<tr>
<td>lpOutBuffer</td>
<td>Output</td>
<td>Address of data area which receives driver's return data.</td>
</tr>
<tr>
<td>nOutBufferSize</td>
<td>Input</td>
<td>Byte length of data area which receives driver's return data.</td>
</tr>
<tr>
<td>lpBytesReturned</td>
<td>Output</td>
<td>Real byte length of data received from driver.</td>
</tr>
<tr>
<td>lpOverlapped</td>
<td>Input</td>
<td>If <a href="https://msdn.microsoft.com/en-us/library/windows/desktop/aa366850(v=vs.85).aspx">CreateFile</a> uses asynchronous open, asynchronous operation must be supported here. Please refer to MSDN for detailed instruction to asynchronous open. <em>Note In WINDOWS CE, set to NULL.</em></td>
</tr>
</tbody>
</table>
Return Value
Successful: return non-zero values. Unsuccessful: return zero value. Please call GetLastError function.

It will be considered unsuccessful if zero values are returned in the following situation. Please call GetLastError.
In work mode, drivers can not complete some of the requests, GetLastError will be called to return ERROR_GEN_FAILURE.
Example

- *Set baud rate:*

```c
DWORD dwReturned;
Command_par_t cmd;
Config_par_t config;
cmd.cmd = CMD_STOP;

BOOL bSuccess = DeviceIoControl (  
    hDevice, 
    CAN_IOCTL_COMMAND, 
    &cmd, 
    sizeof(Command_par), 
    NULL, 
    0, 
    &dwReturned, 
    NULL 
);
if(!bSuccess)
{
    //error
}

config.target = CONF_TIMING;
config.val1 = 1000; //set 1000K

bSuccess = DeviceIoControl (  
    hDevice, 
    CAN_IOCTL_CONFIG, 
    &config, 
    sizeof(Command_par), 
    NULL, 
    0, 
    &dwReturned, 
    NULL 
);
if(!bSuccess)
{
    //error
}

cmd.cmd = CMD_START;
bSuccess = DeviceIoControl (  
    hDevice, 
    CAN_IOCTL_COMMAND, 
    &cmd,
```
Set acceptance filter:

DWORD dwReturned;
Command_par_t cmd;
Config_par_t config;
cmd.cmd = CMD_STOP;

BOOL bSuccess = DeviceIoControl (  
    hDevice,  
    CAN_IOCTL_COMMAND,  
    &cmd,  
    sizeof(Command_par),  
    NULL,  
    0,  
    &dwReturned,  
    NULL  
);
if(!bSuccess)  
{  
    //error  
}

config.target = CONF_ACC_FILTER;
config.val1 = 1;  //1: set single filter mode; 0: set dual filter

bSuccess = DeviceIoControl (  
    hDevice,  
    CAN_IOCTL_CONFIG,  
    &config,  
    sizeof(Command_par),  
    NULL,  
    0,  
    &dwReturned,  
    NULL  
);
if(!bSuccess)  
{  
    //error  
}
config.target = CONF_ACC;
config.val1 = 0xffffffff;
config.val2 = 0xffffffff;
bSuccess = DeviceIoControl (hDevice,
CAN_IOCTL_CONFIG,
&config,
sizeof(Command_par),
NULL,
0,
&dwReturned,
NULL);
if(!bSuccess)
{
    //error
}

cmd.cmd = CMD_START;
bSuccess = DeviceIoControl (hDevice,
CAN_IOCTL_COMMAND,
&cmd,
sizeof(Command_par),
NULL,
0,
&dwReturned,
NULL);
if(!bSuccess)
{
    //error
}

• Set listen only mode:

DWORD dwReturned;
Command_par_t cmd;
Config_par_t config;
cmd.cmd = CMD_STOP;

BOOL bSuccess = DeviceIoControl (hDevice,
CAN_IOCTL_COMMAND,
&cmd,
sizeof(Command_par),
NULL,
0,
&dwReturned,
NULL
);
if(!bSuccess)
{
    //error
}

config.target = CONF_LISTEN_ONLY_MODE;
config.val1 = 1; //1: ON; 0: OFF
bSuccess = DeviceIoControl ( 
    hDevice,
    CAN_IOCTL_CONFIG,
    &config,
    sizeof(Command_par),
    NULL,
    0,
    &dwReturned,
    NULL
);
if(!bSuccess)
{
    //error
}

cmd.cmd = CMD_START;
bSuccess = DeviceIoControl ( 
    hDevice,
    CAN_IOCTL_COMMAND,
    &cmd,
    sizeof(Command_par),
    NULL,
    0,
    &dwReturned,
    NULL
);
if(!bSuccess)
{
    //error
}

• Reset chip:
DWORD dwReturned;
Command_par_t cmd;

cmd.cmd = CMD_RESET;

BOOL bSuccess = DeviceIoControl (    
hDevice,    
CAN_IOCTL_COMMAND,    
&cmd,    
sizeof(Command_par),    
NULL,    
0,    
&dwReturned,    
NULL    
);
if(!bSuccess)    
{
   //error
}

cmd.cmd = CMD_START;
bSuccess = DeviceIoControl (    
hDevice,    
CAN_IOCTL_COMMAND,    
&cmd,    
sizeof(Command_par),    
NULL,    
0,    
&dwReturned,    
NULL    
);
if(!bSuccess)    
{
   //error
}

• Clear receive buffer:

DWORD dwReturned;
Command_par_t cmd;

// Command_par_t cmd;
cmd.cmd = CMD_CLEARBUFFERS;

BOOL bSuccess = DeviceIoControl (    
hDevice,    
CAN_IOCTL_COMMAND,    
&cmd,    
sizeof(Command_par),    
NULL,    
0,    
&dwReturned,    
NULL    
);
DWORD dwReturned;
CanStatusPar_t status;

BOOL bSuccess = DeviceIoControl ( 
    hDevice, 
    CAN_IOCTL_STATUS, 
    NULL, 
    0, 
    &status, 
    sizeof(CanStatusPar_t), 
    &dwReturned, 
    NULL 
);
See Also

Command_par
CanStatusPar_t
ReadFile

Users can use this interface to read data from CAN port which was opened. One or more frames can be selected each time.
Note
The third and fourth parameters of ReadFile are defined as byte length in MSDN, but stand for the number of frames in Advantech CAN Windows WDM&CE Driver.
BOOL ReadFile(
    HANDLE hDevice,
    LPVOID lpBuffer,
    DWORD nNumberOfFramesToRead,
    LPDWORD lpNumberOfFramesRead,
    LPOVERLAPPED lpOverlapped
);
### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>lpBuffer</td>
<td>Output</td>
<td>Start address of receive buffer.</td>
</tr>
<tr>
<td>nNumberOfFramesToRead</td>
<td>Input</td>
<td>Number of frames to be received from drivers. The definition here is different from that in MSDN.</td>
</tr>
<tr>
<td>lpNumberOfFramesRead</td>
<td>Output</td>
<td>Real number of frames received from the driver. The definition here is different from that in MSDN.</td>
</tr>
<tr>
<td>lpOverlapped</td>
<td>Input</td>
<td>If <a href="#">CreateFile</a> uses asynchronous operation, an asynchronous operation must be supported here. Please refer to MSDN for details on instruction to asynchronous open. *Note In WINDOWS CE, set to NULL.</td>
</tr>
</tbody>
</table>
Return Value

It will be considered successful if non-zero values are returned in the following situations:
1. The driver reads data and there is no data in the receive buffer. At this time, the data received is less than the requested data.
2. The driver reads all the requested data.
3. The driver cannot read any data and time is out.

It will be considered unsuccessful if zero values are returned in the following situations. Please call GetLastError.
If users cancel the operation or reset chip while drivers are receiving data, GetLastError will be called to return ERROR_OPERATION_ABORTED.
If busoff of device is discovered before drivers read any frames, GetLastError will be called to return ERROR_GEN_FAILURE.
If drivers cannot allocate resources according to the number defined by the third parameter frame, GetLastError will be called to return ERROR_INVALID_PARAMETER.
In asynchronous mode, operation will be pending if drivers cannot complete user's read request, and GetLastError will be called to return ERROR_IO_PENDING.
See MSDN for more information.
Example
Read one frame data in synchronous mode.

DWORD dwRead;
canmsg_t ReadBuffer;
ZeroMemory(&ReadBuffer, sizeof(canmsg_t));
BOOL bSuccess = ReadFile(hDevice, &ReadBuffer, 1, &dwRead, 0);
if(bSuccess)
{
    if(dwRead == 1)
    {
        //SUCCESS
        if(ReadBuffer.flags & MSG_EXT)
        {
            //Extended frame
        }
        else{
            //Standard frame
        }
        if(ReadBuffer.flags & MSG_RTR)
        {
            //Remote frame
        }
        if(ReadBuffer.flags & MSG_SELF)
        {
            //self reception
        }
        if(ReadBuffer.flags & MSG_BOVR)
        {
            //receive buffer overflow
        }
        if(ReadBuffer.flags & MSG_BUSOFF)
        {
            //CAN controller bus off
        }
        if(ReadBuffer.flags & MSG_OVR)
        {
            //CAN controller Msg overflow error
        }
        if(ReadBuffer.flags & MSG_PASSIVE)
        {
            //CAN controller in error passive
        }
    }
    else{
        //Timeout
    }
}
else{
    DWORD dwError = GetLastError();

    if (dwError == ERROR_IO_PENDING)
    {
        // pending
    }
    else if (dwError == ERROR_INVALID_PARAMETER)
    {
        // parameter error
    }
    else if (dwError == ERROR_OPERATION_ABORTED)
    {
        // cancelled
    }
    else if (dwError == ERROR_GEN_FAILURE)
    {
        // bus off
    }
}
WriteFile

Users can use this interface to send data to CAN port which was opened. One or more frames can be selected each time.
Note
The third and fourth parameters of WriteFile are defined as byte length in MSDN, but stand for the number of frames here.
BOOL WriteFile(
    HANDLE hDevice,
    LPCVOID lpBuffer,
    DWORD nNumberOfFramesToWrite,
    LPDWORD lpNumberOfFramesWritten,
    LPOVERLAPPED lpOverlapped
);
<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>lpBuffer</td>
<td>Input</td>
<td>Start address of send buffer.</td>
</tr>
<tr>
<td>nNumberOfFramesToWrite</td>
<td>Input</td>
<td>Number of frames to be sent to drivers. The definition here is different from that in MSDN.</td>
</tr>
<tr>
<td>lpNumberOfFramesWritten</td>
<td>Output</td>
<td>Real number of frames sent to drivers. The definition here is different from that in MSDN.</td>
</tr>
<tr>
<td>lpOverlapped</td>
<td>Input</td>
<td>If <a href="http://msdn.microsoft.com">CreateFile</a> uses asynchronous operation, asynchronous operation must be supported here. Please refer to MSDN for detailed instruction to asynchronous open.</td>
</tr>
</tbody>
</table>
Return Value

It will be considered successful if non-zero values are returned in the following situations:
- Driver send all the requested data.
- Timeout occur when driver only send part of the data or no data.

It will be considered unsuccessful if zero values are returned in the following situations. Please call GetLastError.
- If user cancel the operation or reset chip while drivers are sending data, GetLastError will be called to return ERROR_OPERATION_ABORTED.
- If busoff of device is discovered before drivers send any frames, GetLastError will be called to return ERROR_GEN_FAILURE.
- If drivers cannot allocate resources according to the number defined by the third parameter frame, GetLastError will be called to return ERROR_INVALID_PARAMETER.
- In asynchronous mode, operation will be pending if drivers cannot complete user's write request at present, and GetLastError will be called to return ERROR_IO_PENDING.

See MSDN for more information.
Example

Write one frame data in synchronous mode.

```c
DWORD dwWrite;
canmsg_t WriteBuffer;
ZeroMemory(&WriteBuffer, sizeof(canmsg_t));

//WriteBuffer.flags = 0;  //Standard frame
WriteBuffer.flags = MSG_EXT;  //Extended frame
//WriteBuffer.flags |= MSG_RTR;  //Remote frame

WriteBuffer.id = 0;
WriteBuffer.length = DATALENGTH;

for(int i=0; i<DATALENGTH; i++)
{
    WriteBuffer.data[i] = i;
}

BOOL bSuccess = WriteFile(hDevice, &WriteBuffer, 1, &dwWrite, 0);
if(bSuccess)
{
    if(dwWrite == 1)
    {
        //SUCCESS
    }
    else{
        //Timeout
    }
}
else{
    DWORD dwError = GetLastError();

    if (dwError == ERROR_IO_PENDING)
    {
        //pending
    }
    else if(dwError == ERROR_INVALID_PARAMETER)
    {
        //parameter error
    }
    else if(dwError == ERROR_OPERATION_ABORTED)
    {
        //cancelled
    }
    else if(dwError == ERROR_GEN_FAILURE)
    {
```
// bus off
}
}
SetCommMask

Users can use this interface to set event for CAN port. Users have to call \texttt{WaitCommEvent} function to wait event. Users can call \texttt{GetCommMask} to get the current event set before. If users do not set any of the supported event types, the real event type will be zero. In this case, if \texttt{WaitCommEvent} is called, the call will be returned and the event type returned will be zero.
Note
Events supported at present are only:

<table>
<thead>
<tr>
<th>Type</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV_ERR</td>
<td>Error</td>
</tr>
<tr>
<td>EV_RXCHAR</td>
<td>Receive one frame data</td>
</tr>
</tbody>
</table>
Syntax

BOOL SetCommMask(
    HANDLE hDevice,
    DWORD dwEvtMask
);
## Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>dwEvtMask</td>
<td>Input</td>
<td>Event type.</td>
</tr>
</tbody>
</table>
Return Value
Successful: return non-zero values. Unsuccessful: return zero value. Please call GetLastError function.
Example

```c
BOOL bSuccess = SetCommMask(hDevice, EV_ERR | EV_RXCHAR);
```
See Also
WaitCommEvent
GetCommMask
GetCommMask

Users can call GetCommMask to get event type set in SetCommMask.
Syntax

BOOL GetCommMask(
    HANDLE hDevice,
    LPDWORD lpEvtMask
);
<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>lpEvtMask</td>
<td>Output</td>
<td>Store event type which return from drivers.</td>
</tr>
</tbody>
</table>
Return Value
Successful: return non-zero values. Unsuccessful: return zero value. Please call GetLastError function.
Example

DWORD dwMask = 0;
BOOL bSuccess = GetCommMask(hDevice, &dwMask);
See Also

SetCommMask
WaitCommEvent

After calling `SetCommMask` to set event, users should also call `WaitCommEvent` function to wait event.
Syntax

```c
BOOL WaitCommEvent(
    HANDLE hDevice,
    LPDWORD lpEvtMask,
    LPOVERLAPPED lpOverlapped
);
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>lpEvtMask</td>
<td>Output</td>
<td>Event type.</td>
</tr>
<tr>
<td>lpOvelapped</td>
<td>Output</td>
<td>If CreateFile uses asynchronous open, asynchronous operation must be supported here. Please refer to MSDN for detailed instruction to asynchronous open. *Note In WINDOWS CE, set to NULL.</td>
</tr>
</tbody>
</table>
Return Value
Successful: return non-zero values. Unsuccessful: return zero value. Please call GetLastError function.
If users cancel the operation or reset chip, GetLastError will be called to return ERROR_OPERATION_ABORTED.
In asynchronous mode, operation will be pending if drivers cannot complete user's request, and GetLastError will be called to return ERROR_IO_PENDING. See MSDN for more information.
Example
Wait event in synchronous mode.

```c
BOOL bSuccess = SetCommMask(hDevice, EV_ERR | EV_RXCHAR);
if(!bSuccess)
{
    //error
}
DWORD dwMask = 0;
bSuccess = WaitCommEvent(hDevice, &dwMask, NULL);
if(bSuccess)
{
    if(dwMask & EV_ERR)
    {
        //to do
        DWORD dwError;
        ClearCommError(hDevice, &dwError, NULL);
    }
    if(dwMask & EV_RXCHAR)
    {
        //to do
    }
}
```
See Also

SetCommMask
ClearCommError
ClearCommError

When error occurs, users can use ClearCommError to get the specific type of error.
Note

Definitions of error codes supported are as below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE_RXOVER</td>
<td>Receive Queue overflow.</td>
</tr>
<tr>
<td>CE_OVERRUN</td>
<td>Receive Overrun Error.</td>
</tr>
<tr>
<td>CE_FRAME</td>
<td>Passive error</td>
</tr>
<tr>
<td>CE_BREAK</td>
<td>Busoff</td>
</tr>
</tbody>
</table>

The third parameter is unused, please set it to NULL.
BOOL ClearCommError(
    HANDLE hDevice,
    LPDWORD lpErrors,
    LPCOMSTAT lpStat
);
<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>lpErrors</td>
<td>Output</td>
<td>Store error codes which return from drivers.</td>
</tr>
<tr>
<td>lpStat</td>
<td>Output</td>
<td>Empty. If users want to know specific information about error register, please call <a href="https://msdn.microsoft.com/en-us/library/windows/hardware/ff553614(v=vs.85).aspx">DeviceIoControl</a> to get status of the device.</td>
</tr>
</tbody>
</table>
Return Value
Successful: return non-zero values. Unsuccessful: return zero value. Please call GetLastError function.
Example
Wait event in synchronous mode.

```c
DWORD dwMask;
BOOL bSuccess = WaitCommEvent(hDevice, &dwMask, NULL);
if(bSuccess)
{
    if(dwMask & EV_ERR)
    {
        //to do
        DWORD dwError;
        bSuccess = ClearCommError(hDevice, &dwError, NULL);
        if(bSuccess)
        {
            //to do
            if(dwError& CE_FRAME || dwError& CE_BREAK)
            {
                CanStatusPar_t status;
                DWORD dwReturned;
                DeviceIoControl (hDevice,
                                CAN_IOCTL_STATUS,
                                NULL,
                                0,
                                &status,
                                sizeof(CanStatusPar_t),
                                &dwReturned,
                                NULL
                                );
            }
        }
    }
}
```
See Also
WaitCommEvent
DeviceIoControl
GetOverlappedResult

When user's operation cannot be finished immediately in asynchronous mode, this function should be called to wait operation to be completed.
BOOL GetOverlappedResult(
    HANDLE hDevice,
    LPOVERLAPPED lpOverlapped,
    LPDWORD lpNumberOfFramesTransferred,
    BOOL bWait
);
<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDevice</td>
<td>Input</td>
<td>Handle of the device which was opened.</td>
</tr>
<tr>
<td>lpOverlapped</td>
<td>Input</td>
<td>Events need to be waited are included. Refer to MSDN for more information.</td>
</tr>
<tr>
<td>lpNumberOfFramesTransferred</td>
<td>Output</td>
<td>Real numbers of data written and read; <em>WaitCommEvent</em> is not defined.</td>
</tr>
<tr>
<td>bWait</td>
<td>Input</td>
<td>TRUE, will not return until read/write operation is finished.    FALSE, return immediately no matter the operation is finished or not. Call GetLastErrorto return ERROR_IO_INCOMPLETE. Refer to MSDN for detailed information.</td>
</tr>
</tbody>
</table>
Return Value
Successful: return non-zero values. Unsuccessful: return zero value. Please call GetLastError function.
```c
#include <windows.h>
#include <stdio.h>

void main( )
{
    HANDLE hDevice;
    OVERLAPPED ov;
    BOOL bSuccess;
    DWORD dwEvtMask;
    DWORD dwLength;

    hDevice = CreateFile( "\\.\\\CAN1",
        GENERIC_READ | GENERIC_WRITE,
        0, // exclusive access
        NULL, // default security attributes
        OPEN_EXISTING,
        FILE_FLAG_OVERLAPPED,
        NULL
    );

    if (hDevice == INVALID_HANDLE_VALUE)
    {
        // Handle the error.
        printf("CreateFile failed with error %d.\n", GetLastError());
        return;
    }

    // Set the event mask.
    bSuccess = SetCommMask(hDevice, EV_ERR | EV_RXCHAR);
    if (!bSuccess)
    {
        // Handle the error.
        printf("SetCommMask failed with error %d.\n", GetLastError());
        return;
    }

    // Create an event object for use by WaitCommEvent.
    ov.hEvent = CreateEvent(
        NULL, // default security attributes
        FALSE, // auto reset event
        FALSE, // not signaled
        NULL // no name
    );
}
```
// Initialize the rest of the OVERLAPPED structure to zero.
ov.Internal = 0;
ov.InternalHigh = 0;
ov.Offset = 0;
ov.OffsetHigh = 0;

if (WaitCommEvent(hDevice, &dwEvtMask, &ov))
{
  if (dwEvtMask & EV_ERR)
  {
    // To do.
  }

  if (dwEvtMask & EV_RXCHAR)
  {
    // To do.
  }
}
else
{
  DWORD dwRet = GetLastError();
  if( ERROR_IO_PENDING == dwRet)
  {
    printf("I/O is pending...
");

    if(GetOverlappedResult(hDevice, &ov, &dwLength, TRUE))
    {
      //To do
    }
    else
    {
      //To do
    }
  }
  else
    printf("Wait failed with error %d.
", GetLastError());
}
See Also
ReadFile  WriteFile
WaitCommEvent
Guide for Visual C++ development

We will give an example by opening a CAN port and reading its current status so as to simply explain how to write base applications in Visual C++ environment.
Guide for Visual Basic development

We will give an example by opening a CAN port and reading its current status so as to simply explain how to write base applications in Visual Basic environment.
Guide for VC.NET development

We will give an example by opening a CAN port and reading its current status so as to simply explain how to write base applications in VC.NET environment.
Guide for VB.NET development

We will give an example by opening a CAN port and reading its current status so as to simply explain how to write base applications in VB.NET environment.
We will give an example by opening a CAN port and reading its current status so as to simply explain how to write base applications in C# environment.
Guide for LabVIEW development

We will introduce the LabVIEW VIs which use "Call Library Function Node" function to call Advantech CAN driver API and describe the format, purpose, and parameters for each VI. In addition, we also introduce how to create a new VI and use can_configure.vi as an example to describe how to design the actual program step by step.
## Data Type

Describe the data type of each object.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Boolean ( TRUE or FALSE).</td>
</tr>
<tr>
<td>String</td>
<td>String.</td>
</tr>
<tr>
<td>Cluster of mixed data type.</td>
<td>Cluster of mixed data type.</td>
</tr>
<tr>
<td>Cluster of numerics.</td>
<td>Cluster of numerics.</td>
</tr>
<tr>
<td>1-D array of cluster of mixed data type.</td>
<td>1-D array of cluster of mixed data type.</td>
</tr>
<tr>
<td>1-D array of unsigned byte.</td>
<td>1-D array of unsigned byte.</td>
</tr>
<tr>
<td>32-bit integer.</td>
<td>32-bit integer.</td>
</tr>
<tr>
<td>Unsigned byte [ 8-bit integer ].</td>
<td>Unsigned byte [ 8-bit integer ].</td>
</tr>
<tr>
<td>Unsigned word [ 16-bit integer ].</td>
<td>Unsigned word [ 16-bit integer ].</td>
</tr>
<tr>
<td>Unsigned long [ 32-bit integer ].</td>
<td>Unsigned long [ 32-bit integer ].</td>
</tr>
</tbody>
</table>
# Introduction to VI

All VIs used in current development are:

<table>
<thead>
<tr>
<th>VI Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acCanOpen.vi</code></td>
<td>Open CAN port by name.</td>
</tr>
<tr>
<td><code>acCanClose.vi</code></td>
<td>Close CAN port.</td>
</tr>
<tr>
<td><code>acEnterResetMode.vi</code></td>
<td>Enter reset mode.</td>
</tr>
<tr>
<td><code>acEnterWorkMode.vi</code></td>
<td>Enter work mode.</td>
</tr>
<tr>
<td><code>acClearRxFifo.vi</code></td>
<td>Clear CAN port receive buffer.</td>
</tr>
<tr>
<td><code>acSetBaud.vi</code></td>
<td>Set baud rate of the CAN Controller.</td>
</tr>
<tr>
<td><code>acSetBaudRegister.vi</code></td>
<td>Configure baud rate by custom mode.</td>
</tr>
<tr>
<td><code>acSetTimeOut.vi</code></td>
<td>Set timeout for read and write.</td>
</tr>
<tr>
<td><code>acSetSelfReception.vi</code></td>
<td>Set support for self reception.</td>
</tr>
<tr>
<td><code>acSetListenOnlyMode.vi</code></td>
<td>Set listen only mode of the CAN Controller.</td>
</tr>
<tr>
<td><code>acSetAcceptanceFilterMode.vi</code></td>
<td>Set acceptance filter mode of the CAN Controller.</td>
</tr>
<tr>
<td><code>acSetAcceptanceFilter.vi</code></td>
<td>Set acceptance filter code and mask of the CAN Controller.</td>
</tr>
<tr>
<td><code>acGetStatus.vi</code></td>
<td>Get the current status of the driver and the CAN Controller.</td>
</tr>
<tr>
<td><code>acCanWrite.vi</code></td>
<td>Write CAN message by unsigned char array.</td>
</tr>
<tr>
<td><code>acCanWriteCluster.vi</code></td>
<td>Write CAN message by structure array.</td>
</tr>
<tr>
<td><code>acCanRead.vi</code></td>
<td>Read CAN message.</td>
</tr>
<tr>
<td><code>acSetCommMask.vi</code></td>
<td>Execute SetCommMask of the CAN Controller.</td>
</tr>
<tr>
<td><code>acGetCommMask.vi</code></td>
<td>Execute GetCommMask of the CAN Controller.</td>
</tr>
</tbody>
</table>
We list the Advantech CAN driver VIs and describes the format, purpose and parameters for each VI.

All VIs use "Call Library Function Node" to call Advantech CAN driver API at AdvCanVI.dll. The default path is "C:\WINDOWS\system32"; user also can find it at "C:\Program Files\Advantech\AdvCAN\LabVIEW\Bin".
acCanOpen.vi

Purpose
Open CAN port by name.

Format

Input
CANPort
CANPort is the name of the CAN Port to open.
This name uses the syntax "CANx", where x is a decimal number.

Synchronization
The function will not return until the I/O function call is completed in
synchronous mode, while an asynchronous file handle makes it
possible for the function to return immediately, no matter the I/O
operation is completed or not. TRUE, synchronous ; FALSE, asynchronous. Refer to CreateFile.

Error in
Error in is a cluster of 3 elements that describes the error status
before this VI executes. If error in indicates that an error occurred
before this VI was called, this VI will not execute its function, and
pass the error through to its error out cluster.
1. status - Status is true when error occurs; If status is true,
this VI will not do any thing.
2. code - When error occurs, the value specifies error code
returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.

Output
Handle
An open handle to the specified CANport for all subsequent Advantech
CAN VIs for this object, including the final call to acCanClose.vi.

Overlapped
0 Asynchronous
Overlapped will be clear and use the CreateEvent API to create a
Handle to hEvent of Overlapped.
1 Synchronous (default)
Error out

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
acCanClose.vi

Purpose
Close CAN port by handle.

Format

Input

**Handle in**
Handle in is the name of the CAN port to close. Refer to [CloseHandle](#).

**Overlapped**
Close the hEvent handle of Overlapped.

**Error in**
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. **status** - Status is true when error occurs. If status is true, this VI will not do anything.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.

Output

**Error out**
Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicated an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
acEnterResetMode.vi

Purpose
Stop chip and enter Reset Mode. When users want to configure the SJA1000 chip, use this VI first.

Format

Input

- **Handle in**
  Handle in is the handle which originates from acCanOpen.vi.

- **Error in**
  Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.
  1. **status** - Status is true when error occurs. If status is true, this VI will not do anything.
  2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
  3. **source** - This string describes the origin of the error.

Output

- **Handle out**
  Handle out is the handle for the next Advantech CAN driver VI.

- **Error out**
  Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicated an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.
  1. **status** - Status is true when error occurs.
  2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
  3. **source** - This string describes the origin of the error.
acEnterWorkMode.vi

Purpose

Start chip and enter Work Mode. Use this VI after configuring the SJA1000 chip.

Format

Input

Handle in

Handle in is the handle which originates from acCanOpen.vi.

Error in

Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI will not to execute its function, and pass the error through to its error out cluster.

1. status - Status is true when error occurs. If status is true, this VI will not do anything.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.

Output

Handle out

Handle out is the handle for the next Advantech CAN driver VI.

Error out

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicated an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. status - Status is true when error occurs.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.
acClearRxFifo.vi

Purpose

Clear CAN port receive buffer by handle in.

Format

```
   Handle in     Handle out
  Error in      Error out
```

Input

**Handle in**

Handle in is the handle which originates from acCanOpen.vi.

**Error in**

Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. **status** - Status is true when error occurs; If status is true, this VI will not do anything.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.

Output

**Handle out**

Handle out is the handle for the next Advantech CAN driver VI.

**Error out**

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
**acSetBaud.vi**

**Purpose**
Set the baud rate for CAN bus communication.

**Format**

```plaintext
Handle in  Baudrate  Error in
Input
Handle in
Handle in is the handle which originates from acCanOpen.vi.

**Baudrate (K)**
This is the standard mode to set baud rate, and the value is in units of K. Refer to Command_par.

**Error in**
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. **status** - Status is true when error occurs; If status is true, this VI will not do anything.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.

**Output**

**Handle out**
Handle out is the handle for the next Advantech CAN driver VI.

**Error out**
Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
acSetBaudRegister.vi

Purpose

This is the custom mode to set the baud rate register for CAN bus communication.

Format

Input

Handle in

Handle in is the handle which originates from acCanOpen.vi.

BTR 0

BTR 0 Timing Registers are used to define the bit-rate on CAN bus. Refer to Command_par.

BTR 1

BTR 1 Timing Registers are used to define the bit-rate on CAN bus. Refer to Command_par.

Error in

Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. status - Status is true when error occurs; If status is true, this VI will not do anything.

2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.

3. source - This string describes the origin of the error.

Output

Handle out

Handle out is the handle for the next Advantech CAN driver VI.

Error out

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.
1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
acSetTimeOut.vi

Purpose
Configure the maximum waiting time in milliseconds between two messages on the bus.

Format

Input

Handle in
Handle in is the handle which originates from acCanOpen.vi.

ReadTimeOut(ms)
ReadTimeOut is the maximum number of milliseconds to wait to receive messages.

WriteTimeOut(ms)
WriteTimeOut is the maximum number of milliseconds to wait to transmit messages.

Error in
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. status - Status is true when error occurs; If status is true, this VI will not do anything.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.

Output

Handle out
Handle out is the handle for the next Advantech CAN driver VI.

Error out
Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.
1. **status** - Status is true when error occurs.

2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.

3. **source** - This string describes the origin of the error.
acSetSelfReception.vi

Purpose
Configure the CAN bus into Self Reception Mode.
Note, before entering the Self Reception Mode the Reset Mode has to be entered.

Format

Input
- **Handle in**
  Handle in is the handle which originates from acCanOpen.vi.

- **SelfReception**
  0  Off (default)
    Self Reception Mode is disabled. Transmitted frames do not appear in receive buffer.
  1  On
    Self Reception Mode is enabled. When the message transmitted on the bus, it also entered in the receive buffer.

- **Error in**
  Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.
  1. **status** - Status is true when error occurs; If status is true, this VI will not do anything.
  2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
  3. **source** - This string describes the origin of the error.

Output
- **Handle out**
  Handle out is the handle for the next Advantech CAN driver VI.

- **Error out**
  Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicated an error, this VI will not do anything, and
the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.

2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.

3. **source** - This string describes the origin of the error.
acSetListenOnlyMode.vi

Purpose
Configure the CAN bus into Listen Only Mode.
Note: Before entering the Listen Only Mode, you have to enter the Reset Mode first.

Format

Input
- **Handle in**
  Handle in is the handle which originates from acCanOpen.vi.
- **ListenOnly**
  0    Off (default)
  Listen Only Mode is disabled. CAN port can transmit message or receive message.
  1    On
  Listen Only Mode is enabled. CAN port can only receive frames.
- **Error in**
  Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.
  1. **status** - Status is true when error occurs; If status is true, this VI will not do anything.
  2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
  3. **source** - This string describes the origin of the error.

Output
- **Handle out**
  Handle out is the handle for the next Advantech CAN driver VI.
- **Error out**
  Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.
1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
acSetAcceptanceFilterMode.vi

Purpose
Configure the Acceptance Filter Mode of CAN port.

Format

```
Handle in — Mode — Handle out
FilterMode — Error in — Error out
```

Input

**Handle in**
Handle in is the handle which originates from acCanOpen.vi.

**FilterMode**
0   Dual
    Refer to Acceptance filtering.
1   Single (default)
    Refer to Acceptance filtering.

**Error in**
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. **status** - Status is true when error occurs; If status is true, this VI will not do anything.
2. **code** - When error occurred, the value specifies error code that returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.

Output

**Handle out**
Handle out is the handle for the next Advantech CAN driver VI.

**Error out**
Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurred, the value specifies error code that returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
buacSetAcceptanceFilter.vi

Purpose

Set the Acceptance code and Acceptance Mask.

Format

Input

Handle in

Handle in is the handle which originates from acCanOpen.vi.

FilterCode

Refer to Acceptance filtering.

FilterMask

Refer to Acceptance filtering.

Error in

Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. status - Status is true when error occurs; If status is true, this VI will not do anything.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.

Output

Handle out

Handle out is the handle for the next Advantech CAN driver VI.

Error out

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. status - Status is true when error occurred.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
Purpose

Get the CAN port current status.

Format

Input

Handle in
Handle in is the handle which originates from acCanOpen.vi.

Error in
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. status - Status is true when error occurs; If status is true, this VI will not do anything.

2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.

3. source - This string describes the origin of the error.

Output

Handle out
Handle out is the handle for the next Advantech CAN driver VI.

Status
Return value is structure of CanStatusPar_t. Refer to CanStatusPar_t.

Error out
Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. status - Status is true when error occurred.

2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.

3. source - This string describes the origin of the error.
acCanWrite.vi

Purpose
Write multiple frames to a CAN bus.

Format

Input

Handle in
Handle in is the handle which originates from acCanOpen.vi.

MSG
MSG is an array of frames. Each frame uses the typedef canmsg_t. 4 bytes for flags, 4 bytes for cob (In MSDN, this is reserved to set to zero), 4 bytes for id, 2 bytes for length, 8 bytes for data. And this VI divide MSG array by the size of canmsg_t to calculate how many number of frames to write.

Overlapped
If user uses the synchronous mode at acCanOpen.vi, this parameter can be ignored. In the other mode, hEvent of Overlapped is the handle from acCanOpen.vi.

Error in
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.
1. status - Status is true when error occurs. If status is true, this VI will not do anything.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.

Output

Handle out
Handle out is the handle for the next Advantech CAN driver VI.

NumberOfWritten
NumberOfWritten describes how many frames have send.

**Error out**

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicated an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
**acCanWriteCluster.vi**

**Purpose**
Write multiple frames to CAN bus.

**Format**

![Diagram](image)

**Input**

- **Handle in**
  Handle in is the handle which originates from **acCanOpen.vi**.

- **MSG**
  MSG is an array of clusters. Each cluster represents a CAN frame which uses the typedef `canmsg_t`.
  The dimsize of MSG array represents number of frames to write at one time.

- **Overlapped**
  If user use the synchronous mode at **acCanOpen.vi**, this parameter can be ignored.
  In the other mode, hEvent of Overlapped is the handle from **acCanOpen.vi**.

- **Error in**
  Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI will not to execute its function, and pass the error through to its error out cluster.
  1. **status** - Status is true when error occurs. If status is true, this VI will not do anything.
  2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
  3. **source** - This string describes the origin of the error.

**Output**

- **Handle out**
  Handle out is the handle for the next Advantech CAN driver VI.

- **NumberOfWritten**
  NumberOfWritten described how many frames have send.
Error out

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicated an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. status - Status is true when error occurs.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.
acCanRead.vi

Purpose
Read multiple frames from CAN bus.

Format

Input

**Handle in**
Handle in is the handle which originates from acCanOpen.vi.

**Overlapped**
If user uses the synchronous mode at acCanOpen.vi, this parameter can be ignored.
In the other mode, hEvent of Overlapped is the handle from acCanOpen.vi.

**ReadCount**
Indicates the number of frames to read at one time from the CAN bus.

**Error in**
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI will not execute its function, and pass the error through to its error out cluster.

1. **status** - Status is true when error occurs. If status is true, this VI will not do anything.

2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.

3. **source** - This string describes the origin of the error.

Output

**Handle out**
Handle out is the handle for the next Advantech CAN driver VI.

**MSG**
MSG returns an array of frames. Each frame in the array uses the typedef `canmsg_t`.

**NumberOfRead**
Specifies the number of frames returned in MSG.

**Error out**

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicated an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
acSetCommMask.vi

Purpose
Set the Event flags for CAN port.

Format

![Diagram of the acSetCommMask.vi format with inputs and outputs]

Input

**Handle in**
Handle in is the handle which originates from acCanOpen.vi.

**EventMask**
This parameter fixes the value of "EV_ERR | EV_RXCHAR". Refer to SetCommMask.

**Error in**
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.

1. **status** - Status is true when error occurs; If status is true, this VI will not do anything.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.

Output

**Handle out**
Handle out is the handle for the next Advantech CAN driver VI.

**Error out**
Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
acGetCommMask.vi

Purpose
Get event type set in acSetCommMask.vi.

Format

```
   Input

   Handle in
   Handle in is the handle which originates from acCanOpen.vi.

   Error in
   Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.
   1. status - Status is true when error occurs; If status is true, this VI will not do anything.
   2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
   3. source - This string describes the origin of the error.

   Output

   Handle out
   Handle out is the handle for the next Advantech CAN driver VI.

   GetEventMask
   Return the Event type. Refer to GetCommMask.

   Error out
   Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.
   1. status - Status is true when error occurred.
   2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
   3. source - This string describes the origin of the error.
```
acWaitEvent.vi

Purpose

Wait for one state to occur in this VI.
If state is EX_CHAR, it will read one or more frames from CAN bus. Or if status is EX_ERR, it will return error code. Refer to acSetCommMask.vi.

Format

Input

Handle in
Handle in is the handle originates from acCanOpen.vi.

Overlapped
If user use the synchronous mode at acCanOpen.vi, this parameter can be ignored.
In the other mode, hEvent of Overlapped is the handle from acCanOpen.vi.

ReadCount
Indicates the number of frames to read at one time from the CAN bus.

Error in
Error in is a cluster of 3 elements that describes the error status before this VI executes. If error in indicates that an error occurs before this VI is called, this VI will not execute its function, and pass the error through to its error out cluster.
1. status - Status is true when error occurs; If status is true, this VI will not do anything.
2. code - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. source - This string describes the origin of the error.

Output

Handle out
Handle out is the handle for the next Advantech CAN driver VI.

MSG
MSG returns an array of frames. Each frame in the array uses the
typedef `canmsg_t`.

**NumberOfRead**

Specifies the number of frames returned in MSG.

**ErrorCode**

Return error code when the CAN Controller has error.

**Error out**

Error out is a cluster of 3 elements that describes error conditions. If the Error in cluster indicates an error, this VI will not do anything, and the Error out cluster contains the same error information of error in. Otherwise, Error out describes the error status of this VI.

1. **status** - Status is true when error occurs.
2. **code** - When error occurs, the value specifies error code returned by the GetLastError function of Windows API.
3. **source** - This string describes the origin of the error.
Guide for LabVIEW development

We will give an example by opening a CAN port and reading its current status so as to simply explain how to write base applications in LabVIEW environment.
Add the `can_configure.vi` each `vi` will be used.

We use example "`can_configure.vi`" to describe how to design the function on block Diagram. Please refer to `ConfigureFlowChart` for the design flow chart.

1. Right click on block diagram and select **Select a VI....**

2. Use the advcanvi.llb or common.llb to open VI, and then select a VI added to the block panel. The default path of *.llb file is "C:\WINDOWS\system32"; user can also find it at "C:\Program Files\Advantech\AdvCAN\LabVIEW".
3. Repeat steps one and two, then your block diagram should look like the diagram below.
Run the VI at front panel, and the following result will be displayed.
Click the toolbar or the menu File -> New, the "New Connecting..." dialog box for opening device pops up:

Users should select CAN device first, and then configure Baudrate, Acceptance Code, Acceptance Mask. Users should set Max Rows In View, which decides the number of CAN messages that should be saved (Min: 1000; Max: 1000000).
Close Device

Click the toolbar or the menu File -> Close to close the device and abandon all the data received.
Load Data from File

Click the toolbar or the menu File -> Load, and then CANMonitor will load message from the file that saves data.
Save Data to File

Click the toolbar or the menu File -> Save, and then CANMonitor will save the messages received to the file with the suffix of acb.
Start Communication

Click the toolbar or the menu File -> Start, and CANMonitor will use the configuration before stopping the current communication to restart communication.
Stop Communication

Click the toolbar or the menu File -> Stop, and CANMonitor will stop the current communication.
Show the Message and Hide the Message

If the communication is started, click the toolbar or the menu File -> Show. Then the user can see the messages being received and sent.

Click the toolbar or the menu File -> Hide, and CANMonitor will hide the messages being sent and received.
Clear Data

Click the toolbar or the menu File -> Clear to delete all the data currently stored in CANMonitor.
If the toolbar or the menu File -> Auto Scroll is selected and the new data is being sent or received, CANMonitor will automatically show the data in the last line.
Click the toolbar or the menu File -> Goto, users will see the below interface:

After entering the line number in Text Box, click Goto button will go to the specified line.
Listen Only Mode

Under Listen only mode, the system will not send answer message after receiving CAN message. This can ensure that CANMonitor will not be a burden to the network, with an exception to that the user of CANMonitor needs to send message.
CAN Message Filter

CAN Message Filter has four modes. The selection will only affect CAN display.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFF</td>
<td>Whether to show the Standard frame. After this mode is selected, filter will start from the setting if it is receiving data. The Standard frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The Standard frame can be showed.</td>
</tr>
<tr>
<td>EFF</td>
<td>Whether to show the Extended frame. After this mode is selected, filter will start from the setting if it is receiving data. The Extended frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The Extended frame can be showed.</td>
</tr>
<tr>
<td>RTR</td>
<td>Whether to show the Remote frame. After this mode is selected, filter will start from the setting if it is receiving data. The Remote frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The Remote frame can be showed.</td>
</tr>
<tr>
<td>Self Reception</td>
<td>Whether to show the self reception frame. After this mode is selected, filter will start from the setting if it is receiving data. The self reception frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The self reception frame can be showed.</td>
</tr>
</tbody>
</table>
CAN Message View

CAN Message View is shown as below:

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Tx/Rx</th>
<th>ID (HEX)</th>
<th>Flag</th>
<th>Length</th>
<th>Data (HEX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.087479</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 20 12 00 00 00 00 00</td>
</tr>
<tr>
<td>2</td>
<td>0.087846</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 20 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>3</td>
<td>0.196849</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 21 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>4</td>
<td>0.197299</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 21 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>5</td>
<td>0.306206</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 22 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>6</td>
<td>0.306700</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 22 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>7</td>
<td>0.416537</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 23 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>8</td>
<td>0.416996</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 23 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>9</td>
<td>0.525910</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 24 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>10</td>
<td>0.526350</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 24 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>11</td>
<td>0.536486</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 25 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>12</td>
<td>0.536961</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 25 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>13</td>
<td>0.636889</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 26 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>14</td>
<td>0.636943</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 26 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>15</td>
<td>0.805217</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 27 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>16</td>
<td>0.805699</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 27 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>17</td>
<td>0.916565</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 28 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>18</td>
<td>0.917089</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 28 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>19</td>
<td>1.026800</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 29 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>20</td>
<td>1.026847</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 29 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>21</td>
<td>1.137267</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 30 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>22</td>
<td>1.137738</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 30 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>23</td>
<td>1.195083</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 32 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>24</td>
<td>1.196294</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 32 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>25</td>
<td>1.307172</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 34 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>26</td>
<td>1.307668</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 34 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>27</td>
<td>1.417528</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 36 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>28</td>
<td>1.417960</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 36 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>29</td>
<td>1.526004</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 38 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>30</td>
<td>1.526370</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 38 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>31</td>
<td>1.636246</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 40 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>32</td>
<td>1.636714</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 40 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>33</td>
<td>1.696781</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 42 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>34</td>
<td>1.697240</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 42 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>35</td>
<td>1.806060</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 44 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>36</td>
<td>1.806070</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 44 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>37</td>
<td>1.916509</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 46 12 00 00 00 00 00 00</td>
</tr>
<tr>
<td>38</td>
<td>1.916962</td>
<td>Rx</td>
<td>582</td>
<td>SFF</td>
<td>8</td>
<td>60 46 12 00 00 00 00 02 06</td>
</tr>
<tr>
<td>39</td>
<td>2.025866</td>
<td>Rx</td>
<td>602</td>
<td>SFF</td>
<td>8</td>
<td>40 48 12 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

**Column**

- No:
  - The number of the message.
- Time:
  - The time to open device; the smallest unit is ms.
- Tx/Rx:
  - Sends or receives the message.
- ID or ID (HEX) or ID (ASCII):
  - The ID of the message. Click the column name, and then the data in the column will switch between decimal system, hexadecimal system and ASCII system.
- Flag:
  - Shows the message type: SFF, EFF, RTR, Self Reception.
- Length:
  - The data length of the message.
- Data or Data (HEX) or Data (ASCII):
  - The data of the message. Click the column name, and then the data in the column will switch between decimal system, hexadecimal system and ASCII system.
List Node

If the system is not at the Listen Only mode, click the Function/CANopen/List Node. Then the List Node dialog box pops up. Here users can see all CANopen nodes and their states detected.

![List Node dialog box](image)
Send CAN Message

If the system is not at the Listen Only mode, please click Function/CANopen/Change Node State. Then the Change Node State dialog box pops up:

Users can change the CANopen node state detected.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node ID</td>
<td>Selects node.</td>
</tr>
<tr>
<td>Current State</td>
<td>The current state of the node.</td>
</tr>
<tr>
<td>New State</td>
<td>The state of the node changed.</td>
</tr>
</tbody>
</table>
Read Data from Node

If the system is not at the Listen Only mode, click Function/CANopen/Read object. The Read Object dialog box pops up:

![Read Object dialog box](image)

Users can read data of a certain node that has been detected.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node ID</td>
<td>Selects node.</td>
</tr>
<tr>
<td>Index</td>
<td>Index.</td>
</tr>
<tr>
<td>Sub Index</td>
<td>Sub index.</td>
</tr>
<tr>
<td>Read Length</td>
<td>Data length returned.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Return value. If the operation is failed, error message will be returned.</td>
</tr>
</tbody>
</table>
Write Data to Node

If the system is not at the Listen Only mode, click Function/CANopen/Write object. The Write Object dialog box pops up:

Users can change data of a certain node that has been detected.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node ID</td>
<td>Selects node.</td>
</tr>
<tr>
<td>Index</td>
<td>Index.</td>
</tr>
<tr>
<td>Sub Index</td>
<td>Sub index.</td>
</tr>
<tr>
<td>Write Length</td>
<td>Data length to be sent.</td>
</tr>
<tr>
<td>Write Value</td>
<td>Data to be sent.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Correct: Ok; Error: Error code; Timeout: Timeout.</td>
</tr>
</tbody>
</table>
Detect All Nodes in CANopen

If the system is not at the Listen Only mode, click Function/CANopen/Node Guard All. Then CANMonitor will detect all the nodes in CANopen.
**CANopen Message Filter**

CANopen Message Filter is divided into two parts:

- The analytic method of COB-ID

Users can select to analyze COB-ID according to pre-defined connection of CANopen or according to EDS file. The default selection is the pre-defined connection.

- Show the messages

The selection will only affect CANopen interface display.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO</td>
<td>Whether to show SDO (Service Data Object) frame. After this mode is selected, filter will start from the setting if it is receiving data. The SDO frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The SDO frame can be showed.</td>
</tr>
<tr>
<td>PDO</td>
<td>Whether to show PDO (Process Data Object) frame. After this mode is selected, filter will start from the setting if it is receiving data. The PDO frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The PDO frame can be showed.</td>
</tr>
<tr>
<td>NMT</td>
<td>Whether to show NMT (Network Management) frame. After this mode is selected, filter will start from the setting if it is receiving data. The NMT frames received later can be showed. If it is not receiving data, i.e., file data is being</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NMT Error Control</td>
<td>Whether to show NMT Error Control (Error Control services) frame. After this mode is selected, filter will start from the setting if it is receiving data. The NMT Error Control frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The NMT Error Control frame can be showed.</td>
</tr>
<tr>
<td>EMCY</td>
<td>Whether to show EMCY (Emergency Object) frame. After this mode is selected, filter will start from the setting if it is receiving data. The EMCY frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The EMCY frame can be showed.</td>
</tr>
<tr>
<td>TIME-STAMP</td>
<td>Whether to show TIME-STAMP (Time Stamp Object) frame. After this mode is selected, filter will start from the setting if it is receiving data. The TIME-STAMP frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The TIME-STAMP frame can be showed.</td>
</tr>
<tr>
<td>SYNC</td>
<td>Whether to show SYNC (Synchronisation Object) frame. After this mode is selected, filter will start from the setting if it is receiving data. The SYNC frames received later can be showed. If it is not receiving data, i.e., file data is being viewed or the communication is stopped, filter will start form the first frame of the message. The SYNC frame can be showed.</td>
</tr>
</tbody>
</table>
CANopen Message View

CANopen Message View is shown as below:

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Protocol</th>
<th>ID or ID (HEX)</th>
<th>Data or Data (HEX)</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>390.989924</td>
<td>MNT Error Control</td>
<td>702 00</td>
<td>40 00 10 00 00 00 00 00</td>
<td>Current state of the node[2] is Initialization</td>
</tr>
<tr>
<td>2</td>
<td>400.361489</td>
<td>SD0</td>
<td>602</td>
<td>40 00 10 00 00 00 00 00</td>
<td>The client request to read byte from [100c:0] of node[2]</td>
</tr>
<tr>
<td>3</td>
<td>400.306069</td>
<td>SD0</td>
<td>582</td>
<td>40 00 10 00 01 00 00 00</td>
<td>The server node[2] responds ok</td>
</tr>
<tr>
<td>4</td>
<td>488.188142</td>
<td>SD0</td>
<td>602</td>
<td>23 00 10 00 00 00 00 00</td>
<td>The client request to write 4 bytes to [100c:0] of node[2]</td>
</tr>
<tr>
<td>5</td>
<td>488.161774</td>
<td>SD0</td>
<td>582</td>
<td>80 00 10 00 10 00 07 06</td>
<td>The server node[2] returns error: 6 7 0 1 0</td>
</tr>
</tbody>
</table>

**Column**
- No: The number of the message.
- Time: The time to open device; the smallest unit is ms.
- Protocol: The protocol type of CANopen message.
- ID or ID (HEX) or ID (ASCII): The ID of the message. Click the column name, and then the data in the column will switch between decimal system, hexadecimal system and ASCII system.
- Data or Data (HEX) or Data (ASCII): The data of the message. Click the column name, and then the data in the column will switch between decimal system, hexadecimal system and ASCII system.
- Info: The explanation of the message.
Send CAN Message

If the user is not at Listen Only mode, please click Function/CAN/Send CAN Message. Then Send CAN Message dialog box pops up:

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>CAN message ID.</td>
</tr>
<tr>
<td>Flag</td>
<td>SFF: Standard Frame; EFF: Extended Frame.</td>
</tr>
<tr>
<td>RTR</td>
<td>Whether this message is a Remote frame.</td>
</tr>
<tr>
<td>Self Reception</td>
<td>Whether to enable self reception. If it is enabled, the user can see the message sent by himself.</td>
</tr>
<tr>
<td>Cycle</td>
<td>Whether to send repeatedly.</td>
</tr>
<tr>
<td>ms</td>
<td>The interval period of the cycle.</td>
</tr>
<tr>
<td>Auto Increase</td>
<td>The data sent will increase automatically. If check it, user do not to input length and data.</td>
</tr>
<tr>
<td>Length</td>
<td>The length of the message, from 1 to 8.</td>
</tr>
<tr>
<td>Data</td>
<td>The data sent, up to 8 byte.</td>
</tr>
<tr>
<td>Send Time Out</td>
<td>Time out of sending the data.</td>
</tr>
</tbody>
</table>