



***AXIOMTEK***

**AX92903**

**Full-size CAN Bus Mini PCI-Express  
Module**

**User's Manual**



## **Disclaimers**

This manual has been carefully checked and believed to contain accurate information. Axiomtek Co., Ltd. assumes no responsibility for any infringements of patents or any third party's rights, and any liability arising from such use.

Axiomtek does not warrant or assume any legal liability or responsibility for the accuracy, completeness or usefulness of any information in this document. Axiomtek does not make any commitment to update the information in this manual.

Axiomtek reserves the right to change or revise this document and/or product at any time without notice.

No part of this document may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Axiomtek Co., Ltd.

**©Copyright 2015 Axiomtek Co., Ltd.**

**All Rights Reserved**

**March 2015, Version A1**

**Printed in Taiwan**

## ESD Precautions

The boards have integrated circuits sensitive to static electricity. To prevent chipsets from electrostatic discharge damage, please take care of the following jobs with precautions:

- Do not remove boards or integrated circuits from their anti-static packaging until you are ready to install them.
- Before holding the board or integrated circuit, touch an unpainted portion of the system unit chassis for a few seconds. It discharges static electricity from your body.
- Wear a wrist-grounding strap, available from most electronic component stores, when handling boards and components.

## Trademarks Acknowledgments

Axiomtek is a trademark of Axiomtek Co., Ltd.

Windows<sup>®</sup> is a trademark of Microsoft Corporation.

IBM, PC/AT, PS/2, VGA are trademarks of International Business Machines Corporation.

Intel<sup>®</sup> is a trademark of Intel Corporation.

Other brand names and trademarks are the properties and registered brands of their respective owners.

# Table of Contents

---

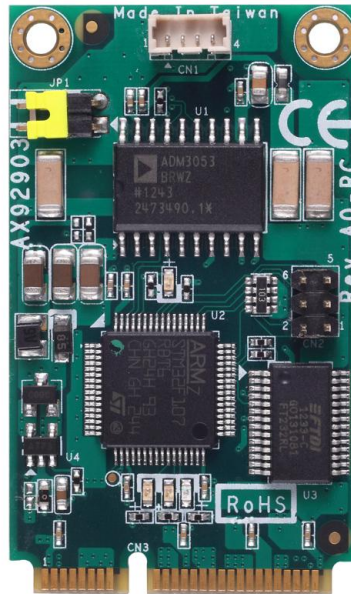
Disclaimers.....	ii
ESD Precautions .....	iii
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Features .....	1
1.2 Specifications.....	2
<b>Chapter 2 Module and Pin Assignments.....</b>	<b>3</b>
2.1 Dimensions and Fixing Holes .....	3
2.2 Module Layout.....	4
2.3 Connectors .....	5
2.3.1 CAN Bus Connector (CN1) .....	5
2.4 CAN Cable.....	6
<b>Chapter 3 Software Installation .....</b>	<b>7</b>
3.1 Driver Installation .....	7
3.2 Utility Installation .....	14
<b>Chapter 4 Using Software Utility.....</b>	<b>19</b>
4.1 Overview .....	19
4.2 Connect Setting Window .....	20
4.3 Transmit Setting Window .....	22
4.4 Receive-Formatted Setting Window .....	26
4.5 Receive-Raw Setting Window .....	30
4.6 Config Setting Window .....	32
4.7 Others.....	33
<b>Chapter 5 Commands Available .....</b>	<b>37</b>
5.1 Overview .....	37
5.2 Summary Table of All Commands.....	38
5.3 Identifier Filtering.....	52
5.4 Data Format .....	52
5.5 Error Code .....	53
5.6 Some Examples.....	53

<b>Chapter 6</b>	<b>Programming Guide .....</b>	<b>55</b>
6.1	COM Port Device Functions .....	55
6.2	CAN Device Functions .....	59
6.3	NVRAM Functions .....	79
6.4	Type Definitions .....	82

**This page is intentionally left blank.**

# Chapter 1

## Introduction



The CAN (Controller Area Network) is a serial bus system and is usually used in structuring intelligent industrial device networks and building smart automatic control systems. With the cost-effective AX92903, a full-size mini PCI-Express module with one CAN bus interface, user is easily capable of integrating computer with PCI-Express Mini Card socket to CAN network.

Besides, it comes with CAN monitor tool that displays the CAN received message in decimal or hexadecimal mode and also shows the time stamp of each received message. Both of the GUI management program and programming guide are ready for use. The CAN host, monitor or HMI, can access and control CAN devices through AX92903.

### 1.1 Features

- One CAN bus interface
- Supports both CAN 2.0A and CAN 2.0B
- Up to 1Mbps CAN transmission speed
- 2KV isolation
- Drivers support Windows® 7/ 7 x64/ 8/ 8 x64/ 8.1/ 8.1 x64

## 1.2 Specifications

- **Controller**
  - STM32107.
- **CAN Bus Interface**
  - One channel.
- **Transceiver**
  - ADM3053.
- **Operating Temperature**
  - -20°C~70°C.
- **Isolation**
  - 2KV.
- **CAN Bus Connection**
  - 4-pin connector.
- **Terminator Resistor**
  - Onboard 120Ω.
- **Support USB 2.0 to CAN 2.0A and CAN 2.0B**
  - USB 2.0 full speed.
- **LED**
  - Power, CAN TX/RX, CAN error, firmware update.
- **Form Factor**
  - Mini PCI-Express.
- **Software**
  - Protocol: CAN 2.0A/2.0B.
  - Transmission Speed: Up to 1Mbps.
  - Receive Buffer: 1000 data frames.
  - NVRAM: 64Kbyte (for user-defined).
  - Filter ID supported.
  - Time Stamp: Up to 1us.
  - RTR supported.
  - OS Driver Support: Windows® 7/ 7 x64/ 8/ 8 x64/ 8.1/ 8.1 x64.



**Note**

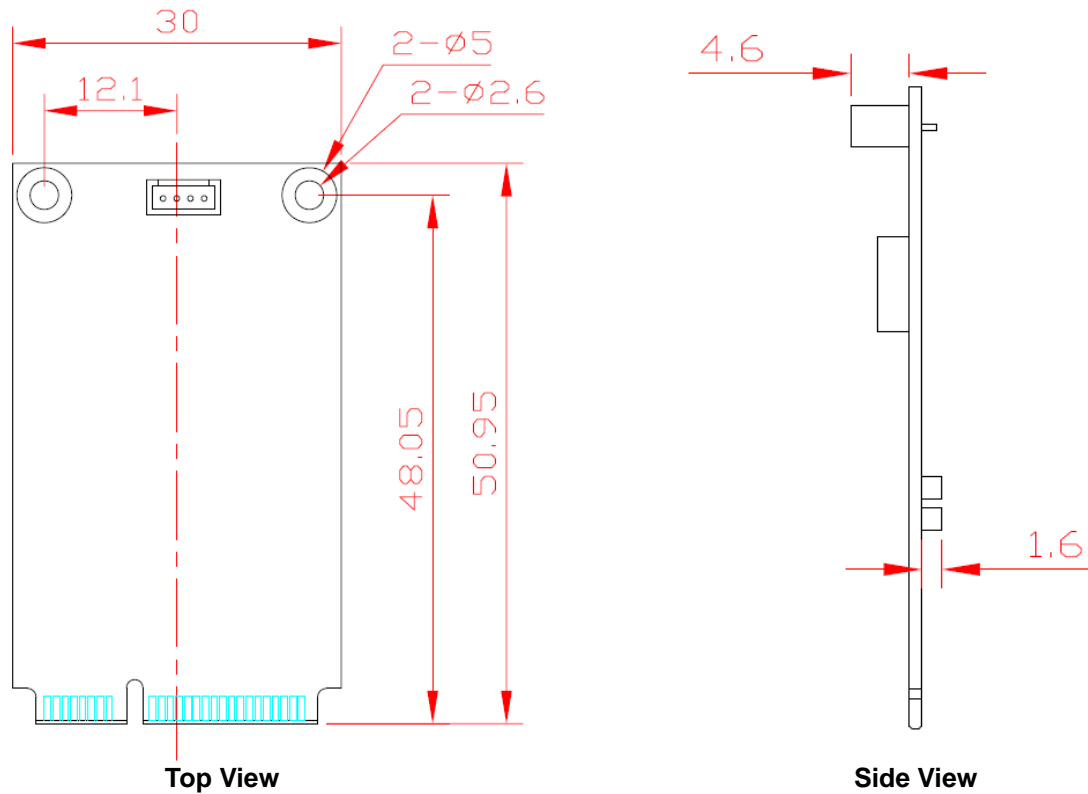
*All specifications and images are subject to change without notice.*



# Chapter 2

## Module and Pin Assignments

### 2.1 Dimensions and Fixing Holes



## 2.2 Module Layout



Top View

## 2.3 Connectors

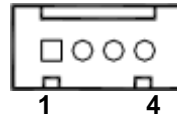
Signals go to the other parts of the hardware through connectors. Loose or improper connection might cause problems, please make sure all connectors are properly and firmly connected. Here is a summary table which shows connectors on the hardware.

Connector	Description
CN1	CAN Bus Connector

### 2.3.1 CAN Bus Connector (CN1)

This is a MOLEX 53047-0410 4-pin connector for CAN bus interface.

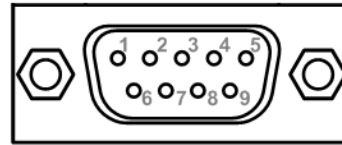
Pin	Signal
1	GND
2	CAN_H
3	CAN_L
4	GND



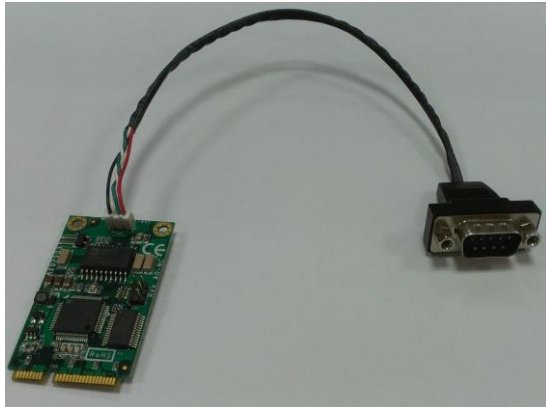
## 2.4 CAN Cable

The AX92903 comes with a cable where one of its ends is a 9-pin D-Sub connector. The pin assignment corresponds to CiA® 102 specification.

Pin	Signal
1	N.C.
2	CAN_L
3	GND
4	N.C.
5	N.C.
6	GND
7	CAN_H
8	N.C.
9	N.C.



Attach this cable to CN1, see figure below.



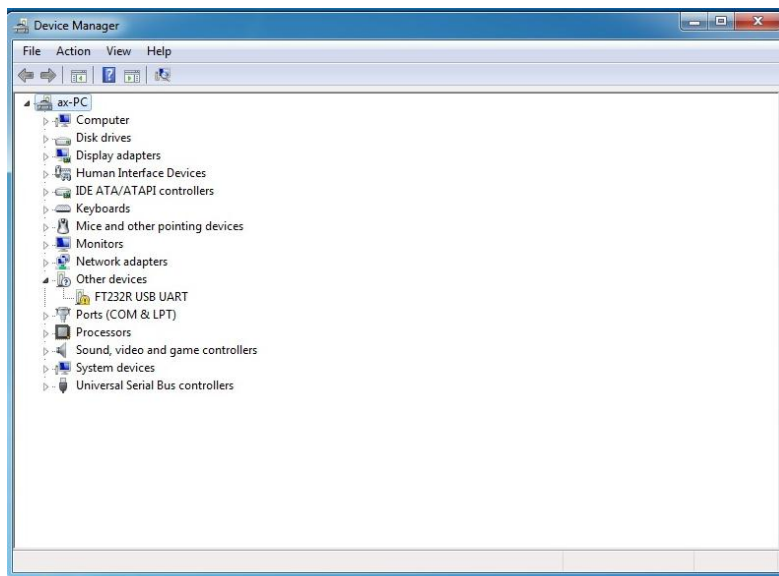
# Chapter 3

## Software Installation

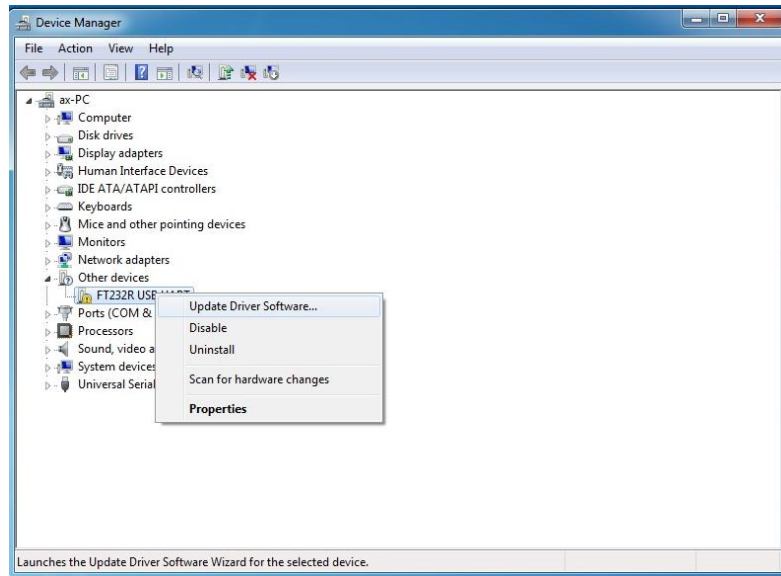
### 3.1 Driver Installation

It is necessary to install driver before you can use AX92903. This section provides brief guide of how to install the device driver under Windows® 7. Before installing the driver software, make sure that you have installed the AX92903 into host computer's PCI-Express Mini Card socket. You can follow the onscreen instructions to install the driver.

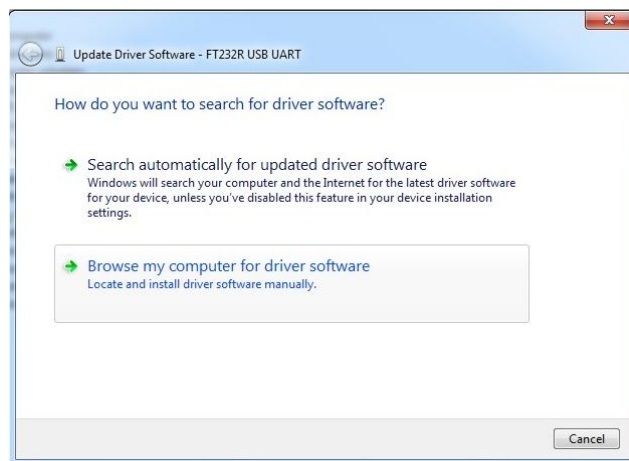
1. Before you start installing driver, power on your computer and we recommend copying driver files to your desktop under directory named "CDM20808". Open Device Manager from the Control Panel in Windows® 7 and you can see the following screen.



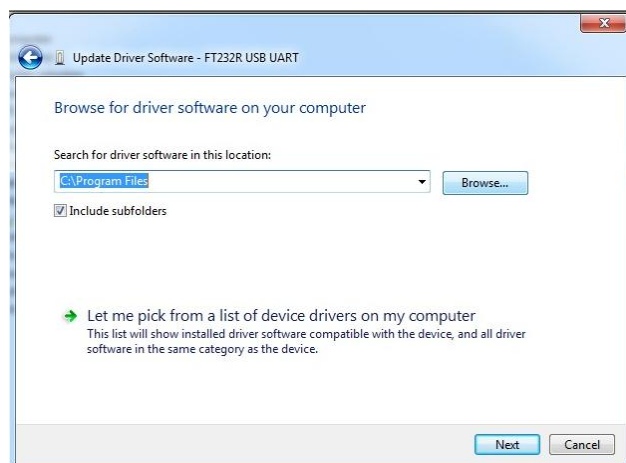
2. Right click on *Other devices* > *FT232R USB UART* and launch Update Driver Software for the selected device.



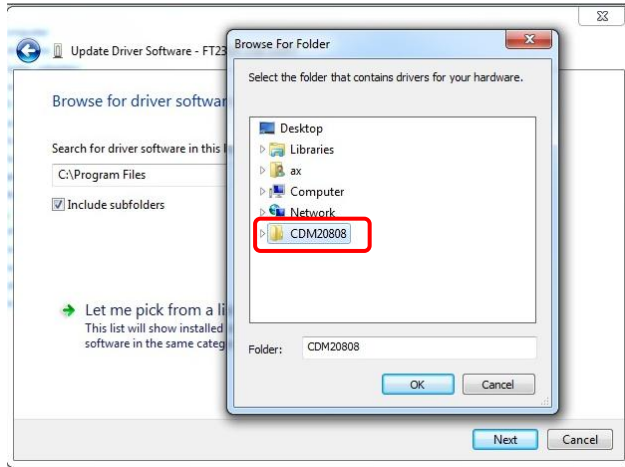
3. In the next window, select "Browse my computer for driver software" to continue to next step.



4. When you see this screen, click "Browse".



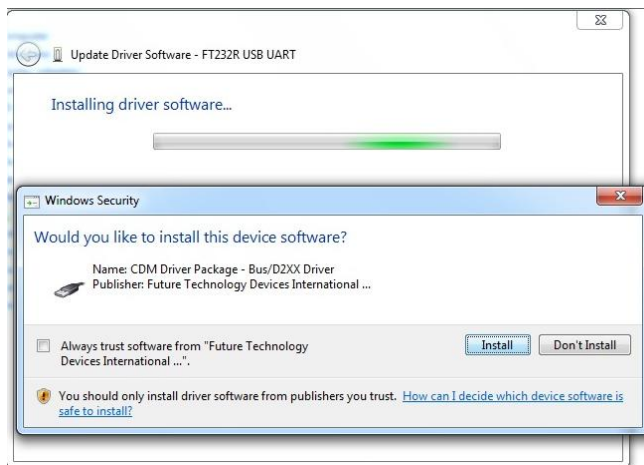
5. Select the folder that contains drivers for your hardware. In the Browse For Folder window, choose “CDM20808” and click “OK”.



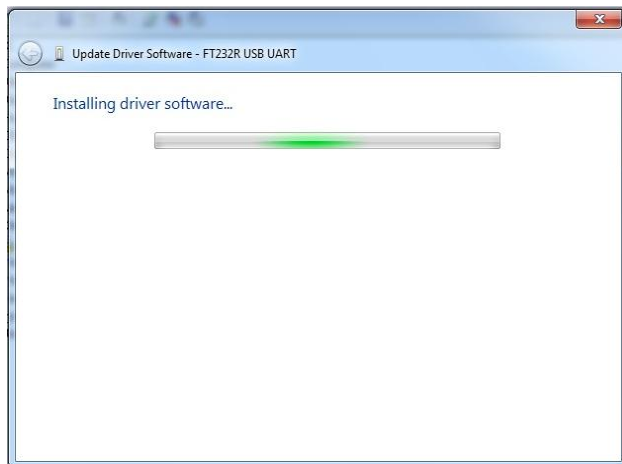
6. The selected folder path is automatically added into folder location bar. Click “Next” to continue.



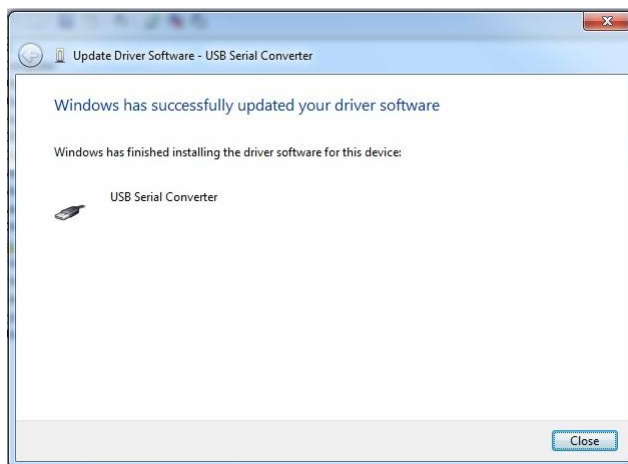
7. When you see the following screen asking “Would you like to install this device software?”, click “Install” to continue.



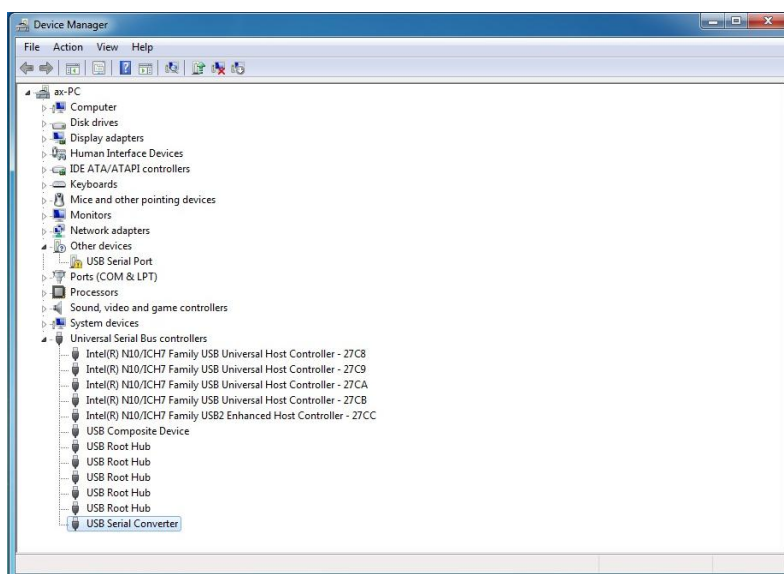
8. The setup program is installing driver software, please wait.



9. The setup program has finished installing the driver software, click "Close" to continue.

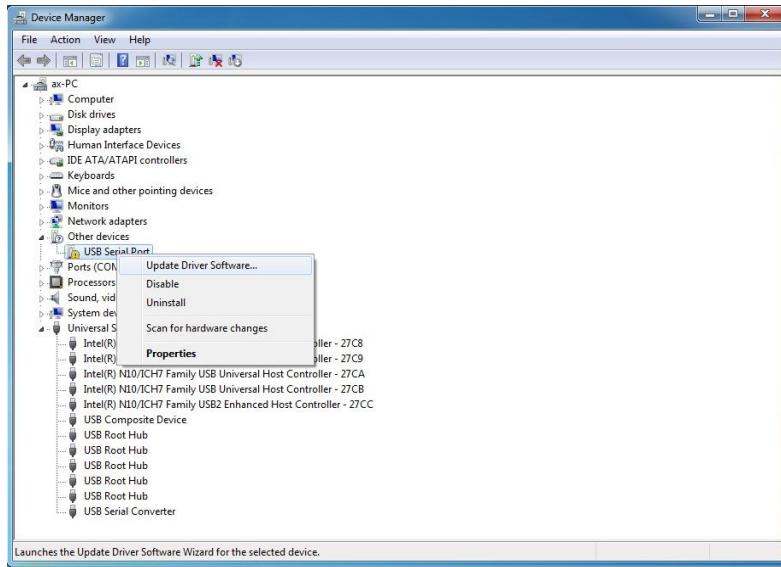


10. In the next window, you can see the following screen.

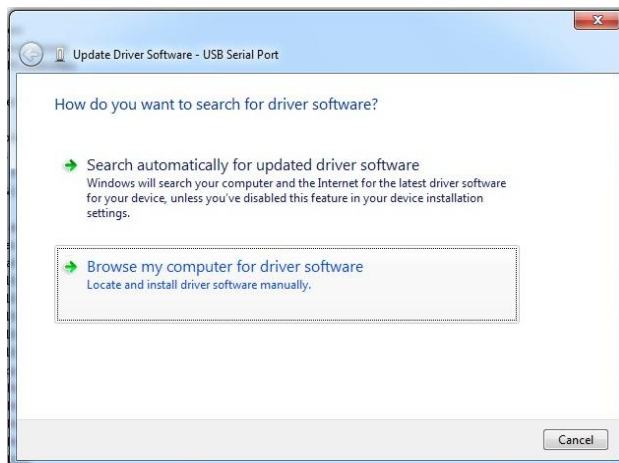




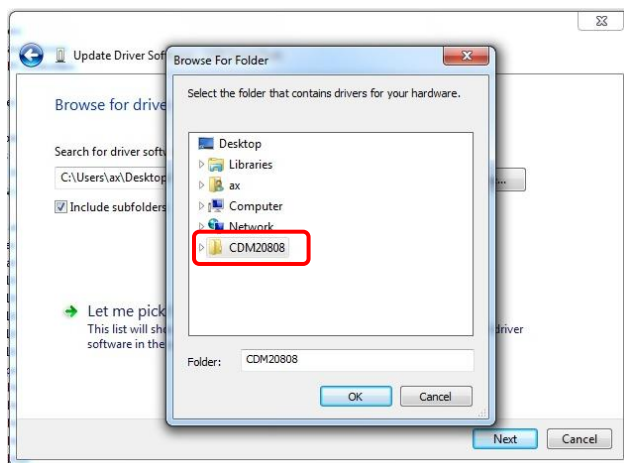
- Again right click on *Other devices* > *USB Serial Port* and launch Update Driver Software for the selected device.



- In the next window, select "Browse my computer for driver software" to continue to next step.



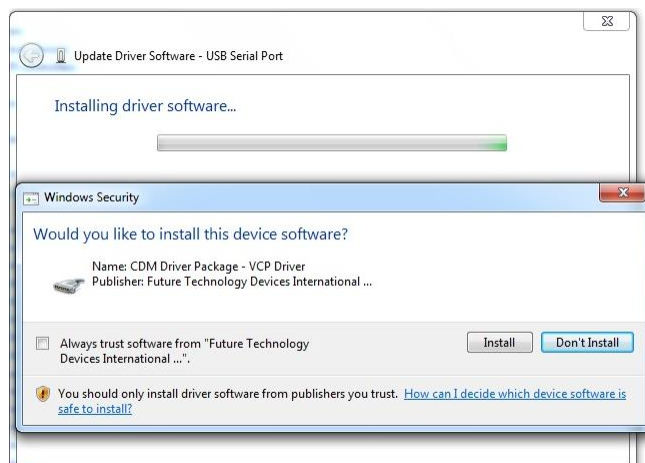
- Again select "CDM20808" folder which contains your driver files and click OK.



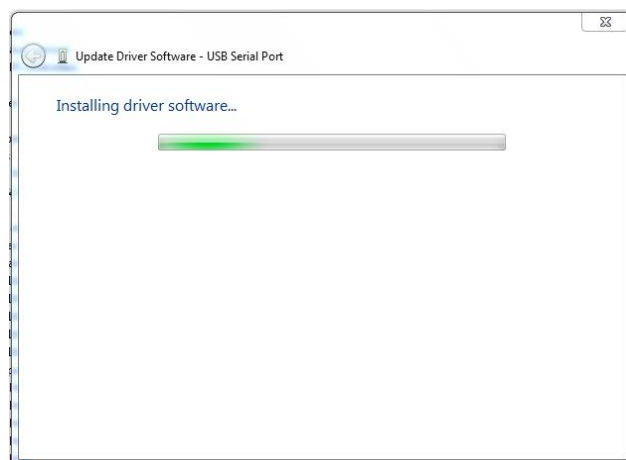
14. The selected folder path is automatically added into folder location bar. Click “Next” to continue.



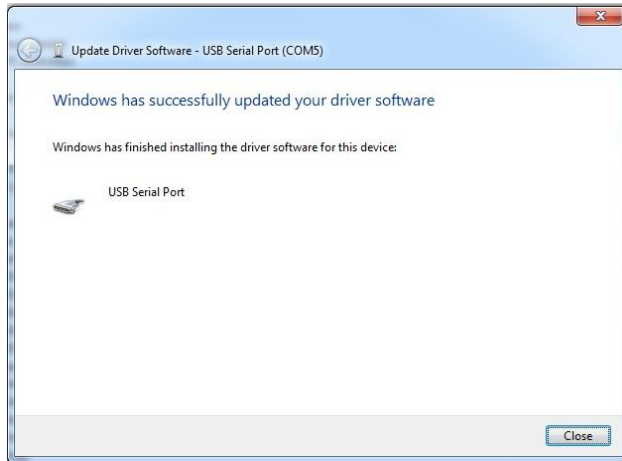
15. When you see the following screen asking “Would you like to install this device software?”, click “Install” to continue.



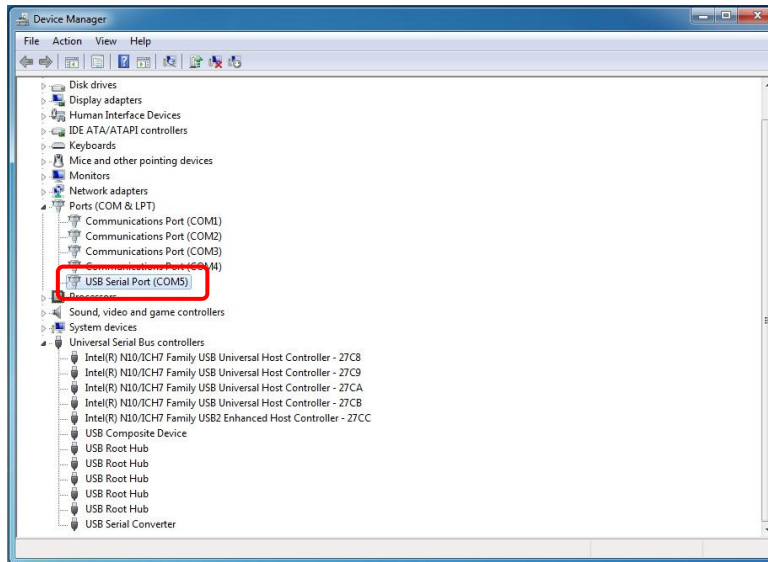
16. The setup program is installing driver software, please wait.



17. The setup program has finished installing the driver software, click “Close” to continue.



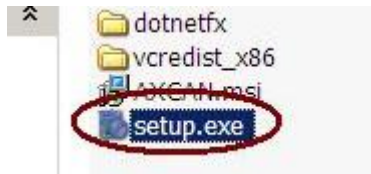
18. You can see that a new “USB Serial Port (COM5)” device is successfully added to *Device Manager > Ports (COM & LPT)*.



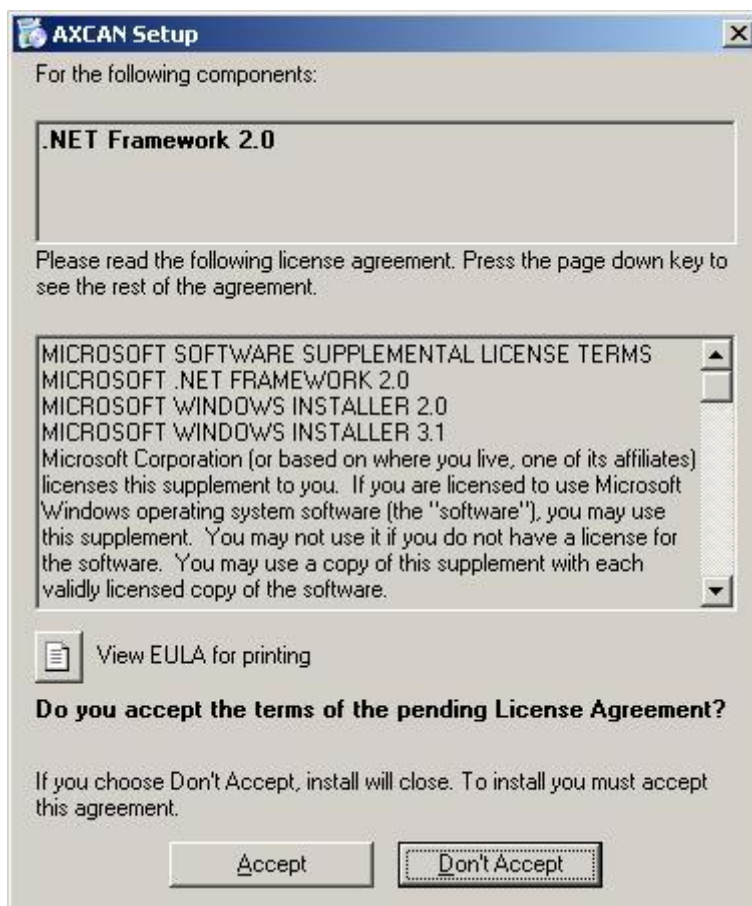
## 3.2 Utility Installation

This section describes how to install AXCAN software utility. Please follow the step by step instructions given below.

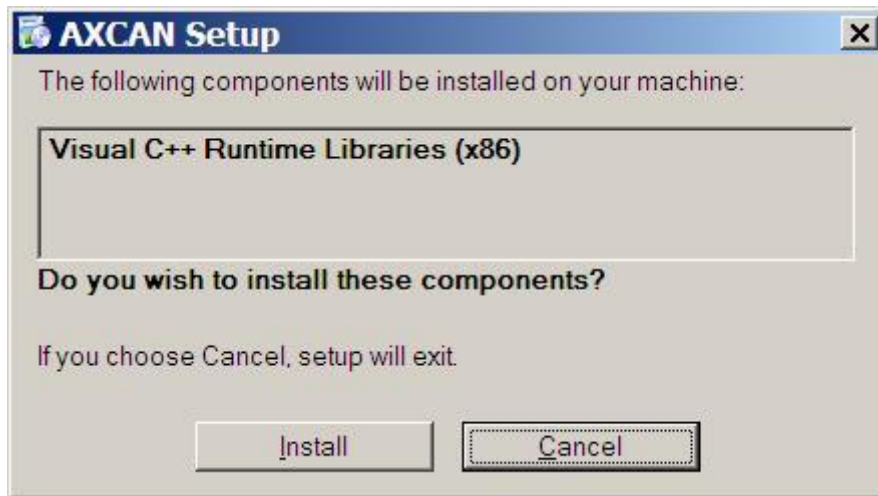
1. Double click the setup icon to run setup.exe program and begin installation.



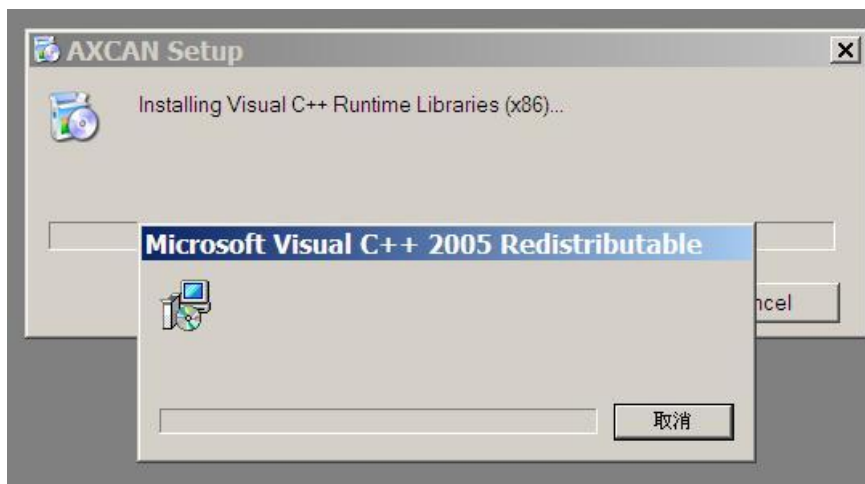
2. The setup program automatically detect whether .NET Framework 2.0 exists in your computer. Read the license agreement and make your choice.



3. Click "Install" to install Visual C++ Runtime Libraries (x86).



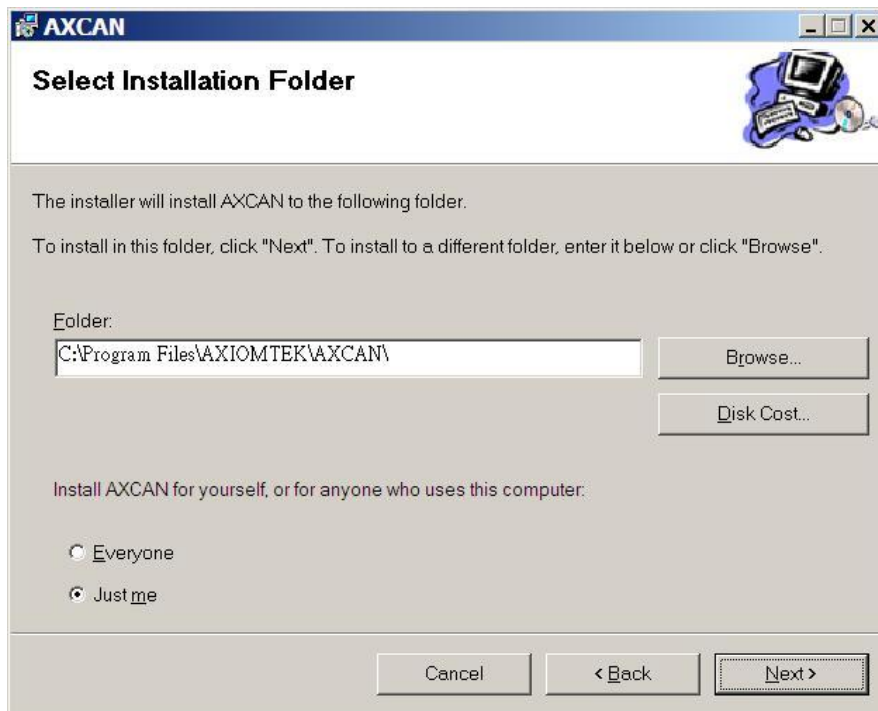
4. The setup program is installing files of Visual C++ Runtime Libraries (x86) into your computer.



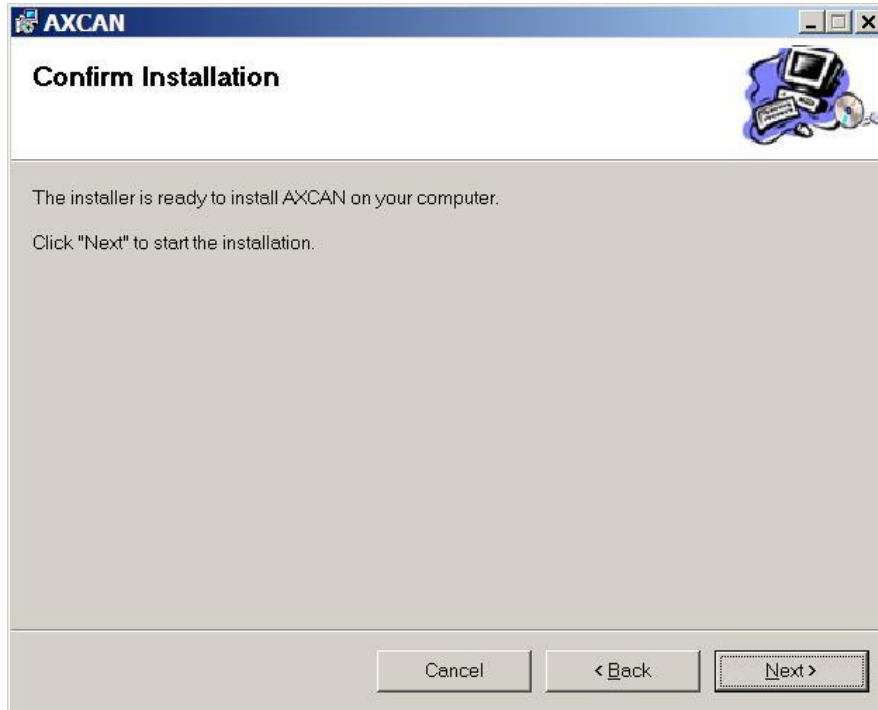
5. After all of the necessary components are installed, let's continue with AXCAN utility setup wizard.



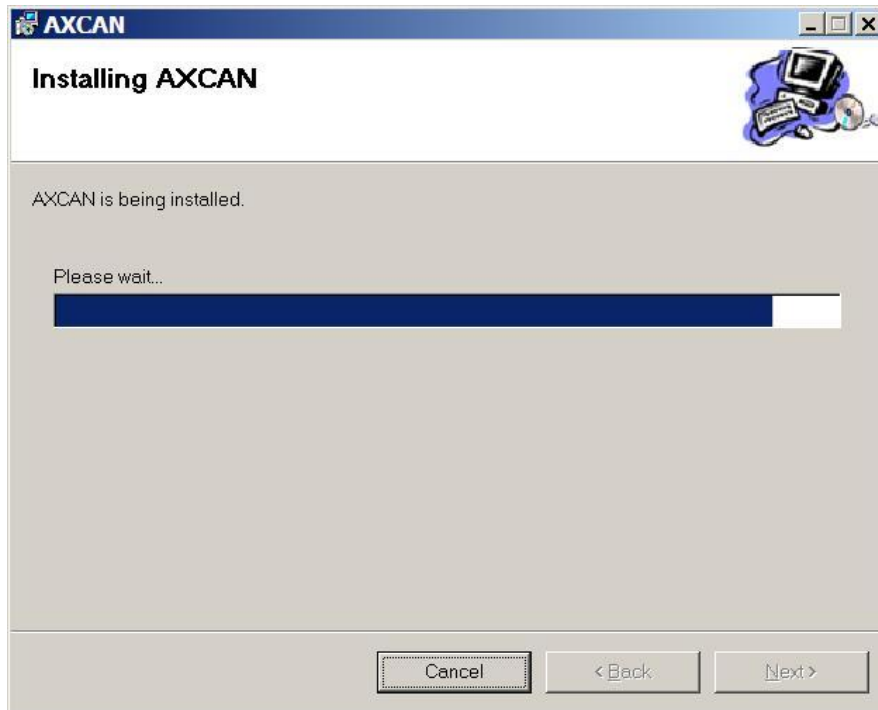
6. By default, the utility is installed to C:\Program Files\AXIOMTEK\AXCAN. Change the location if needed. Select the permission to install AXCAN for yourself or for anyone who uses this computer. Click "Next" to continue



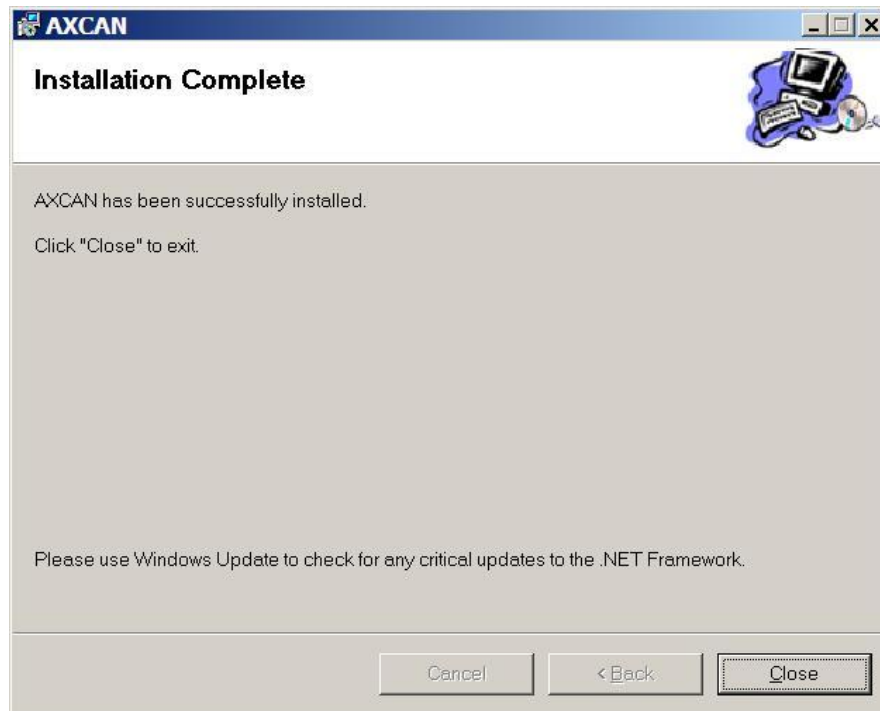
7. Now the setup program is ready to install AXCAN software utility. Click "Next" to continue.



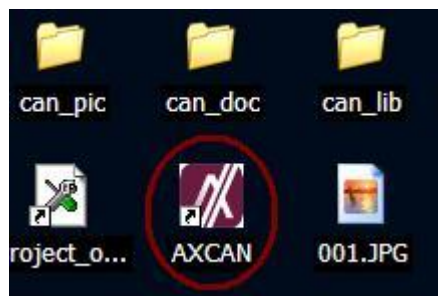
8. The setup program is installing AXCAN software utility. Wait until the installation is complete before continuing to the next step.



9. The AXCAN software utility has been successfully installed. Click "Close" to exit.



10. After installation, a shortcut icon is automatically created on your desktop.



11. The setup program also creates program menu shortcut of AXCAN.





# Chapter 4

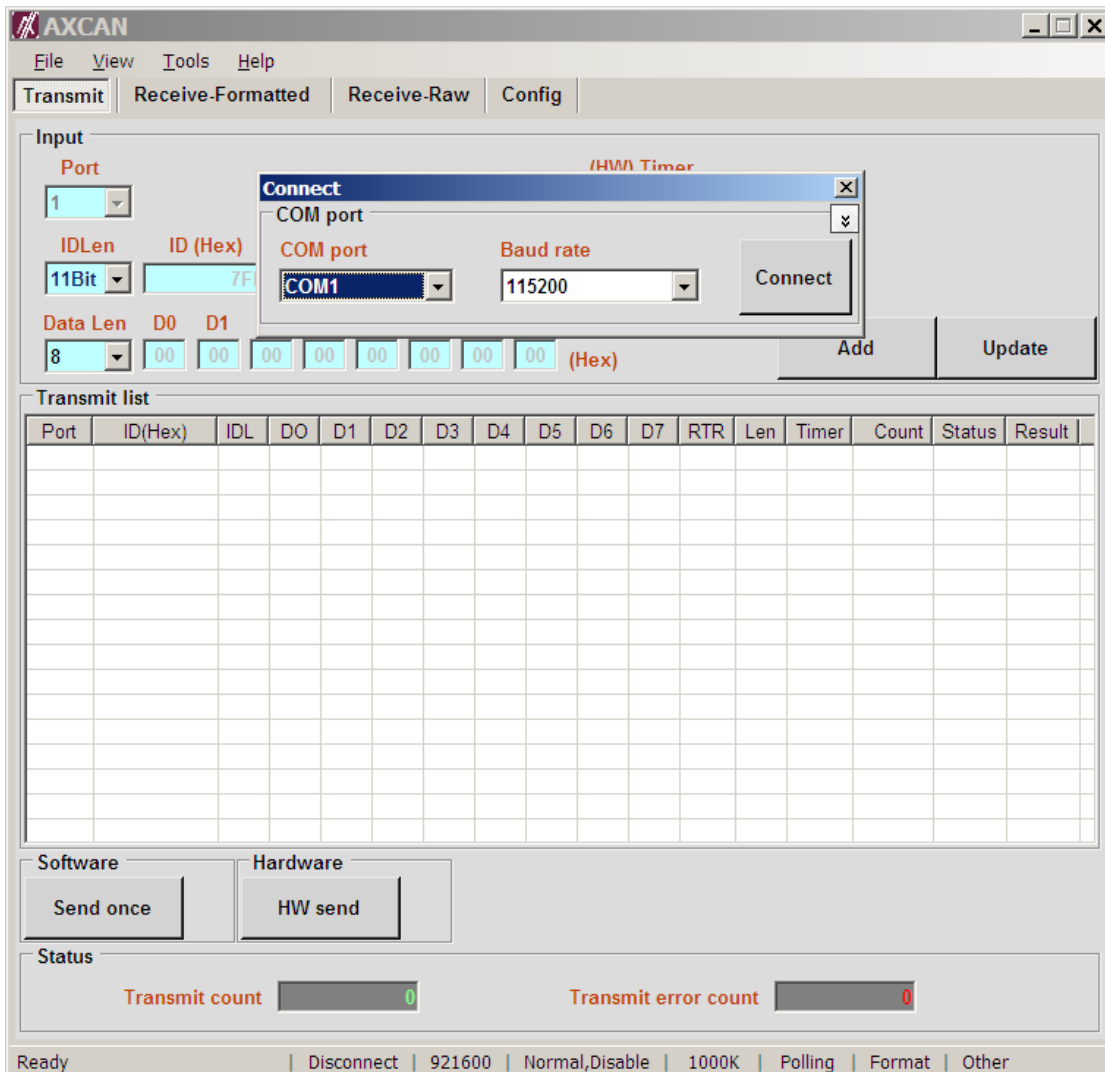
## Using Software Utility

### 4.1 Overview



Before you start using AXCAN software utility, please check the hardware connection:

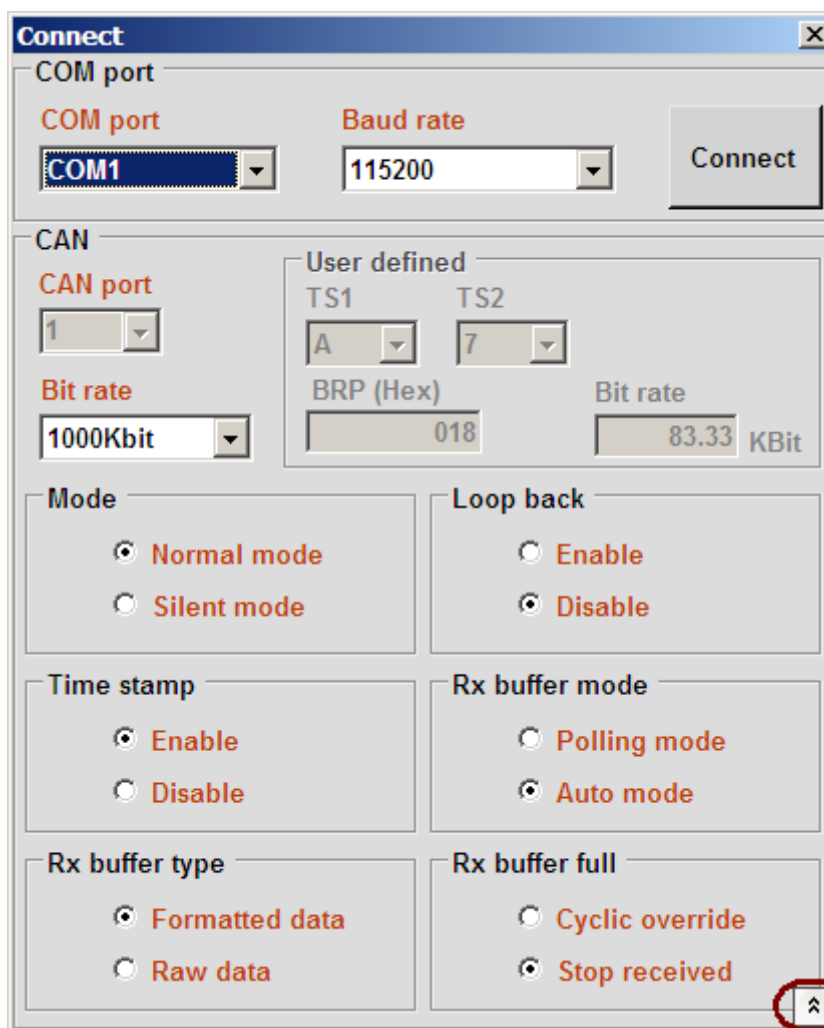
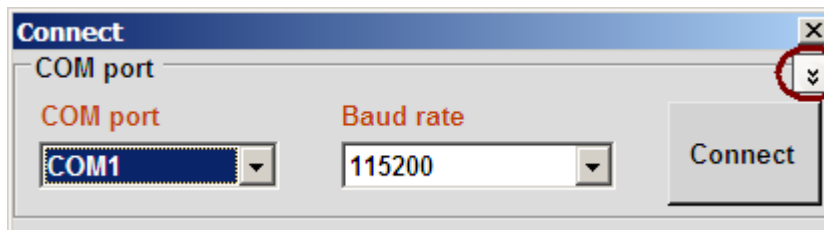
- Firmly install the AX92903 into computer's PCI-Express Mini Card socket.
- Firmly connect the AX92903 to CAN devices.

Use this utility to configure CAN communication parameters. It is also used to transmit or receive CAN messages for simple function testing. Open utility program by clicking AXCAN.exe.



## 4.2 Connect Setting Window

Find the  button on Connect Setting window. Click this button to open more detailed hidden settings window. To close, click on the  button. Use the parameters on this window for connection settings.



Enter connection parameters properly, see below information, then click the Connect button. The software automatically saves all settings when the CAN connection is established successfully.

### COM port

Use this item to select COM port no. or user's virtual COM port no. which is an interface between device and a PC.

**Baud rate**

Select COM port baud rate.

**CAN port**

Select CAN port no. which is an interface between device and device.

**Bit rate**

Select CAN communication bit rate. Bit rate is the number of data bits transmitted per second in CAN communication channel.

**User-defined : TS1, TS2, BRP**

User can define bit rate based on this parameters. Bit rate =  $(36M / ((TS1+TS2+1) * BRP))$ .

**CAN Mode**

Normal mode is for CAN normal operation. In silent mode, the device is set as receive-only.

**Loop back**

Enable/disable loopback. If loopback is enabled, the device is set as transmit-only. It receives only messages routed internally from TX to RX.

**Time stamp**

Enable/disable timestamp. Time stamp is the time at which message is received by PC (per bit rate).

**Rx buffer mode**

Polling mode	Device sends message back upon PC request.
Auto mode	Device sends messages back to PC automatically. In auto mode, you cannot transmit data.

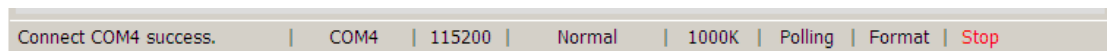
**Rx buffer type**

Formatted Data	Device sends back formatted data.
Raw Data	Device sends back raw data.

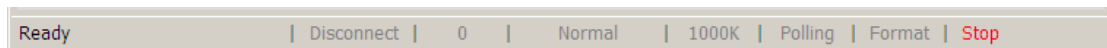
**Rx buffer full**

Cyclic override	Overwrite old data when device buffer is full.
Stop received	Stop receiving data when device buffer is full.

The status bar shows “Connect COMx success” message, when success connection is established. You can also see the connection parameters on the status bar.



If connection attempt failed, the status bar shows “Disconnect” message. And at the same time, the connection parameters show up in gray.





- Input**

Enter proper parameters and click Add button to add data value into Transmit list. Or you can directly click on each item in Transmit list to update data. All items in Transmit list have default values and protection against out-of-range input data.

**Port**

This is the same CAN port specified in Connect Setting window.

**IDLen**

This is message ID type selection; 11-bit(CAN 2.0A) or 29-bit(CAN 2.0B).

**ID**

This is message ID in hexadecimal.

**RTR**

Determine whether RTR message is set or not. Note data length of RTR message without data field is 0.

**Data Len**

Transmit data length.

**D0~D7**

CAN data in hexadecimal.

**(HW) Timer**

Transmit interval in millisecond. This item is only suitable for hardware (HW) send.

**(HW) Count**

Set the number of times the data is transmitted (for hardware (HW) send).

- Transmit List**

All added data is filled into below Transmit list.

Transmit list																
Port	ID(Hex)	IDL	D0	D1	D2	D3	D4	D5	D6	D7	RTR	Len	Timer	Count	Status	Result
Port1	123	11b	11	11	11	11	11	11	11	11	N	8	1000	1	Y	
Port1	123	11b	11	11	11	11	11	11	11	11	N	8	1000	1	Y	
Port1	123	11b	11	11	11	11	11	11	11	11	N	8	1000	1	N	
Port1	123	11b	11	11	11	11	11	11	11	11	N	8	1000	1	Y	
Port1	123	11b	11	11	11	11	11	11	11	11	N	8	1000	1	Y	

**Port**

This is the same CAN port specified in Connect Setting window.

**ID**

This is message ID in hexadecimal.

**IDL**

This is message ID type selection; 11-bit(CAN 2.0A) or 29-bit(CAN 2.0B).

**D0~D7**

CAN data in hexadecimal.

**RTR**

Indicate whether RTR message is set or not.

**Len**

Transmit data length.

**Timer**

Transmit interval in millisecond.

**Count**

The number of times the data is transmitted.

**Status**

This option is status of whether to transmit data or not. If disabled, data selected in Transmit list will show up in gray and will be ignored during data transmission.

**Result**

Show result of data transmission. If error occurred, double click (with mouse left button) on list item to display error message box. If no error occurred, you won't see any error message.

Right click on list item to show additional function.

ID	D0	D1	D2	D3	D4	D5	D6	D7	RTR	Len
1	11	11	11	11	11	11	11	11	N	8
1	11	11	11	11	11	11	11	11	N	8
1	11						11	11	N	
1	11						11	11	N	
1	11									

**Enable/Disable**

Enable or disable selected row.

**Delete**

Delete selected row from list.

**Clean\Both**

Remove both; data and status data from selected row.

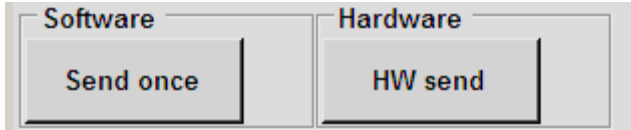
**Clean\List**

Remove data from selected row.

**Clean\Status**

Remove status data from selected row.

- **Action**



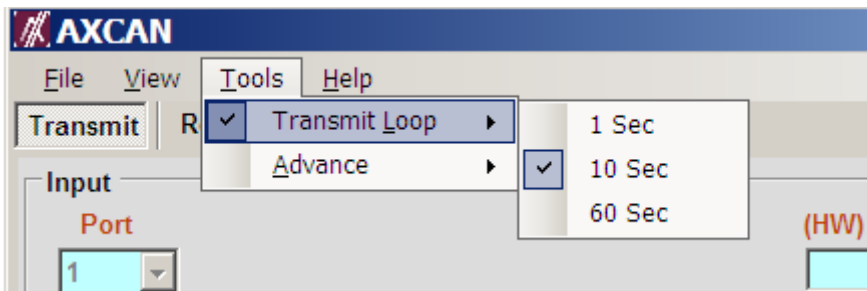
**Send once**

Click this button to send data in list regardless of (HW) Timer and (HW) Count settings.

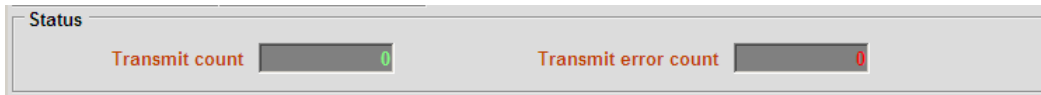
**HW Send**

Click this button to send data in list based on (HW) Timer and (HW) Count settings.

To transmit data continuously, check Tools menu\Transmit Loop\1 or 10 or 60 sec. Then click Send once or HW send button to transmit data repeatedly. The transmitting will not stop until Stop loop button is pressed.



- **Status**



**Transmit count**

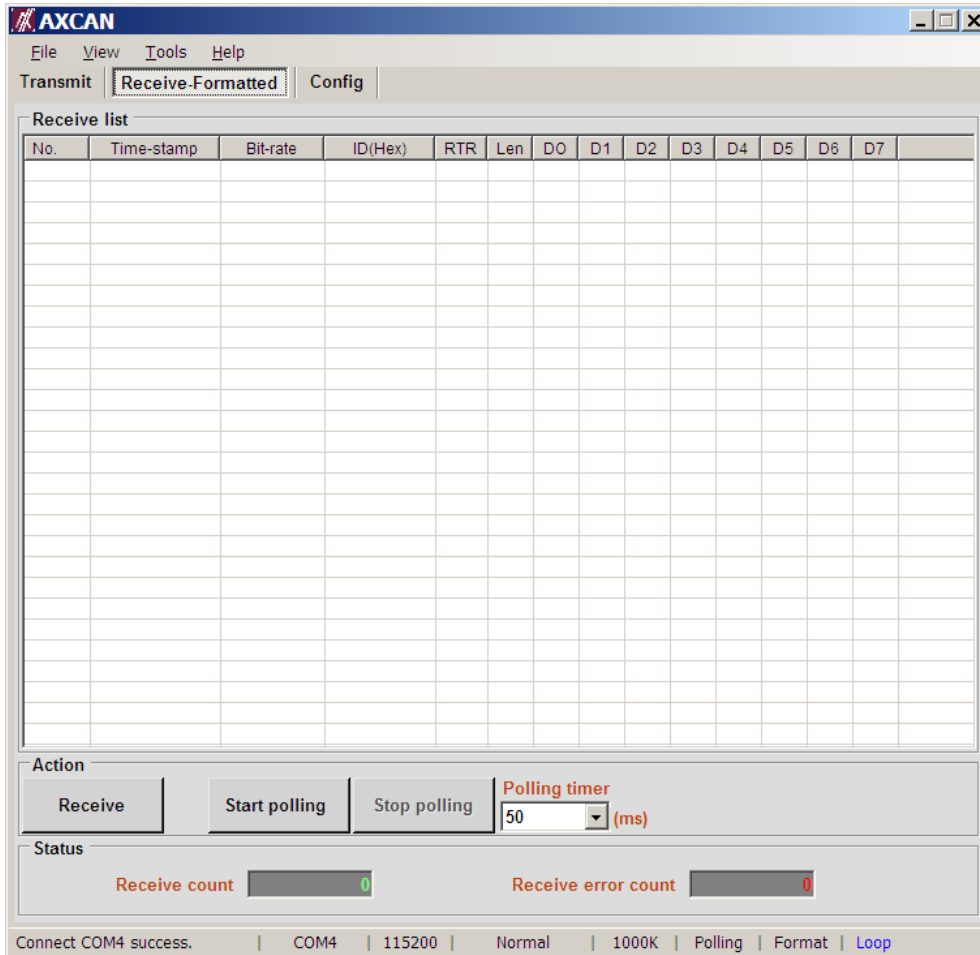
This is the total amount of transmitted data.

**Transmit error count**

The amount of transmit errors.

## 4.4 Receive-Formatted Setting Window

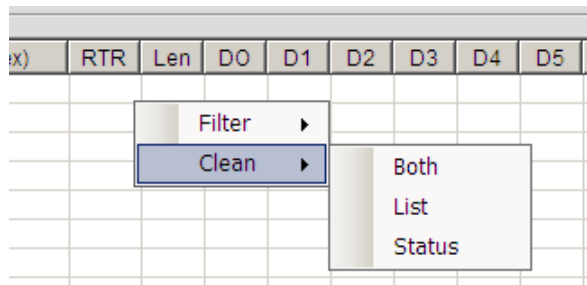
If Rx buffer type is set to formatted data, this page switches to Receive-Formatted Setting page. This page consists of 3 function groups: List, Action and Status.







Right click on list item to show additional function.



**Filter**

This function is for software filter setting. More detailed information is given below.

**Clean\Both**

Remove both; data and status data from selected row.

**Clean>List**

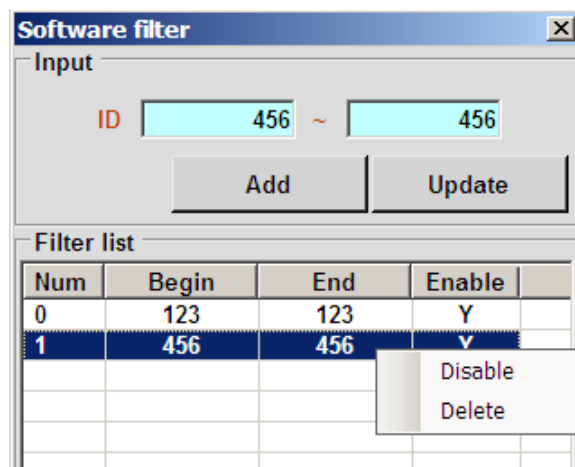
Remove data from selected row.

**Clean>Status**

Remove status data from selected row.

**Software filter**

Adds filters to block out of range values, so only the data in range will be shown in Receive list. The valid range of data is 0~1FFFFFFF (hex).



In addition, there is a right-click menu over the Filter list which allows you to choose Disable or Delete function.

**Enable/Disable**

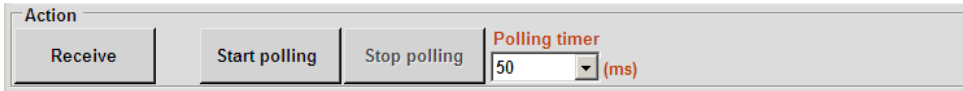
Enable or disable range filter.

**Delete**

Remove the selected range filter.

- **Action**

When the Rx buffer is in Polling mode, the Action page appears as follow.



When the Rx buffer is in Auto mode, the Action page appears as follow.



**Receive**

Click this button to request device to send data.

**Start polling**

Click this button to start polling. It requests device to send data continuously based on time interval in Polling timer.

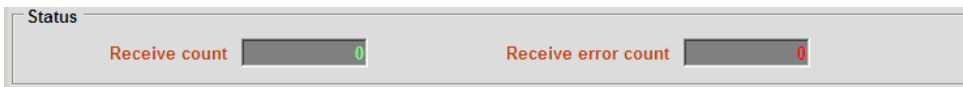
**Stop polling**

Click this button to stop polling.

**Stop auto-mode**

Click this button to stop Auto mode and switch back to Polling mode.

- **Status**



**Receive count**

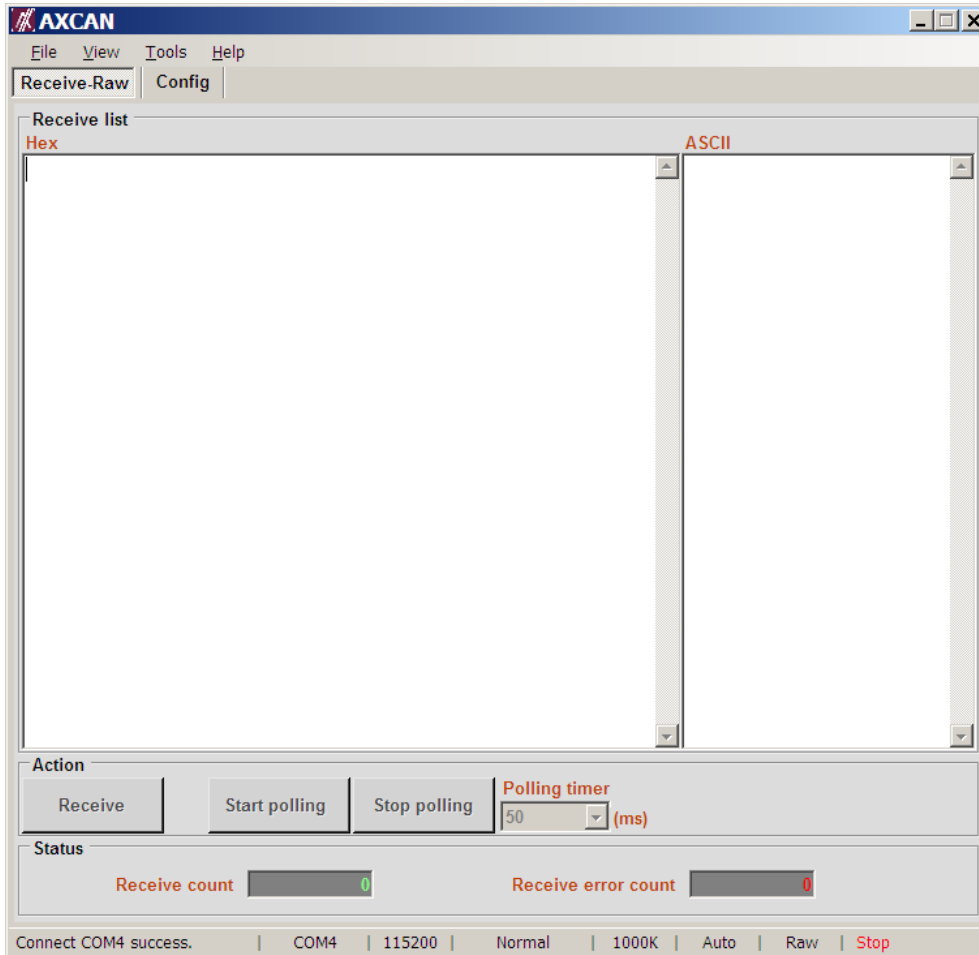
This is the total amount of received data.

**Receive error count**

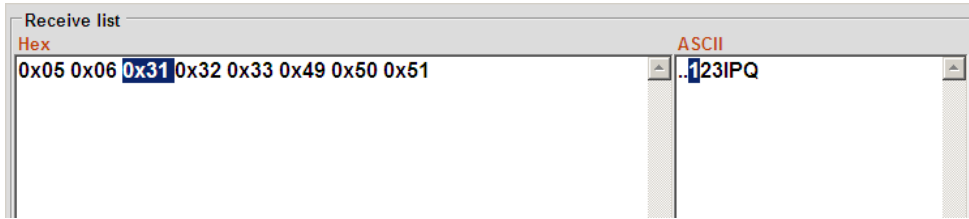
The amount of receive errors.

## 4.5 Receive-Raw Setting Window

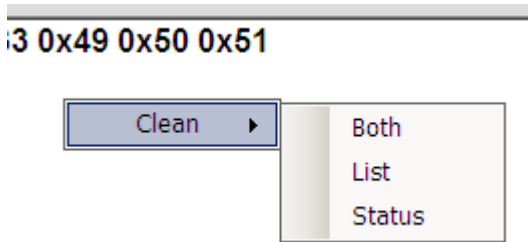
If Rx buffer type is set to raw data, this page switches to Receive-Raw Setting page. This page consists of 3 function groups: List, Action and Status.



- **List**  
Received raw data (in hexadecimal and ASCII) is shown in this list.



Select Hex data and at the same time its corresponding ASCII data is also selected. Conversely, if you select ASCII data, its mapped Hex data is also selected. In addition, there is a right-click menu over the Receive list which allows you to use Clean function.



**Clean\Both**

Remove both; selected data and status of received data.

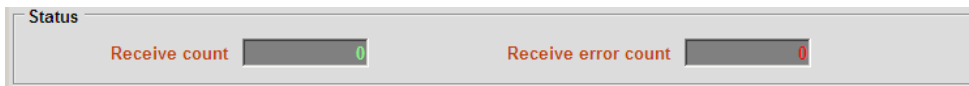
**Clean>List**

Remove selected data.

**Clean>Status**

Remove status of received data.

- **Status**



**Receive count**

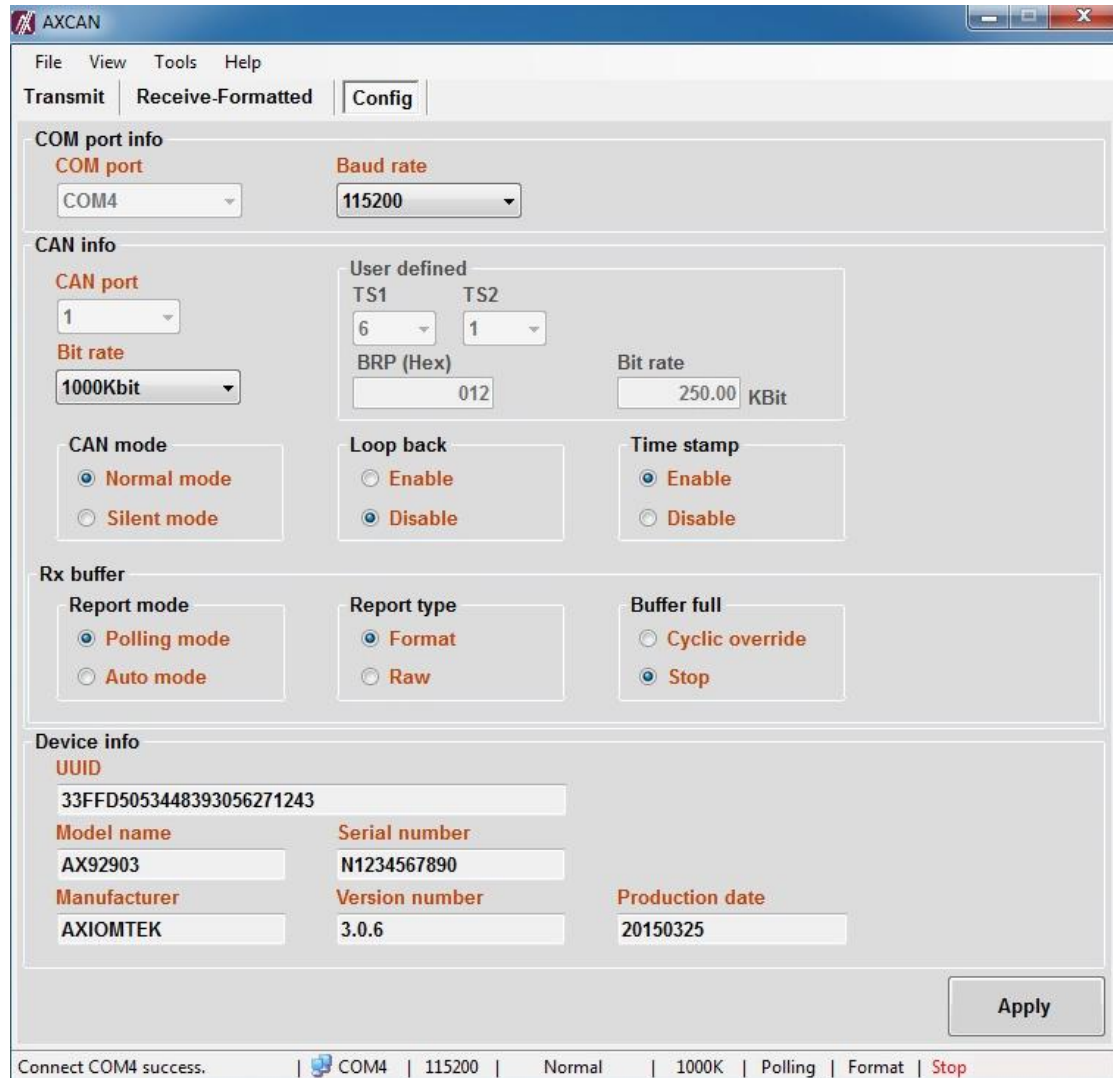
This is the total amount of received data.

**Receive error count**

The amount of receive errors.

## 4.6 Config Setting Window

You can use functions in this window only if the connection is successful. This page consists of 2 function groups: Setting (refer to section 4.2 Connect Setting Window) and Device info.

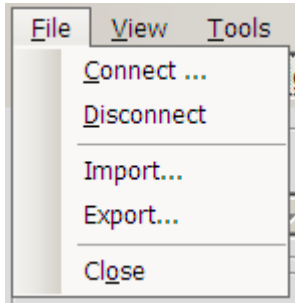


- **Device info**

It contains the following information: UUID, Model name, Serial number, Manufacturer, Version number and Production date.

## 4.7 Others

- **File Menu**



**Connect**

Click File\Connect menu to launch Connect window.

**Disconnect**

Click File\Disconnect menu to terminate current connection.

**Import**

Click File\Import menu to import setting file.

**Export**

Click File\Export menu to store current settings into a file.

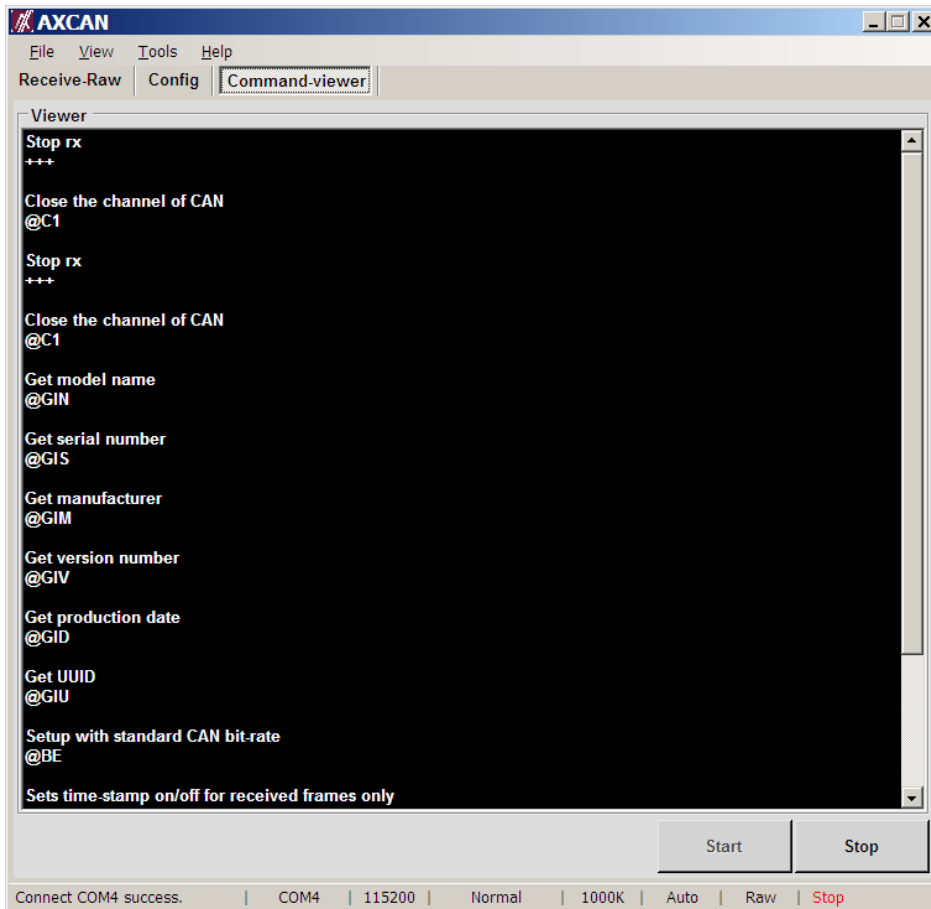
**Close**

Click File\Close menu to exit program.

- **View Menu**

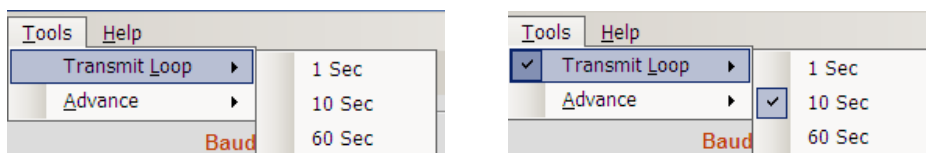


Click View\Command-viewer menu to open Command-viewer window. After the window is opened successfully, the Command-viewer item will be checked automatically.



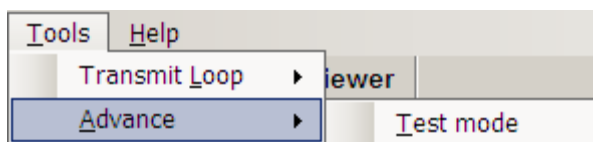
- **Tools Menu**

**Transmit Loop**



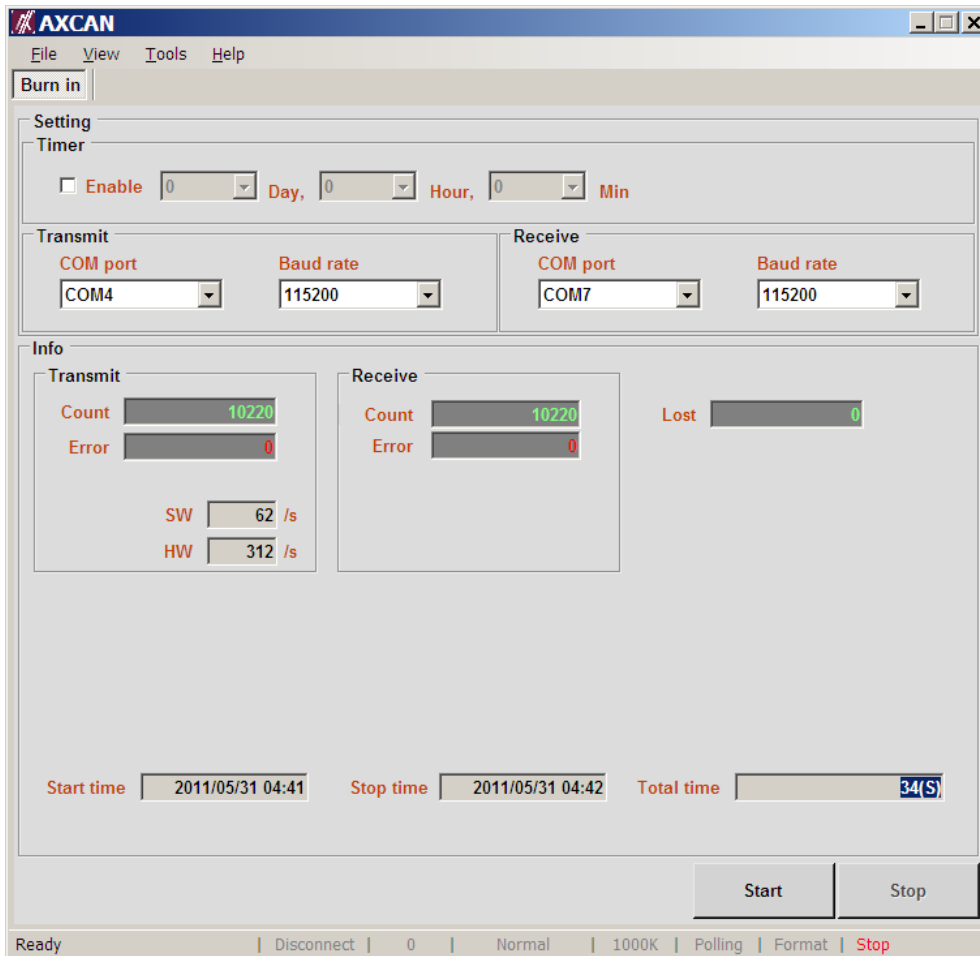
Select Tools\Transmit Loop\1 or 10 or 60 sec to trigger transmit loop function. After transmit loop has started, a check sign is added simultaneously.

**Test Mode**



Click Tools\Advance\Test mode to trigger burn in operation.





### Timer

Enter timer value using the Day, Hour and Minute combo box. You can enable/disable the time duration of device testing. If this timer is enabled, the device testing stops at the end of the time duration. Or you can press Stop button to terminate immediately. If this timer is disabled, the device testing terminates only when Stop button is pressed.

### Transmit and Receive

Please refer to section 4.2 Connect Setting Window.

### Info\Transmit\Count

This is the total amount of transmitted data.

### Info\Transmit>Error

The amount of transmit errors.

### Info\Transmit\SW

This is the total amount of transmitted data per second via software.

### Info\Transmit\HW

This is the total amount of transmitted data per second via hardware.

### Info\Receive\Count

This is the total amount of received data.

**Info\Receive\Error**

The amount of receive errors.

**Info\Lost**

This is the total amount of messages lost.

**Info\Start time**

This is the start time information.

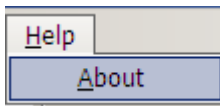
**Info\Stop time**

This is the stop time information.

**Info\Total time**

This is the total amount of burn in time.

- **Help Menu**



**Help>About**

Click Help>About menu to display About AXCAN window.



# Chapter 5

## Commands Available

### 5.1 Overview

To meet user's application purpose, commands are provided for CAN module management, transmitting and receiving messages through AX92903. Each command consists of 3 parts: StartByte, ContentOfCommand and StopByte.

StartByte	ContentofCommand	StopByte
@	CMD	0x0d (carriage return)

Operation can be set to command and data mode. In command mode, CAN module accepts commands and this is the default mode. Data mode is the mode that only and automatically report CAN messages to UART port or USB port based on settings until user inputs command string '+++'.  
**Command Mode**

#### Command Mode

In general, CAN module stays at command mode as default and accepts all commands.

#### Data Mode

Once CAN port is opened in auto report mode, it enters data mode and doesn't accept any command input except string '+++' in order to switch from data mode back to command mode.

## 5.2 Summary Table of All Commands

No.	Command	Description
1	Un	Set UART baud rate, where n is 0~A (Default: U7).
2	Bn	Set CAN bit rate, where n is 0~E and M (Default: BE).
3	bxyzzz	Set TS1, TS2 and BRP parameters for user-defined CAN bit rate, where x, y and zzz are in hexadecimal.
4	Sx	Set report method, data format and receiving behavior if buffer is full (Default: S0).
5	Onxy	Open CAN port n with arguments x and y.
6	Cn	Close CAN port n.
7	tnpppprrriiidd...	Transmit 2.0A CAN messages without RTR through CAN port n.
8	tnpppprrriiOR	Transmit 2.0A CAN messages with RTR bit 1 through CAN port n.
9	Tnpppprrriiiiiidd...	Transmit 2.0B CAN messages without RTR through CAN port n.
10	TnpppprrriiiiiOR	Transmit 2.0B CAN messages with RTR bit 1 through CAN port n.
11	Tx	Inform CAN controller to stop transmitting messages if repeat counter is not zero.
12	Fnnxi..i	Set acceptance filter as identifier list mode.
13	Mnnxi..i	Apply mask mode to specified acceptance filter.
14	Wppxxxllidd...	Write data to NVRAM (64Kbyte in total).
15	Rppxxxlll	Read data from NVRAM.
16	Eppxxxlll	Erase NVRAM.
17	SC	Save settings to NVRAM.
18	Zn	Enable timestamp of received messages.
19	SITnx	Configure termination of specified CAN port.
20	GITn	Read termination setting of specified CAN port.
21	SIEx	Configure the CAN Error LED.
22	+++	Escape from data mode to command mode.
23	SIB+++	Reset CAN interface (module).
24	Glx	Retrieve CAN system information.
25	Gx	Retrieve current CAN status and settings.

## 1. Command: Un

<b>Command</b>	<b>Un</b>
<b>Description</b>	Set UART baud rate, where n is 0~A (Default: U7). <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	@U0: 1200 bps @U1: 2400 bps @U2: 4800 bps @U3: 9600 bps @U4: 19200 bps @U5: 38400 bps @U6: 57600 bps @U7: 115200 bps @U8: 230400 bps @U9: 460800 bps @UA: 921600 bps
<b>Example</b>	@U6 Set UART baud rate to 57600 bps.
<b>Return</b>	OK or ERRORxx
<b>Others</b>	None

## 2. Command: Bn

<b>Command</b>	<b>Bn</b>
<b>Description</b>	<p>Set CAN bit rate, where n is 0~E and M (Default: BE).</p> <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ This command must be used prior to open CAN port.</li> </ul>
<b>Arguments</b>	<p>@B0: 5 Kbit                  @B1: 10 Kbit                  @B2: 20 Kbit                  @B3: 40 Kbit                  @B4: 50 Kbit                  @B5: 80 Kbit                  @B6: 100 Kbit                  @B7: 125 Kbit                  @B8: 200 Kbit                  @B9: 250 Kbit                  @BA: 400 Kbit                  @BB: 500 Kbit                  @BC: 600 Kbit                  @BD: 800 Kbit                  @BE: 1000 Kbit                  @BM: User-defined</p>
<b>Example</b>	<p>@B4                  Set CAN bit rate to 50 Kbit.</p>
<b>Return</b>	OK or ERRORxx
<b>Others</b>	None

### 3. Command: bxyzzz

<b>Command</b>	<b>bxyzzz</b>
<b>Description</b>	Set TS1, TS2 and BRP parameters for user-defined CAN bit rate, where x ,y and zzz are in hexadecimal. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ This command must be used prior to open CAN port.</li> </ul>
<b>Arguments</b>	x: TS2 (Time Segment 2), its data range is from 1 to 7. y: TS1 (Time Segment1), its data range is from 1 to F. zzz: BRP (Baud Rate Prescaler), its data range is from 1 to 3FF.  Bit rate = Fcl/(BRP(TS1+TS2+1)), where Fcl=36MHz.
<b>Example</b>	@b16012 Set TS2=0x1, TS1=0x6 and BRP=0x12, and calculate according to above formula. Result is the user-defined CAN bit rate = 250Kbit.
<b>Return</b>	OK or ERRORxx
<b>Others</b>	None

### 4. Command: Sx

<b>Command</b>	<b>Sx</b>
<b>Description</b>	Set report method, data format and receiving behavior if buffer is full (Default: S0). <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	x: Its data range is from 0 to 7 in hexadecimal. Bit[0]: Report method 0: Polling; report one message per polling 1: Auto; report automatically Bit[1]: Data format 0: Report formatted data 1: Report raw data Bit[2]: Receiving behavior if buffer is full. 0: Overwrite the old data 1: Suspend receiving until receiving space is available
<b>Example</b>	@S5 Set CAN to auto-report, report formatted data and overwrite buffer if it is full.
<b>Return</b>	OK or ERRORxx
<b>Others</b>	None

## 5. Command: Onxy

<b>Command</b>	<b>Onxy</b>
<b>Description</b>	Open CAN port n with arguments x and y. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Before using this command, please make sure that the CAN port has been initialized via @Bn or @bxyzzz command.</li> </ul>
<b>Arguments</b>	n: CAN port number, only 1 is allowed for AX92903 series. x: Enable/disable silent mode. Set/unset CAN port as transmit-only. 0: Disable silent mode. This is the default normal mode. 1: Enable silent mode. CAN port is not allowed to receive messages. y: Enable/disable loopback mode. 0: Disable loopback mode 1: Enable loopback mode
<b>Example</b>	@O101 Open CAN port 1 and set it to operate at normal mode with loopback.
<b>Return</b>	OK or ERRORxx
<b>Others</b>	None

## 6. Command: Cn

<b>Command</b>	<b>Cn</b>
<b>Description</b>	Close CAN port n <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	n: CAN port number, only 1 is allowed for AX92903 series.
<b>Example</b>	@C1 Close CAN port 1
<b>Return</b>	OK or ERRORxx
<b>Others</b>	None



## 7. Command: tnpppprrriiidd...

<b>Command</b>	<b>tnpppprrriiidd...</b>
<b>Description</b>	Transmit 2.0A CAN messages without RTR through CAN port n. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port has opened.</li> </ul>
<b>Arguments</b>	n: CAN port number, only 1 is allowed for AX92903 series. pppp: Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms). rrrr: Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Send once iii: Identifier in hexadecimal. Its range is 000~7FF. l: Data length in hexadecimal. Its range is 0~8. dd: One data byte in hexadecimal. Its range is 00~FF. dd fields must match the setting of data length l.
<b>Example 1</b>	@t10000000010021133 Send one 2.0A CAN message with 11-bit ID 0x100, 2 bytes data 0x11 and 0x33 through CAN port 1.
<b>Example 2</b>	@t100010005002133 Send one 2.0A CAN message with 11-bit ID=0x002 and 1 byte data 0x33 every 0.1ms for 5 times through CAN port 1.
<b>Return</b>	OK or ERRORxx
<b>Others</b>	The @GR command can be used to pull data out of receive buffer.

## 8. Command: tnpppprrrii0R

<b>Command</b>	<b>tnpppprrrii0R</b>
<b>Description</b>	Transmit 2.0A CAN messages with RTR bit 1 through CAN port n. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port has opened.</li> </ul>
<b>Arguments</b>	n: CAN port number, only 1 is allowed for AX92903 series. pppp: Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms). rrrr: Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Set repeat counter to one. iii: Identifier in hexadecimal. Its range is 000~7FF.
<b>Example</b>	@t1000000011000R Send one 2.0A CAN message with 11-bit ID 0x100 and RTR bit 1 through CAN port 1. Note this command sends message without data field which means the data length is zero.
<b>Return</b>	OK or ERRORxx
<b>Others</b>	The @GR command can be used to pull data out of receive buffer.

## 9. Command: Tnpppprrrrriiiiiidd...

<b>Command</b>	<b>Tnpppprrrrriiiiiidd...</b>												
<b>Description</b>	Transmit 2.0B CAN messages without RTR through CAN port n. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port has opened.</li> </ul>												
<b>Arguments</b>	<table border="0"> <tr> <td>n:</td> <td>CAN port number, only 1 is allowed for AX92903 series.</td> </tr> <tr> <td>pppp:</td> <td>Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms).</td> </tr> <tr> <td>rrrr:</td> <td>Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Send once</td> </tr> <tr> <td>iiiiiii:</td> <td>Identifier in hexadecimal. Its range is 00000000~1FFFFFFF.</td> </tr> <tr> <td>l:</td> <td>Data length in hexadecimal. Its range is 0~8.</td> </tr> <tr> <td>dd:</td> <td>One data byte in hexadecimal. Its range is 00~FF. dd fields must match the setting of data length l.</td> </tr> </table>	n:	CAN port number, only 1 is allowed for AX92903 series.	pppp:	Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms).	rrrr:	Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Send once	iiiiiii:	Identifier in hexadecimal. Its range is 00000000~1FFFFFFF.	l:	Data length in hexadecimal. Its range is 0~8.	dd:	One data byte in hexadecimal. Its range is 00~FF. dd fields must match the setting of data length l.
n:	CAN port number, only 1 is allowed for AX92903 series.												
pppp:	Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms).												
rrrr:	Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Send once												
iiiiiii:	Identifier in hexadecimal. Its range is 00000000~1FFFFFFF.												
l:	Data length in hexadecimal. Its range is 0~8.												
dd:	One data byte in hexadecimal. Its range is 00~FF. dd fields must match the setting of data length l.												
<b>Example 1</b>	@T100000000000456783616265 Send one 2.0B CAN message with 29-bit ID 0x00045678, 3 bytes data 0x61, 0x62 and 0x65 through CAN port 1.												
<b>Example 2</b>	@T100010005010203043212223 Send one 2.0B CAN message with 29-bit ID=0x01020304 and 3 bytes data 0x21, 0x22 and 0x23 every 0.1ms for 5 times through CAN port 1.												
<b>Return</b>	OK or ERRORxx												
<b>Others</b>	The @GR command can be used to pull data out of receive buffer.												

## 10. Command: TnpppprrrrriiiiiOR

<b>Command</b>	<b>TnpppprrrrriiiiiOR</b>								
<b>Description</b>	Transmit 2.0B CAN messages with RTR bit 1 through CAN port n. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port has opened.</li> </ul>								
<b>Arguments</b>	<table border="0"> <tr> <td>n:</td> <td>CAN port number, only 1 is allowed for AX92903 series.</td> </tr> <tr> <td>pppp:</td> <td>Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms).</td> </tr> <tr> <td>rrrr:</td> <td>Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Set repeat counter to one.</td> </tr> <tr> <td>iiiiiii:</td> <td>Identifier in hexadecimal. Its range is 00000000~1FFFFFFF.</td> </tr> </table>	n:	CAN port number, only 1 is allowed for AX92903 series.	pppp:	Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms).	rrrr:	Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Set repeat counter to one.	iiiiiii:	Identifier in hexadecimal. Its range is 00000000~1FFFFFFF.
n:	CAN port number, only 1 is allowed for AX92903 series.								
pppp:	Transmission intervals in hexadecimal. Its range is 0000~FFFF. Send one CAN message every pppp * 0.1ms for rrrr times. Unit: 100us (0.1ms).								
rrrr:	Repeat counter in hexadecimal. Its range is 0000~FFFF. 0000: Set repeat counter to one.								
iiiiiii:	Identifier in hexadecimal. Its range is 00000000~1FFFFFFF.								
<b>Example</b>	@T100000000123456780R Send one 2.0B CAN message with 29-bit ID 0x12345678 and RTR bit 1 through CAN port 1. Note this command sends message without data field, which means the data length is zero.								
<b>Return</b>	OK or ERRORxx								
<b>Others</b>	The @GR command can be used to pull data out of receive buffer.								

## 11. Command: Tx

<b>Command</b>	Tx
<b>Description</b>	Inform CAN controller to stop transmitting messages if repeat counter is not zero. <ul style="list-style-type: none"> <li>■ This command is only available at command mode.</li> <li>■ Command is available only if CAN port has been opened.</li> </ul>
<b>Arguments</b>	None
<b>Example</b>	@Tx Stop transmitting if repeat counter is not zero.
<b>Return</b>	OK or ERRORxx
<b>Others</b>	None

## 12. Command: Fnnxi..i

<b>Command</b>	Fnnxi..i
<b>Description</b>	Set acceptance filter as identifier list mode. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port is not opened.</li> </ul>
<b>Arguments</b>	nn: Index of filter in hexadecimal. Its data range is 00~1B. x: Format of identifier. Its data range is 0~1. 0: Standard ID (11-bit) 1: Extend ID (29-bit) i..i: Identifier
<b>Example 1</b>	@F000111 Set filter index 0 as identifier list mode for and accept identifier 0x111 only.
<b>Example 2</b>	@F00112345678 Set filter index 0 as identifier list mode for and accept identifier 0x12345678 only.
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	<ul style="list-style-type: none"> <li>■ Remember to close CAN port with the @C1 before using this command.</li> <li>■ Filter index 0x0 has been set as mask mode with identifier id 0 by default. If you want to enable some filters at identifier list mode, please do remember to set index to any other value except 0x00. Otherwise, messages won't be filtered out because of this default mask mode filter.</li> <li>■ For more detailed information, please refer to section 5.3 Identifier Filtering.</li> </ul>

### 13. Command: Mnnxi..i

<b>Command</b>	<b>Mnnxi..i</b>
<b>Description</b>	Apply mask mode to specified acceptance filter. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port is opened.</li> </ul>
<b>Arguments</b>	nn: Index of filter in hexadecimal. Its data range is 00~1B. x: Format of identifier. Its data range is 0~1. 0: Standard ID (11-bit) 1: Extend ID (29-bit) i..i: Identifier mask value 0: Don't care. The bit of identifier is not used for comparison. 1: Must match. The bit of the incoming identifier must have the same level.
<b>Example</b>	@F000123 and @M000FF0 combination. Set standard acceptance identification #0; only accepts incoming messages with identifiers from range 0x120 to 0x12F.
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	<ul style="list-style-type: none"> <li>■ Remember to close CAN port with the @C1 command before applying mask mode.</li> <li>■ Filter index 0x0 has been set as mask mode with identifier id 0 by default. If you want to enable some filters at identifier list mode, please do remember to set index to any other value except 0x00. Otherwise, no message will be filtered out because of this default mask mode filter.</li> <li>■ For more detailed information, please refer to section 5.3 Identifier Filtering.</li> </ul>

### 14. Command: Wppxxxllldd...

<b>Command</b>	<b>Wppxxxllldd...</b>
<b>Description</b>	Write data to NVRAM (64Kbyte in total) <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port is not opened.</li> </ul>
<b>Arguments</b>	pp: Number of page (2KB/page) written in hexadecimal. Its data range is 00~1F. xxx: Offset in hexadecimal. Its data range is 000~7FF. lll: Data length in hexadecimal. Its data range is 001~800. ddd: Data byte in hexadecimal. Its data range is 00~FF. Note dd must repeat for lll times.
<b>Example 1</b>	@ W00000002AA55 Write 2 bytes of data 0xAA and 0x55 to NVRAM at page 0 with offset 0x0.
<b>Example 2</b>	@W030600053132333435 Write 5 bytes of data 0x31, 0x32, 0x33, 0x34 and 0x35 to NVRAM at page 03 with offset 0x060.
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	None

## 15. Command: RppxxxIII

<b>Command</b>	<b>RppxxxIII</b>
<b>Description</b>	Read data from NVRAM. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port is not opened.</li> </ul>
<b>Arguments</b>	pp: Number of page (2KB/page) written in hexadecimal. Its data range is 00~1F. xxx: Offset in hexadecimal. Its data range is 000~7FF. III: Data length in hexadecimal. Its data range is 001~800.
<b>Example 1</b>	@ R00000002 Read 2 bytes from of NVRAM at page 0 with offset 0x0.
<b>Example 2</b>	@R03060005 Read 5 bytes from NVRAM at page 03 with offset 0x060.
<b>Others</b>	Note the return value will display in hexadecimal with 'R' as prefix character.

## 16. Command: EppxxxIII

<b>Command</b>	<b>EppxxxIII</b>
<b>Description</b>	Erase NVRAM <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port is not opened.</li> </ul>
<b>Arguments</b>	pp: Number of page (2KB/page) written in hexadecimal. Its data range is 00~1F. xxx: Offset in hexadecimal. Its data range is 000~7FF. III: Data length in hexadecimal. Its data range is 001~800.
<b>Example</b>	@ E00000002 Erase 2 bytes of NVRAM from page 0 with offset 0x0.
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	Note the return value will display in hexadecimal with 'R' as prefix character.

## 17. Command: SC

<b>Command</b>	<b>SC</b>
<b>Description</b>	<p>Save settings to NVRAM.</p> <p>The settings include:</p> <ol style="list-style-type: none"> <li>1. TimeStamp.</li> <li>2. Data report method of CAN port.</li> <li>3. Receiving behavior if buffer is full.</li> <li>4. Enable/disable timestamp information (refer to @Zn command)</li> <li>5. Baud rate of COM port</li> </ol> <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	None
<b>Example</b>	@SC
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	None

## 18. Command: Zn

<b>Command</b>	<b>Zn</b>
<b>Description</b>	<p>Enable timestamp of received messages.</p> <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> <li>■ Command is available only if CAN port is not opened.</li> </ul>
<b>Arguments</b>	<p>n: Enable/disable timestamp</p> <p>0: Disable (default)</p> <p>1: Enable</p>
<b>Example</b>	<p>@Z0</p> <p>Disable timestamp.</p>
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	None

## 19. Command: SITnx

<b>Command</b>	<b>SITnx</b>
<b>Description</b>	Configure termination of specified CAN port. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	n: CAN port number x: Enable/disable termination 0: Disable termination 1: Enable termination
<b>Example</b>	@SIT11 Enable termination of CAN port 1.
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	None

## 20. Command: GITn

<b>Command</b>	<b>GITn</b>
<b>Description</b>	Read termination setting of specified CAN port. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	n: CAN port number
<b>Example</b>	@GIT1 Must return #T0 or #T1.
<b>Returns</b>	#T0(Disabled) or #T1(Enabled)
<b>Others</b>	None

## 21. Command: SIEx

<b>Command</b>	<b>SIEx</b>
<b>Description</b>	Configure the CAN Error LED. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	x: Turn on/off CAN Error LED. 0: Turn off 1: Turn on
<b>Example</b>	@SIE0 Turn off the CAN Error LED.
<b>Returns</b>	OK or ERRORxx
<b>Others</b>	None

## 22. Command: +++

<b>Command</b>	<b>+++</b>
<b>Description</b>	Escape from data mode to command mode. <ul style="list-style-type: none"> <li>■ This command is only available in data mode.</li> </ul>
<b>Arguments</b>	None
<b>Example</b>	+++ Escape from data mode to command mode
<b>Returns</b>	None
<b>Others</b>	After sending characters '+++', user can send 0x0d carriage return to see whether string '#OK' shows on terminal or not. If '#OK' appears, it means device is already in command mode.

## 23. Command: SIB+++

<b>Command</b>	<b>SIB+++</b>
<b>Description</b>	Reset CAN interface (module). <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	None
<b>Example</b>	@SIB+++ Module reboots.
<b>Returns</b>	Rebooting...!
<b>Others</b>	None

## 24. Command: Glx

<b>Command</b>	<b>Glx</b>
<b>Description</b>	Retrieve CAN system information. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	x: Its definition is as follows: N: Model name S: Serial number U: Unique ID(UUID) M: Manufacturer V: Version D: Production Date
<b>Example</b>	@GIV Retrieve version information.
<b>Returns</b>	V3.0.6
<b>Others</b>	None



## 25. Command: Gx

<b>Command</b>	<b>Gx</b>
<b>Description</b>	Retrieve current CAN status and settings. <ul style="list-style-type: none"> <li>■ This command is only available in command mode.</li> </ul>
<b>Arguments</b>	<p>x: Its definition is as follows:</p> <ul style="list-style-type: none"> <li>-O: Retrieve open mode settings of CAN port for comparing with the settings of @Onxy command.</li> <li>-U: Retrieve current baud rate of COM port for comparing with the setting of @Un command.</li> <li>-B: Retrieve current bit rate of CAN port for comparing with the setting of @Bn command.</li> <li>-S: Retrieve settings of CAN for comparing with the settings of @Sx command.</li> <li>-Fnn: Retrieve settings of acceptance filter (list mode) nn for comparing with the settings of @Fnnxiii... command.</li> <li>-Mnn: Retrieve settings of acceptance filter (mask mode) nn for comparing with the settings of @Mnnxiii... command.</li> <li>-Z: Retrieve setting of timestamp for comparing with the setting of @Zn command.</li> <li>-R: Retrieve one message out of receive buffer.</li> <li>-E: Retrieve the latest error code.</li> </ul>
<b>Returns</b>	<p>@GO: Refer to command description for more details, e.g.: O100.</p> <p>@GU: U1200, U2400, U4800, U9600, U19200, U38400, U57600, U115200, U230400, U460800 or U921600.</p> <p>@GB: Refer to command description for more details, e.g.: B0, B1...or B7.</p> <p>@GS: Refer to command description for more details, e.g.: S0, S1...or S7.</p> <p>@GFnn: Refer to command description for more details.</p> <p>@GMnn: Refer to command description for more details.</p> <p>@GZ: Refer to command description for more details, e.g.: Z0 or Z1.</p> <p>@GR: Please refer to section 5.4 Data Format for details.</p> <p>@GE: A prefix character of 'E' plus a 4-byte value indicate the latest error status.</p> <ul style="list-style-type: none"> <li>Bit[24:31] : REC - Receive Error Counter.</li> <li>Bit[16:23] : TEC - Transmit Error Counter.</li> <li>Bit[07:15] : Reserved.</li> <li>Bit[04:06] : LEC - Last Error Code. <ul style="list-style-type: none"> <li>000: No error.</li> <li>001: Stuff error.</li> <li>010: Form error.</li> <li>011: Acknowledgment error.</li> <li>100: Bit recessive error.</li> <li>101: Bit dominant error.</li> <li>110: CRC error.</li> <li>111: Reserved.</li> </ul> </li> <li>Bit[3] : Reserved.</li> <li>Bit[2] : Bus off flag.</li> <li>Bit[1] : Error passive flag (REC &gt; 127 or TEC &gt; 127).</li> <li>Bit[0] : Error warning flag (REC &gt;= 96 or TEC &gt;= 96).</li> </ul>
<b>Others</b>	None

## 5.3 Identifier Filtering

Each CAN port provides 14 configurable identifier filters for selecting the incoming message the software needs and discarding the others. Filters can be configured in mask mode or in identifier list mode.

### Mask mode

In mask mode the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”.

### Identifier list mode

In identifier list mode, the mask registers are used as identifier registers. Thus instead of defining an identifier and a mask, two identifiers are specified, doubling the number of single identifiers. All bits of the incoming identifier must match the bits specified in the filter registers.



Note

*Filter index 0x0 has been set as mask mode with identifier id 0 by default, which means all CAN messages will not be filtered out.*

## 5.4 Data Format

Raw Data:

Raw data in hexadecimal

Formatted Data:

Standard CAN 2.0A

Byte	1	1	4	1	3	1	1	Data Len*2	1
@	F	stamp	0	ID	RTR	Data Len	Data		0x0D

Extended CAN 2.0B

Byte	1	1	4	1	3	1	1	Data Len*2	1
@	F	stamp	1	ID	RTR	Data Len	Data		0x0D

## 5.5 Error Code

Error No.	Error Code	Description
0	OK	No error.
1	ERROR01	CAN has already been opened.
2	ERROR02	Error detected in opening CAN port.
3	ERROR03	CAN is not open.
4	ERROR04	CAN bit time is not set.
5	ERROR05	CAN Tx buffer is full.
6	ERROR06	CAN buffer is empty.
7	ERROR07	CAN device does not exist.
8	ERROR08	Command error.
9	ERROR09	Flash error.
10	ERROR10	Argument error.

## 5.6 Some Examples

In this section, we provides some examples of how to use command messages.

### Transmit Messages:

1. Command: @BE  
Do: Set CAN bit rate to 1Mbps.
2. Command: @S0  
Do: Configure receiving behaviors of CAN port; stop receiving when buffer is full, report in raw data and polling mode.
3. Command: @O100  
Do: Open CAN Port 1. Set it to normal mode and non-loopback.
4. Command: @t1000A006412383132333435363738  
Do: Transmit 100 messages every 1ms; ID=0x123, length=8 and data='12345678'.

### Receive Messages: (Auto Report)

1. Command: @BE  
Do: Set CAN bit rate to 1Mbps.
2. Command: @S3  
Do: Configure receiving behaviors of CAN port; stop receiving if buffer is overflow, report data format and auto report mode.
3. Command: @O100  
Do: Open CAN port 1. Set it to normal mode and non-loopback.
4. Command: @S3  
Do: Set to data mode and report data automatically if CAN port is open.

**Receive Messages:** (Polling)

1. Command: @BE  
Do: Set CAN bit rate to 1Mbps.
  
2. Commands: @S2  
Do: Configure receiving behaviors of CAN port to stop receiving if buffer is overflow, report data format and polling mode.
  
3. Command: @O100  
Do: Open CAN port 1. Set it to normal mode and non-loopback.
  
4. Command: @Z1  
Do: Enable timestamp.
  
5. Command: @GR  
Do: Retrieve message out of receive buffer.

# Chapter 6

## Programming Guide

We release a set of application programming interface (API) functions for users to access/control hardware. With these API functions, users can more easily design their own software. This chapter includes detailed description of each API function and step-by-step code samples showing how it works.

### 6.1 COM Port Device Functions

#### 1. AX\_OpenDevice()

This function opens COM port device of AX92903 with baud rate setting.

##### Definition

AX\_STATUS AX\_OpenDevice(UCHAR ComPort, ULONG BaudRate);

##### Parameters

*ComPort* COM port number.

*BaudRate* Baud rate values range from 1200 to 921600.

##### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

##### Example

```
AX_STATUS axStatus;
char comport = 1;          // comport device number = 1
ULONG baudrate = 115200;
axStatus = AX_OpenDevice(comport, baudrate);
if (axStatus == AX_OK)
{
    // AX_OpenDevice success
}

else
{
    // AX_OpenDevice failed
}
```

## 2. AX\_CloseDevice()

This function closes COM port device of AX92903.

### Definition

```
AX_STATUS AX_CloseDevice();
```

### Parameters

None.

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;  
axStatus = AX_CloseDevice();  
if (axStatus == AX_OK)  
{  
    // AX_CloseDevice success  
}  
  
else  
{  
    // AX_CloseDevice failed  
}
```

### 3. AX\_SetComBaudRate()

This function sets baud rate of COM port device.

#### Definition

AX\_STATUS AX\_SetComBaudRate(UCHAR ComPort, ULONG BaudRate);

#### Parameters

*ComPort* COM port number.

*BaudRate* Baud rate values range from 1200 to 921600.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;
char comport = 1;          // comport device number = 1
ULONG baudrate = 115200;
axStatus = AX_SetComBaudRate(comport, baudrate);
if (axStatus == AX_OK)
{
    // AX_SetComBaudRate success
}
else
{
    // AX_SetComBaudRate failed
}
```

#### 4. AX\_GetDeviceNum()

This function gets the number of devices currently connected.

##### Definition

```
AX_STATUS AX_GetDeviceNum(UCHAR *DevNum);
```

##### Parameters

*DevNum* Pointer to the number of devices.

##### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

##### Example

```
AX_STATUS axStatus;
char num;
axStatus = AX_GetDeviceNum(&num);
if (axStatus == AX_OK)
{
// AX_GetDeviceNum success
printf("number of devices currently connected = %d\r\n",num);
}
else
{
// AX_GetDeviceNum failed
}
```



## 6.2 CAN Device Functions

### 1. AX\_SetMcuBaudRate()

This function sets MCU's baud rate.

#### Definition

```
AX_STATUS AX_SetMcuBaudRate(ULONG BaudRate);
```

#### Parameters

*BaudRate* Baud rate value.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
char baudrate = 921600;  
axStatus = AX_SetMcuBaudRate(baudrate);  
if (axStatus == AX_OK)  
{  
    // AX_SetMcuBaudRate success  
}  
else  
{  
    // AX_SetMcuBaudRate failed  
}
```

## 2. AX\_SetCANBitRate()

This function sets CAN's bit rate of AX92903.

### Definition

```
AX_STATUS AX_SetCANBitRate(UCHAR bitrate);
```

### Parameters

*bitrate* CAN's bit rate. This parameter must be a value between 0 and 14.

0	5Kbit
1	10Kbit
2	20Kbit
3	40Kbit
4	50Kbit
5	80Kbit
6	100Kbit
7	125Kbit
8	200Kbit
9	250Kbit
10	400Kbit
11	500Kbit
12	640Kbit
13	800Kbit
14	1000Kbit

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;  
// Set CAN's bit-rate to 1Mbps  
axStatus = AX_SetCANBitRate(14);  
if (axStatus == AX_OK)  
{  
    // AX_SetCANBitRate success  
}  
else  
{  
    // AX_SetCANBitRate failed  
}
```

### 3. AX\_SetCANBitRateEx()

This function sets CAN's bit rate of AX92903 with TS1, TS2 and BRP.

#### Definition

```
AX_STATUS AX_SetCANBitRateEx(UCHAR TS1, UCHAR TS2, USHORT BRP);
```

#### Parameters

*TS1* Time segment 1.  
*TS2* Time segment 2.  
*BRP* Baud Rate Prescaler.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
// Set CAN's bit-rate to 250kbps  
axStatus = AX_SetCANBitRateEx(6,1,18);  
if (axStatus == AX_OK)  
{  
    // AX_SetCANBitRateEx success  
}  
  
else  
{  
    // AX_SetCANBitRateEx failed  
}
```

#### 4. AX\_SetCANRxBuffer()

This function sets behavior of CAN's receiving buffer with CAN receiver buffer override, report type and auto/polling report.

##### Definition

```
AX_STATUS AX_SetCANRxBuffer(CANRX_TYPE type);
```

##### Parameters

*type* Behavior of Rx buffer. This parameter must be:  
STOP\_RAW\_POLLING  
STOP\_RAW\_AUTO  
STOP\_FORMAT\_POLLING  
STOP\_FORMAT\_AUTO  
OVERRIDE\_RAW\_POLLING  
OVERRIDE\_RAW\_AUTO  
OVERRIDE\_FORMAT\_POLLING  
OVERRIDE\_FORMAT\_AUTO.

##### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

##### Example

```
AX_STATUS axStatus;  
char num;  
// setting with stop received if buffer overflow, report raw  
// data and polling receiver buffer data.  
axStatus = AX_SetCANRxBuffer(STOP_RAW_POLLING);  
if (axStatus == AX_OK)  
{  
// AX_SetCANRxBuffer success  
}  
else  
{  
// AX_SetCANRxBuffer failed  
}
```

### 5. AX\_SetCANTimeStamp()

This function sets whether report data has time stamp or not.

#### Definition

```
AX_STATUS AX_SetCANTimeStamp(bool onoff);
```

#### Parameters

*onoff* Report time stamp. Must be TRUE or FALSE.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
char num;  
axStatus = AX_SetCANTimeStamp(TRUE);  
if (axStatus == AX_OK)  
{  
    // AX_SetCANTimeStamp success  
}  
else  
{  
    // AX_SetCANTimeStamp failed  
}
```

## 6. AX\_OpenCAN()

This function opens CAN port and sets operating mode.

### Definition

```
AX_STATUS AX_OpenCAN(UCHAR DevNum, CAN_MODE mode);
```

### Parameters

*DevNum* CAN port number.  
*Mode* Operating mode:  
NORMAL\_NONLOOPBACK  
NORMAL\_LOOPBACK  
SILENT\_NONLOOPBACK  
SILENT\_LOOPBACK

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;  
char num;  
// Open first CAN port and operating at normal mode.  
axStatus = AX_OpenCAN(0, NORMAL_NONLOOPBACK);  
if (axStatus == AX_OK)  
{  
// AX_OpenCAN success  
}  
else  
{  
// AX_OpenCAN failed  
}
```

## 7. AX\_CloseCAN()

This function closes CAN port.

### Definition

```
AX_STATUS AX_CloseCAN(UCHAR DevNum);
```

### Parameters

*DevNum* CAN port number.

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;
char num;
// Close first CAN port.
axStatus = AX_CloseCAN(0);
if (axStatus == AX_OK)
{
// AX_CloseCAN success
}
else
{
// AX_CloseCAN failed
}
```

## 8. AX\_TxStdCAN()

This function transmits standard identifier of CAN data.

### Definition

```
AX_STATUS AX_TxStdCAN(UCHAR DevNum, USHORT Period, USHORT Repeat, USHORT ID, UCHAR Len, UCHAR *TxBuff);
```

### Parameters

*DevNum* CAN port number.  
*Period* Transmit period (1/bit rate second).  
*Repeat* Number of CAN's data.  
*ID* Identification of CAN's data.  
*Len* Length of CAN's data.  
*TxBuff* Content of CAN's data.

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;  
char data[3] = {0x31, 0x32, 0x33};  
// Transmit 10 times CAN's data with ID = 0x123, every 100 times bit-time,  
// data length = 3 and data = 0x31,0x32,0x33 from first CAN port.  
axStatus = AX_TxStdCAN(0, 100, 10, 0x123, 3, data)  
if (axStatus == AX_OK)  
{  
// AX_TxStdCAN success  
}  
else  
{  
// AX_TxStdCAN failed  
}
```



### 9. AX\_TxExtCAN()

This function transmits extend identifier of CAN data.

#### Definition

```
AX_STATUS AX_TxExtCAN(UCHAR DevNum, USHORT Period, USHORT Repeat, USLONG ID, UCHAR Len, UCHAR *TxBuff);
```

#### Parameters

*DevNum* CAN port number.  
*Period* Transmit period (1/bit rate second).  
*Repeat* Number of CAN's data.  
*ID* Identification of CAN's data.  
*Len* Length of CAN's data.  
*TxBuff* Content of CAN's data.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
char data[3] = {0x31, 0x32, 0x33};  
// Transmit 10 times CAN's data with ID = 0x12345678, every 100 times  
// bit-time, data length = 3 and data = 0x31,0x32,0x33  
// from first CAN port.  
axStatus = AX_TxExtCAN (0, 100, 10, 0x12345678, 3, data);  
if (axStatus == AX_OK)  
{  
    // AX_TxExtCAN success  
}  
else  
{  
    // AX_TxExtCAN failed  
}
```

## 10. AX\_SetCANFilter()

This function sets the acceptance identification.

### Definition

AX\_STATUS AX\_SetCANFilter(UCHAR *Active*, UCHAR *Num*, ULONG *Low*, ULONG *Hi*);

### Parameters

*Active* Active filter.  
*Num* Number of filter.  
*Low* Lower acceptance identification.  
*Hi* Upper acceptance identification.

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;  
char pRxBuff[128];  
// Set first filter with acceptance identification range  
// between 0x123 and 0x129.  
axStatus = AX_SetCANFilter(1, 0, 0x123, 0x129);  
if (axStatus == AX_OK)  
{  
    // AX_SetCANFilter success  
}  
else  
{  
    // AX_SetCANFilter failed  
}
```

**11. AX\_GetCANFilter()**

This function gets the acceptance identification.

**Definition**

```
AX_STATUS AX_GetCANFilter(UCHAR Num, UCHAR *Active, ULONG *Low, ULONG *Hi);
```

**Parameters**

*Active* Active filter.  
*Num* Number of filter.  
*Low* Lower acceptance identification.  
*Hi* Upper acceptance identification.

**Return Value**

AX\_OK if successful, otherwise the return value is an AX error code.

**Example**

```
AX_STATUS axStatus;  
ULONG Low,Hi;  
// Get first acceptance identification filter.  
axStatus = AX_GetCANFilter(0, &Low, &Hi);  
if (axStatus == AX_OK)  
{  
// AX_GetCANFilter success  
printf("First acceptance identification filter range :  
0x%X~0x%X",Low,Hi);  
}  
else  
{  
// AX_GetCANFilter failed  
}
```

## 12. AX\_SetCANSysInfo()

This function sets system information.

### Definition

```
AX_STATUS AX_SetCANSysInfo(UCHAR InfoNum, UCHAR *strInfo);
```

### Parameters

*InfoNum* Number of system information. This parameter must be:

- 0 Model name.
- 1 Serial number.
- 2 UUID.
- 3 Manufacturer.
- 4 Firmware version.
- 5 Product data.
- 6 Firmware version.

*strInfo* Specific system information string.

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;
UCHAR ModelName[64] = "AX92903";
// Set model name string.
axStatus = AX_SetCANSysInfo(0, ModelName);
if (axStatus == AX_OK)
{
// AX_SetCANSysInfo success
}
else
{
// AX_SetCANSysInfo failed
}
```

### 13. AX\_GetCANSysInfo()

This function gets system information.

#### Definition

```
AX_STATUS AX_GetCANSysInfo(UCHAR InfoNum, UCHAR *strInfo);
```

#### Parameters

*InfoNum* Number of system information. This parameter must be:

0	Model name.
1	Serial number.
2	UUID.
3	Manufacturer.
4	Firmware version.
5	Product data.
6	Firmware version.

*strInfo* Specific system information string.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
DWORD RxLen;  
UCHAR ModelName[64];  
// Get model name string.  
axStatus = AX_GetCANSysInfo(0, ModelName);  
if (axStatus == AX_OK)  
{  
    // AX_GetCANSysInfo success  
    printf("Model Name = %s\r\n", ModelName);  
}  
else  
{  
    // AX_GetCANSysInfo failed  
}
```

#### 14. AX\_SetCANTermInfo()

This function sets the terminal of system.

##### Definition

```
AX_STATUS AX_SetCANTermInfo(UCHAR DevNum, UCHAR onoff);
```

##### Parameters

*DevNum* CAN port number.

*onoff* Set terminal state. Must be 0 or 1.

##### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

##### Example

```
AX_STATUS axStatus;
// Turn on terminal.
axStatus = AX_SetCANTermInfo(1, 1);
if (axStatus == AX_OK)
{
// AX_SetCANTermInfo success
}
else
{
// AX_SetCANTermInfo failed
}
```

### 15. AX\_GetCANTermInfo()

This function gets the terminal information of system.

#### Definition

```
AX_STATUS AX_GetCANTermInfo(UCHAR DevNum, UCHAR *strInfo);
```

#### Parameters

*DevNum* CAN port number.  
*strInfo* Terminal information string.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
DWORD RxLen;  
UCHAR Buffer[64];  
// Get terminal state.  
axStatus = AX_GetCANTermInfo(1, Buffer);  
if (axStatus == AX_OK)  
{  
// AX_GetCANTermInfo success  
}  
else  
{  
// AX_GetCANTermInfo failed  
}
```

## 16. AX\_GetCANConfig()

This function gets the configuration of CAN.

### Definition

```
AX_STATUS AX_GetCANConfig(UCHAR cmd, UCHAR *Buff);
```

### Parameters

<i>cmd</i>	Type of CAN's setting.
'U'	Get MCU's baud rate.
'B'	Get CAN's bit rate.
'S'	Get the behavior of receiver buffer.
'Z'	Get time stamp setting.
'R'	Request receiver data once.
'E'	Get error status.
<i>Buff</i>	Specific CAN's configuration string.

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;
UCHAR Response[64];
// Get MCU's baud rate
axStatus = AX_GetCANConfig('U', Response);
if (axStatus == AX_OK)
{
// AX_GetCANConfig success
printf("MCU's Baud Rate setting = %s\r\n", Response);
}
else
{
// AX_GetCANConfig failed
}
```



### 17. AX\_SaveSysConfig()

This function saves the configuration of system to NVRAM.

#### Definition

```
AX_STATUS AX_SaveSysConfig();
```

#### Parameters

None.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
// Save the configuration of system to NVRAM  
axStatus = AX_SaveSysConfig();  
if (axStatus == AX_OK)  
{  
    // AX_SaveSysConfig success  
}  
else  
{  
    // AX_SaveSysConfig failed  
}
```

### 18. AX\_ExitCANDataMode()

This function returns AX92903 from data mode to command mode.

#### Definition

```
AX_STATUS AX_ExitCANDataMode();
```

#### Parameters

None.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
// Set AX92903 to the command mode  
axStatus = AX_ExitCANDataMode();  
if (axStatus == AX_OK)  
{  
    // AX_ExitCANDataMode success  
}  
else  
{  
    // AX_ExitCANDataMode Error  
}
```

**19. AX\_GetCANRxBuff()**

This function gets the CAN frames received from AX92903 (in formatted data).

**Definition**

```
AX_STATUS AX_GetCANRxBuff(AX_RX_BUFF *CAN_DATA);
```

**Parameters**

**CAN\_DATA**                    Pointer to AX\_RX\_BUFF structure.

**Return Value**

AX\_OK if successful, otherwise the return value is an AX error code.

**Example**

```
AX_STATUS axStatus;
AX_RX_BUFF can_data;
// Get CAN frames received from AX92903
axStatus = AX_GetCANRxBuff(&can_data);
if (axStatus == AX_OK)
{
printf("TimeStamp = ");
for (int i=0;i<4;i++) printf("%C", can_data.TimeStamp[i]);
printf("ID = ");
for (int i=0;i<8;i++) printf("%C", can_data.ID[i]);
printf("IDE = %C", can_data.IDE);
printf("RTR = %C", can_data.RTR);
printf("DATA Len = %C", can_data.LEN);
printf("DATA = ");
for (int i=0;i<16;i++) printf("%C", can_data.DATA[i]);
}
else
{
// AX_GetCANRxBuff Error
}
```

## 20. AX\_GetCANRawBuff()

This function gets the CAN frames received from AX92903 (in raw data).

### Definition

```
DWORD AX_GetCANRawBuff(UCHAR *Buff, DWORD Len);
```

### Parameters

*Buff* Pointer to buffer for CAN frame.  
*Len* Length of CAN frame.

### Return Value

Return data length if successful, otherwise the return value is 0.

### Example

```
AX_STATUS axStatus;  
UCHAR RawBuff[64];  
// Get raw data of CAN frame received from AX92903  
axStatus = AX_GetCANRawBuff(RawBuff);  
if (axStatus != AX_OK)  
{  
// AX_GetCANRawBuff Error  
}
```

## 6.3 NVRAM Functions

### 1. AX\_WriteNVRam()

This function writes data to user defined NVRAM (max. 2KBytes).

#### Definition

AX\_STATUS AX\_WriteNVRam(UCHAR Page, USHORT Offset, USHORT Len, UCHAR \*WriteBuff);

#### Parameters

*Page* Page number. Must be 0x0~0x1F.  
*Offset* Offset number. Must be 0x0~0x7FF.  
*Len* Length of data. Must be 0x0~0x800.  
*WriteBuff* Pointer to data string.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;
UCHAR writeBuff [11] = "UserDefine";
// write 11 bytes data to user defined NVRAM
axStatus = AX_WriteNVRam(0, 0, 11, writeBuff);
if (axStatus == AX_OK)
{
// AX_WriteNVRam success
}
else
{
// AX_WriteNVRam failed
}
```

## 2. AX\_ReadNVRam()

This function request data from user defined NVRAM (max. 2KBytes).

### Definition

AX\_STATUS AX\_ReadNVRam(UCHAR Page, USHORT Offset, USHORT Len, UCHAR \*ReadBuff);

### Parameters

*Page* Page number. Must be 0x0~0x1F.  
*Offset* Offset number. Must be 0x0~0x7FF.  
*Len* Length of data. Must be 0x0~0x800.  
*ReadBuff* Pointer to data string.

### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

### Example

```
AX_STATUS axStatus;
UCHAR ReadBuff[64];
// Request data from the user defined NVRAM
axStatus = AX_ReadNVRam(0, 0, 11, ReadBuff);
if (axStatus == AX_OK)
{
// AX_ReadNVRam success
}
else
{
// AX_ReadNVRam failed
}
```

### 3. AX\_EraseNVRam()

This function erases data of user defined NVRAM (max. 2KBytes).

#### Definition

AX\_STATUS AX\_EraseNVRam(UCHAR *Page*, USHORT *Offset*, USHORT *Len*);

#### Parameters

*Page* Page number. Must be 0x0~0x1F.  
*Offset* Offset number. Must be 0x0~0x7FF.  
*Len* Length of data. Must be 0x0~0x800.

#### Return Value

AX\_OK if successful, otherwise the return value is an AX error code.

#### Example

```
AX_STATUS axStatus;  
// Erase data of the user defined NVRAM  
axStatus = AX_EraseNVRam(0, 0, 11);  
if (axStatus == AX_OK)  
{  
    // AX_EraseNVRam success  
}  
else  
{  
    // AX_EraseNVRam failed  
}
```

## 6.4 Type Definitions

### Data Type

UCHAR unsigned char (1 byte)  
 USHORT unsigned short (2 bytes)  
 ULONG unsigned long (4 bytes)  
 DWORD unsigned long (4 bytes)  
 PCHAR pointer to unsigned char (1 byte)  
 LPWORD pointer to unsigned long (4 bytes)  
 LPLONG pointer to unsigned long (4 bytes)

CANRX\_TYPE (refer to AX\_SetCANRxBuffer())

STOP\_RAW\_POLLING = 0  
 STOP\_RAW\_AUTO = 1  
 STOP\_FORMAT\_POLLING = 2  
 STOP\_FORMAT\_AUTO = 3  
 OVERRIDE\_RAW\_POLLING = 4  
 OVERRIDE\_RAW\_AUTO = 5  
 OVERRIDE\_FORMAT\_POLLING = 6  
 OVERRIDE\_FORMAT\_AUTO = 7

CAN\_MODE (refer to AX\_OpenCAN())

NORMAL\_NONLOOPBACK = 0  
 NORMAL\_LOOPBACK = 1  
 SILENT\_NONLOOPBACK = 2  
 SILENT\_LOOPBACK = 3

AX\_STATUS (DWORD)

AX\_OK = 0  
 AX\_INVALID\_HANDLE = 1  
 AX\_DEVICE\_NOT\_FOUND = 2  
 AX\_DEVICE\_NOT\_OPENED = 3  
 AX\_FAILED\_TO\_WRITE\_DEVICE = 4  
 AX\_INVALID\_ARGS = 5  
 AX\_NOT\_SUPPORTED = 6  
 AX\_OTHER\_ERROR = 7  
 AX\_DEVICEOPENED = 8  
 AX\_DEVICEOPEN = 9  
 AX\_DEVICENOTOPEN = 10  
 AX\_DEVICEBTRNOTSET = 11  
 AX\_BUFFEROVERFLOW = 12  
 AX\_BUFFEREMPTY = 13  
 AX\_DEVICENOTEXIST = 14  
 AX\_CMDFAILED = 15  
 AX\_FLASHERROR = 16

AX\_RX\_BUFF

```

typedef struct AX_RX_BUFF
{
    UCHAR ReportType;
    UCHAR TimeStamp[4];
    UCHAR IDE;
    UCHAR ID[8];
    UCHAR RTR;
    UCHAR LEN;
    UCHAR DATA[16];
};
  
```