USER MANUAL

USB-CAN CONVERTER SDK

P/N: AX070501SDK

In Europe:
Axiomatic Technologies Oy
Höytämöntie 6
33880 Lempäälä - Finland
Tel. +358 103 375 750
Fax. +358 3 3595 660
www.axiomatic.fi

In North America:
Axiomatic Technologies Corporation
5915 Wallace Street
Mississauga, ON Canada L4Z 1Z8
Tel. 1 905 602 9270
Fax. 1 905 602 9279
www.axiomatic.com

ACRONYMS

AX	Axiomatic
CAN	Controller Area Network
DLL	Dynamic-Link Library
P/N	Part Number
SDK	Software Development Kit
USB	Universal Serial Bus

TABLE OF CONTENTS

1 INTRODUCTION 4
2 SDK DESCRIPTION 5
3 USB-CAN CONVERTER SOFTWARE STACK 6
4 DEVELOPMENT MODULES 7
5 EXAMPLE PROJECTS 8
6 INTERFACE FUNCTIONS 9
6.1 Information and Handle Functions 9
6.2 Baud Rate Control Functions 10
6.3 Data Transfer Functions 13
6.3.1 Notification Frames 15
6.3.1.1 Runtime Errors 15
6.3.1.2 Notification Messages 16
6.4 Version Information Functions 16
7 VERSION HISTORY 19

1 INTRODUCTION

The following user manual describes USB-CAN Converter Software Development Kit (SDK).

The USB-CAN Converter SDK version number contains 3 numerical fields separated by dots and one optional letter field:

`fc.fu.fm<fo>`

Where:
- `fc` – compatibility level, SDK code from different compatibility levels can be incompatible;
- `fu` – backward compatible updates for the given compatibility level;
- `fm` – minor changes not related to the SDK functionality, mainly bug-fixes;
- `fo` – optional letter for updates not related to the SDK code. For example, updates to the user manual or changes in the examples, etc.

The user manual is valid for SDKs with the same two major version numbers as the user manual (fields: `fc` and `fu`). For example, this user manual is valid for any SDK version 2.0.x[1]. Updates specific to the user manual are done by adding letters: A, B, …, Z to the user manual version number.

[1]Before version 2.0.0, the SDK used two numerical fields separated by a dot for the version number. Each version of the SDK had its own user manual with the same version number.

2 SDK DESCRIPTION

USB-CAN Converter SDK is designed to allow independent software developers and system integrators to use Axiomatic USB-CAN converter, p/n AX070501 or AX070505, in their own applications. It includes description of USB-CAN converter software stack and software development modules necessary to create user applications communicating with Axiomatic USB-CAN converters.

USB-CAN Converter SDK version 2.0.x is intended to work with the following USB-CAN software components:

Table 1. Compatible Software Components

Software Component Name	Version
USB-CAN Converter Firmware	1.07 or upper[1]
USB-CAN Device Driver (`usbcan.sys`)	1.1.x.x or upper
USB-CAN Dynamic-Link Library (`USBCANDll.dll`)	1.0.0.1 or upper

[1]Versions 1.xx are used in AX070501, and 2.xx (and upper) are used in AX070505.

The SDK currently supports Microsoft Windows operating system starting from Win2000 and uses Microsoft Visual Studio C++ for software development. It can be modified and used with other programming languages and software development tools supporting function calls from dynamic-link (DLL) libraries. Support for other operating systems is not included.

Each SDK version is installed in its own directory. The SDK version 2.0.x (where x – part of the version number for minor changes) generates the following folder structure in the SDK root folder on the user's computer, see Figure 1:

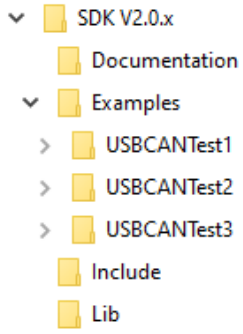

Figure 1. SDK Folder Structure

By default, the SDK root folder is located at:

`%windir%\Program Files\Axiomatic Technologies Corporation\USB-CAN Converter\`

This location can be changed during installation if necessary.

The SDK subfolders contain the following information, see Table 2:

Table 2. SDK directory folder contents

Subfolder Name	Description
`\Documentation`	License Agreement and this User Manual
`\Include`	`USBCAN.h` file
`\Lib`	`USBCANDll.dll` and `USBCANDll.lib` files
`\Examples`	`USBCANTest1, USBCANTest2, USBCANTest3` example projects, each in an individual folder

3 USB-CAN CONVERTER SOFTWARE STACK

The following software stack is used to control USB-CAN converters from Microsoft Windows applications, see Figure 2:

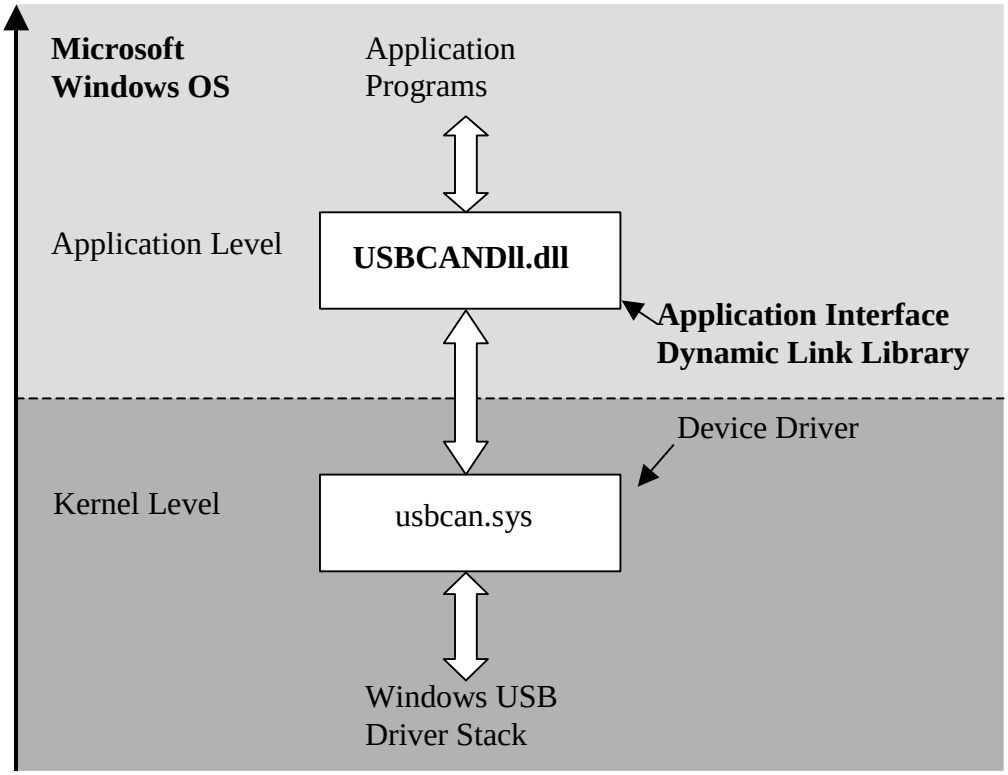

Figure 2. USB-CAN Software Stack

The kernel portion of the stack, `usbcan.sys` driver module, is installed by Windows Hardware Wizard during the converter driver installation.

The application part, `USBCANDll.dll`, should be installed by the application software. The USB-CAN Converter SDK automatically installs `USBCANDll.dll` in the Windows system folder for use with all application software.

The order of installation for the kernel and application parts is not important.

4 DEVELOPMENT MODULES

USB-CAN Converter SDK includes the following software development modules, see Table 3.

Table 3. SDK Software Development Modules

Module Name	Description
`USBCAN.h`	C Include file. Contains: data structures, constants and prototypes of all interface functions used by application software to communicate with USB-CAN converters through `USBCANDll.dll` library. The user should define `_USB_CAN_SDK_V1_XX` macro before including this file if compatibility with older SDK versions is required. `#define _USB_CAN_SDK_V1_XX`
`USBCANDll.lib`	Library file. Contains stubs for all interface functions described in `USBCAN.h`.

5 EXAMPLE PROJECTS

The SDK also contains example projects written in Microsoft Visual C++ describing usage of the interface functions. They are simple Win32 console projects.

Table 4. SDK Example Projects

Example Name	Description
`USBCANTest1`	The program receives and displays messages from a USB-CAN converter.
`USBCANTest2`	The program sends 100 Extended CAN frames to a USB-CAN converter.
`USBCANTest3`	The program retrieves: current USB-CAN converter baud rate, USB-CAN converter firmware version number, Windows device driver version number, and DLL module version number.

All example projects perform enumeration of the installed USB-CAN converters and acquire a handle to the first active converter in the system.

6 INTERFACE FUNCTIONS

6.1 Information and Handle Functions

Name:	**USBCANGetDeviceInfo()**	
Description:	The function retrieves device information records including device paths and device flags for all USB to CAN converters installed in the system.	

Prototype:	`DWORD USBCANGetDeviceInfo(`
	`  PUSBCAN_DEVICE_INFO pDeviceInfoArray,`
	`  size_t DeviceInfoArraySize`
	`);`	

Parameters:	`PUSBCAN_DEVICE_INFO pDeviceInfoArray`	Pointer to an array of `USBCAN_DEVICE_INFO` structures receiving device information records.
	`size_t DeviceInfoArraySize`	Size of the array pointed by `pDeviceInfoArray` in bytes.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	
Comments:	The function populates array pointed by `pDeviceInfoArray` with `USBCAN_DEVICE_INFO` records for USB-CAN converters installed in the system. Each `USBCAN_DEVICE_INFO` record contains a zero terminated device path and device flags. The last device record is zero populated. User must submit an array large enough to receive all device records and the final zero populated record, otherwise the function will only return a part of the device records without the final zero populated record.	

The `USBCAN_DEVICE_INFO` structure is defined in `USBCAN.h` the following way:

```
typedef struct _USBCAN_DEVICE_INFO
{
    DWORD dwFlags;
    TCHAR szDevicePath[MAX_PATH+1];
}USBCAN_DEVICE_INFO,*PUSBCAN_DEVICE_INFO;
```

Name:	**USBCANGetDeviceHandle()**	
Description:	The function retrieves a handle to the USB-CAN converter defined by the device path.	

Prototype:	`DWORD USBCANGetDeviceHandle(`
	`  LPCTSTR cszDevicePath,`
	`  ULONG ulMode,`
	`  HANDLE *pDeviceHandle`
	`);`	

Parameters:	`LPCTSTR cszDevicePath`	Pointer to a zero terminated device path to the USB-CAN converter.
	`ULONG ulMode`	Device mode associated with the handle. Can be either: `USBCAN_BLOCKING_MODE` or `USBCAN_OVERLAPPED_MODE`.
	`HANDLE *pDeviceHandle`	Pointer to a variable, which will receive a device handle.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	

Comments:	If the function succeeds, a variable pointed by `pDeviceHandle` receives a valid device handle; otherwise, it is set to `INVALID_HANDLE_VALUE`. User should submit a pointer to a null terminated device path received from a call to `USBCANGetDeviceInfo()`. Set the device mode value `ulMode` to `USBCAN_BLOCKING_MODE` for synchronous (blocking) operations with the handle. Otherwise, for asynchronous (overlapped) operations, set `ulMode` to `USBCAN_OVERLAPPED_MODE`. The user should close all device handles opened by this function when they are no longer needed by calling `USBCANCloseDeviceHandle()`. Windows will automatically close all device handles requested by a program when the program is terminated.

Name:	**USBCANCloseDeviceHandle()**	
Description:	The function closes a handle to the USB-CAN converter.	

Prototype:	`DWORD USBCANCloseDeviceHandle(`
	`  HANDLE hDeviceHandle`
	`);`	

Parameters:	`HANDLE hDeviceHandle`	Valid device handle to the USB-CAN converter.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	
Comments:	The user should call this function to close a device handle previously open by `USBCANGetDeviceHandle()` when the handle to the USB-CAN converter is no longer needed. Windows will automatically close all device handles requested by a program when the program is terminated.	

6.2 Baud Rate Control Functions

Name:	**USBCANSetBaudRate()**	
Description:	The function sets baud rate for the USB-CAN converter.	

Prototype:	`DWORD USBCANSetBaudRate(`
	`  HANDLE  hDeviceHandle,`
	`  int  iBaudRate,`
	`  LPOVERLAPPED pOverlapped);`	

Parameters:	`HANDLE hDeviceHandle`	Valid device handle to the USB-CAN converter.
	`iBaudRate`	Required baud rate in kbit/s.
	`LPOVERLAPPED pOverlapped`	Pointer to an `OVERLAPPED` Windows structure defined in `Winbase.h` for asynchronous operations with the device handle or `NULL` otherwise.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	

Comments:	iBaudRate value should be picked from the following list of supported baud rates: **iBaudRate={1000,800,667[1],500,250,125,100,50,20,10}** [1]Baud rate 666.666(6) kbit/s, defined by integer value 667, is available starting from USB-CAN firmware version 2.00. It is used in AX070505 converters. This function also performs reset of the CAN peripheral controller and clears up its internal buffers and CAN input/output queues. To avoid data loss, it is recommended that the user first check baud rate with the function `USBCANGetBaudRate()` and, if the baud rate has not been set or if it is different from the required, call `USBCANSetBaudRate()`.

Name:	**USBCANGetBaudRate()**	
Description:	The function gets baud rate of the USB-CAN converter.	

Prototype:	`DWORD USBCANGetBaudRate(`
	`  HANDLE hDeviceHandle,`
	`  int *pBaudRate,`
	`  LPOVERLAPPED pOverlapped`
	`);`	

Parameters:	`HANDLE hDeviceHandle`	Valid device handle to the USB-CAN converter.
	`int *pBaudRate`	Pointer to a variable receiving the baud rate value.
	`LPOVERLAPPED pOverlapped`	Pointer to an `OVERLAPPED` Windows structure defined in `Winbase.h` for asynchronous operations with the device handle or `NULL` otherwise.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	
Comments:	The function returns the baud rate value set by the previous successful call to `USBCANSetBaudRate()` or `CUSTOM_BAUD_RATE` if the baud rate was set by `USBCANSetBaudRateSpecial()`. The converter baud rate is not defined upon power-up. The function will return zero if no baud rate has been set.	

Name:	**USBCANSetBaudRateSpecial()**	
Description:	The function sets any custom baud rate by directly programming CAN timing registers.	

Prototype:	`DWORD USBCANSetBaudRateSpecial(`
	`  HANDLE hDeviceHandle,`
	`  const PCAN_BITRATE_SET pBaudRateSet,`
	`  LPOVERLAPPED pOverlapped`
	`);`	

Parameters:	`HANDLE hDeviceHandle`	Valid device handle to the USB-CAN converter.
	`const PCAN_BITRATE_SET pBaudRateSet`	Pointer to the `CAN_BITRATE_SET` structure containing CAN timing register values.
	`LPOVERLAPPED pOverlapped`	Pointer to an `OVERLAPPED` Windows structure defined in `Winbase.h` for asynchronous operations with the device handle or `NULL` otherwise.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	

Comments:	The function allows the user to set values to CAN peripheral controller registers responsible for CAN bus timing. It is used for setting baud rates not supported by the `USBCANSetBaudRate()` function. The following registers can be set, see Table 5. *Table 5. CAN Timing Registers* See table below [1] Set in CAN time quanta For example, the baud rate 666.66(6) kbit/s, not supported by USB-CAN application firmware version 1.xx, can be set using the following set of register values: `{bTsec1=14; bTsec2=3; bSjw=1; wBrp=2}` Please refer to the Bosch CAN Specification Version 2.0 for more information about configuring CAN timing registers. The `wBaudRate` member variable of the `CAN_BITRATE_SET` structure pointed by `pBaudRateSet` is not used by this function. Consequent calls to `USBCANGetBaudRateSpecial()` after calling `USBCANSetBaudRateSpecial()` will return `CUSTOM_BAUD_RATE` in the `wBaudRate` member variable. Same as `USBCANSetBaudRate()`, this function performs reset of the CAN peripheral controller, clears its internal buffers and CAN input/output queues. Some application data can be lost during this operation.

Name	Range[1]	Description
`bTsec1`	2…16	Time segment before the sample point minus Synchronization Segment. The Synchronization Segment is always equal to 1.
`bTsec2`	1…8	Time segment after the sample point.
`bSjw`	1…4	Synchronization Jump Width.
`wBrp`	1…1024	Baud Rate Pre-Scaler. The value by which the oscillator frequency, equal to 24MHz, is divided for generating the CAN time quantum.

Name:	**USBCANGetBaudRateSpecial()**	
Description:	The function gets values of the CAN peripheral controller timing registers.	

Prototype:	`DWORD USBCANGetBaudRateSpecial(`
	`  HANDLE hDeviceHandle,`
	`  PCAN_BITRATE_SET pBaudRateSet,`
	`  LPOVERLAPPED pOverlapped)`	

Parameters:	`HANDLE hDeviceHandle`	Valid device handle to the USB-CAN converter.
	`PCAN_BITRATE_SET pBaudRateSet`	Pointer to a `CAN_BITRATE_SET` structure that will receive values of the CAN peripheral controller timing registers.
	`LPOVERLAPPED pOverlapped`	Pointer to an `OVERLAPPED` Windows structure defined in `Winbase.h` for asynchronous operations with the device handle or `NULL` otherwise.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	
Comments:	This function complements `USBCANSetBaudRateSpecial()` and is used to get values of the internal CAN peripheral controller timing registers set by	

	USBCANSetBaudRateSpecial() or internally assigned by the USB-CAN converter firmware in response to USBCANSetBaudRate() function call. The wBaudRate member variable of the CAN_BITRATE_SET structure will return a baud rate set by the previous successful call to USBCANSetBaudRate() or CUSTOM_BAUD_RATE if the baud rate was set by USBCANSetBaudRateSpecial(). The CAN peripheral controller timing registers are not defined upon power-up. The function will return a zero populated CAN_BITRATE_SET structure if the converter baud rate has not been set.

The CAN_BITRATE_SET structure and CUSTOM_BAUD_RATE constant for USBCANSetBaudRateSpecial() and USBCANGetBaudRateSpecial() functions are defined in USBCAN.h the following way:

```
typedef struct _CAN_BITRATE_SET
{
    WORD wBaudRate;
    BYTE bTseg1;
    BYTE bTseg2;
    BYTE bSjw;
    WORD wBrp;
} CAN_BITRATE_SET,*PCAN_BITRATE_SET;

#define CUSTOM_BAUD_RATE 0xffff
```

6.3 Data Transfer Functions

Name:	**USBCANSendCANFrame()**	
Description:	The function sends a CAN frame.	
Prototype:	DWORD USBCANSendCANFrame(HANDLE hDeviceHandle, const PCAN_FRAME pCANFrame, LPOVERLAPPED pOverlapped);	
Parameters:	HANDLE hDeviceHandle	Valid device handle to the USB-CAN converter.
	const PCAN_FRAME pCANFrame	Pointer to the CAN frame to be sent.
	LPOVERLAPPED pOverlapped	Pointer to an OVERLAPPED Windows structure defined in Winbase.h for asynchronous operations with the device handle or NULL otherwise.
Return value:	System error code. The function returns ERROR_SUCCESS on successful completion or an error code defined in WinError.h. For more information see GetLastError() function from Windows platform SDK.	
Comments:	The CAN frame is first placed in an internal USB-CAN converter buffer and then transmitted on the CAN bus. The converter receives its own transmitted CAN frames. If the synchronous (blocking) mode handle is used, the function will not return until the CAN frame is transmitted to the USB-CAN converter.	

Name:	**USBCANSendMessage()**
Description:	The function sends a USB-CAN message.

Prototype:	`DWORD USBCANSendMessage(`
	`  HANDLE hDeviceHandle,`
	`  const PUSBCAN_MESSAGE pUSBCANMessage,`
	`  LPOVERLAPPED pOverlapped`
	`);`	

Parameters:	`HANDLE hDeviceHandle`	Valid device handle to the USB-CAN converter.
	`const PUSBCAN_MESSAGE pUSBCANMessage`	Pointer to the CAN frame to be sent.
	`LPOVERLAPPED pOverlapped`	Pointer to an `OVERLAPPED` Windows structure defined in `Winbase.h` for asynchronous operations with the device handle or `NULL` otherwise.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	
Comments:	This function is a generic form of `USBCANSendCANFrame()` function that can send both: CAN and Notification frames. There are no Notification frames defined in this version of SDK that are allowed to be sent to the converter. If the synchronous (blocking) mode handle is used, the function will not return until the CAN frame is transmitted to the USB-CAN converter.	

Name:	**USBCANReceiveMessage()**	
Description:	The function receives a USB-CAN message.	

Prototype:	`DWORD USBCANReceiveMessage(`
	`  HANDLE hDeviceHandle,`
	`  PUSBCAN_MESSAGE pUSBCANMessage,`
	`  LPOVERLAPPED pOverlapped`
	`);`	

Parameters:	`HANDLE hDeviceHandle`	Valid device handle to the USB-CAN converter.
	`PUSBCAN_MESSAGE pUSBCANMessage`	Pointer to a `USBCAN_MESSAGE` structure that will receive the USB-CAN message.
	`LPOVERLAPPED pOverlapped`	Pointer to an `OVERLAPPED` Windows structure defined in `Winbase.h` for asynchronous operations with the device handle or `NULL` otherwise.
Return value:	System error code. The function returns `ERROR_SUCCESS` on successful completion or an error code defined in `WinError.h`. For more information see `GetLastError()` function from Windows platform SDK.	
Comments:	The message can be either a CAN or Notification frame. Notification frames, described in this SDK, are used to report errors or special events in the USB-CAN converter or the device driver. All other Notification frames should be ignored. If the synchronous (blocking) mode handle is used, the function will not return until the new message is sent by the USB-CAN converter.	

The `USBCAN_MESSAGE` and `CAN_FRAME` structures are defined in `USBCAN.h` the following way:

```
typedef enum _USBCAN_MESSAGE_TYPE
{
    VOID_MESSAGE          = 0,
    CAN_FRAME_MESSAGE     = 1,
    NOTIFICATION_FRAME_MESSAGE = 2
} USBCAN_MESSAGE_TYPE;
```

```
typedef struct _CAN_FRAME
{
    ULONG ulDd;             //Standard or Extended CAN Identifier

    BOOLEAN bEIdFlag;       // Extended CAN Identifier Flag
    BOOLEAN bRemFrameFlag;  // Remote Data Frame Flag
    BYTE bLength;           // Data Length
    BYTE Data[8];           // Data Byte Array

    ULONG ulTimeStamp;      // Time Stamp. Show time interval in ms. between
                              // this frame and the previous one. Defined only
                              // for received frames.
} CAN_FRAME,*PCAN_FRAME;

#define NOTIFICATION_FRAME_DATA_LENGTH 4

typedef struct _ NOTIFICATION_FRAME
{
    BYTE bID;                                   // Notification Identifier
    BYTE Data[NOTIFICATION_FRAME_DATA_LENGTH]; // Notification Data
} NOTIFICATION_FRAME,*PNOTIFICATION_FRAME;

typedef struct _USBCAN_MESSAGE
{
    USBCAN_MESSAGE_TYPE USBCANMessageType;
    union
    {
        CAN_FRAME CANFrame;
        NOTIFICATION_FRAME NotificationFrame;
    };
} USBCAN_MESSAGE,*PUSBCAN_MESSAGE;
```

6.3.1 Notification Frames

The user can process the following Notification frames sent by the USB-CAN converter, see Table 6.

Table 6. Notification Frames

Notification Frame	Notification Identifier bID	Notification Data Bytes			
		Data[0]	Data[1]	Data[2]	Data[3]
RUNTIME_ERROR[1]	0x20	Error Code	-	-	-
NOTIFICATION_MESSAGE[1]	0x30	Message Code	-	-	-

[1]Data[1…3] bytes are not used in the described Notification frames and are set to zero.

All other Notification frames should be ignored. Sending Notification frames to the USB-CAN converter is prohibited since they can interfere with proprietary converter functions, for example with flashing new firmware over USB port.

6.3.1.1 Runtime Errors

Runtime Errors are transmitted by the USB-CAN converter or the device driver in case of an error condition, see Table 7.

Table 7. Runtime Errors

Error Name	Error Code	Description
RUNTIME_ERROR_PARSING	1	Converter error during USB packet

Error Name	Error Code	Description
		parsing.
RUNTIME_ERROR_DRIVER_READ_QUEUE_FULL	4	Driver read message queue is full.
RUNTIME_ERROR_IN_CAN_QUEUE_FULL	5	Input CAN Queue is full.
RUNTIME_ERROR_CAN_READ_WRITE_ERROR	6	Error condition during CAN communication.
RUNTIME_ERROR_CAN_IN_MESSAGE_LOST	7	Input CAN frame has been lost.
RUNTIME_ERROR_CAN_WARNING	8	CAN Controller is in Warning state due to communication errors.
RUNTIME_ERROR_CAN_PASSIVE	9	CAN Controller is in Passive state due to communication errors.
RUNTIME_ERROR_CAN_BUS_OFF	10	CAN Controller is in Bus-off state due to communication errors. CAN controller will be reinitialized automatically.
RUNTIME_ERROR_CAN_LOW_POWER[1]	11	CAN transceiver power supply voltage is low. CAN communication can be unstable. Added in AX070505.

[1]Not used in AX070501

6.3.1.2 Notification Messages

Notification Messages are transmitted by the converter to report changes in its status not associated with an error condition, see Table 8.

Table 8. Notification Messages

Message Name	Message Code	Description
NOTIFICATION_CODE_EXT_POWER_ON[1]	1	External power supply has been connected to the converter.
NOTIFICATION_CODE_EXT_POWER_OFF[1]	2	External power supply has been disconnected from the converter.
NOTIFICATION_CODE_SUSPEND_MODE	3	Converter has left Suspend mode. Transmitted immediately after resuming USB operations.
NOTIFICATION_CODE_LOW_POWER	4	USB voltage is less than necessary for stable operation. External power supply should be connected in case of AX070501.
NOTIFICATION_CODE_USB_POWER_ON[1]	5	USB power has been connected. USB cord has been attached to the hub.
NOTIFICATION_CODE_USB_POWER_OFF[1]	6	USB power has been disconnected. USB cord has been detached from the hub.

[1]Not used in AX070505

6.4 Version Information Functions

Name:	**USBCANGetFirmwareVersion()**
Description:	The function gets firmware version information for firmware installed in the USB-CAN converter.

Prototype:	DWORD USBCANGetFirmwareVersion(HANDLE hDeviceHandle, PFIRMWARE_VERSION pFirmwareVersion, LPOVERLAPPED pOverlapped);	
Parameters:	HANDLE hDeviceHandle	Valid device handle to the USB-CAN converter.
	PFIRMWARE_VERSION pFirmwareVersion	Pointer to a FIRMWARE_VERSION structure that will receive a firmware version information.
	LPOVERLAPPED pOverlapped	Pointer to an OVERLAPPED Windows structure defined in Winbase.h for asynchronous operations with the device handle or NULL otherwise.
Return value:	System error code. The function returns ERROR_SUCCESS on successful completion or an error code defined in WinError.h. For more information see GetLastError() function from Windows platform SDK.	
Comments:	The wMainVersionNumber and wAuxVersionNumber members of the FIRMWARE_VERSION structure define the firmware version number. Two auxiliary members: wReserved_1 and wReserved_2 are not used and are set to zeros.	

The FIRMWARE_VERSION structure is defined in USBCAN.h the following way:

```
typedef struct _FIRMWARE_VERSION
{
    WORD wMainVersionNumber;
    WORD wAuxVersionNumber;
    WORD wReserved_1;
    WORD wReserved_2;
} FIRMWARE_VERSION,*PFIRMWARE_VERSION;
```

Name:	**USBCANGetDriverVersion()**	
Description:	The function gets Windows device driver version information for the USB-CAN converter.	
Prototype:	DWORD USBCANGetDriverVersion(HANDLE hDeviceHandle, PDRIVER_VERSION pDriverVersion, LPOVERLAPPED pOverlapped);	
Parameters:	HANDLE hDeviceHandle	Valid device handle to the USB-CAN converter.
	PDRIVER_VERSION pDriverVersion	Pointer to a DRIVER_VERSION structure that will receive a driver version information.
	LPOVERLAPPED pOverlapped	Pointer to an OVERLAPPED Windows structure defined in Winbase.h for asynchronous operations with the device handle or NULL otherwise.
Return value:	System error code. The function returns ERROR_SUCCESS on successful completion or an error code defined in WinError.h. For more information see GetLastError() function from Windows platform SDK.	
Comments:	The DRIVER_VERSION structure presents software version information in the standard Windows form using four variables: wProductMajorPart, wProductMinorPart, wProductBuildPart, wProductPrivatePart. The values of the first two variables: wProductMajorPart, wProductMinorPart will match the major and minor version numbers of the USB-CAN converter driver	

	package. The third and fourth variables: wProductBuildPart, wProductPrivatePart represent values unique to the usbcan.sys kernel driver module.

Name:	**USBCANGetDllVersion()**	
Description:	The function gets dynamic-link library (DLL) version information.	
Prototype:	void USBCANGetDllVersion(PDLL_VERSION pDllVersion);	
Parameters:	PDLL_VERSION pDllVersion	Pointer to a DLL_VERSION structure that will receive a DLL version information.
Return value:	None	
Comments:	The DLL_VERSION structure presents software version information in the standard Windows form using four variables: wProductMajorPart, wProductMinorPart, wProductBuildPart, wProductPrivatePart.	

The DRIVER_VERSION and DLL_VERSION structures are identical and defined in USBCAN.h the following way:

```
typedef struct _DRIVER_VERSION
{
    WORD wProductMajorPart;
    WORD wProductMinorPart;
    WORD wProductBuildPart;
    WORD wProductPrivatePart;
} DRIVER_VERSION,*PDRIVER_VERSION;

typedef struct _DLL_VERSION
{
    WORD wProductMajorPart;
    WORD wProductMinorPart;
    WORD wProductBuildPart;
    WORD wProductPrivatePart;
} DLL_VERSION,*PDLL_VERSION;
```

7 VERSION HISTORY

User Manual Version	SDK Version	Date	Author	Modifications
2.0	2.0.x	May 4, 2022	Olek Bogush	• The SDK was rewritten. For compatibility with the old SDK versions use `_USB_CAN_SDK_V1_XX` macro. • Added Axiomatic USB-CAN converter p/n AX070505.
1.03	1.03	September 6, 2012	Olek Bogush	• USBCAN.h file was replaced with a correct one. • A description of the SDK folder structure was added to the Introduction section of the User Manual.
1.02	1.02	August 28, 2007	Amanda Wilkins	The user manual was reviewed. • European address was changed.
1.01	1.01	May 1, 2007	Olek Bogush	The user manual was changed the following way: • USBCANSetBaudRateSpecial() function description was enhanced. • Acronyms section was added. • Release notes were updated and clarified.
1.00	1.00	January 2, 2007	Olek Bogush	Initial release. Based on the internal ATC-USB-CAN project document “USB-CAN Converter Application Software Interface V1.00”