AI-7280

Central Office Line Simulator

DLL Programmers Guide

Advent Instruments Inc.

Release 4.6

Advent Instruments Inc.
111 - 1515 Broadway Street
Port Coquitlam, BC, V3C6M2
Canada

Internet: techsupport@adventinstruments.com
sales@adventinstruments.com

Web Site: http://www.adventinstruments.com

Telephone: (604) 944-4298
Fax : (604) 944-7488

Contents

1 Getting Started

1.1 Introduction

The AI-7280 Dynamic Link Library (DLL) allows software developers to integrate the functionality of the central office line simulator into a software application written for Microsoft Windows. The functions included in the DLL permit the user application to connect to and control one or more AI-7280 units (connected via serial port or USB) from a common DLL interface.

This manual describes the AI-7280 DLL developer kit that contains all the files required to start using and developing applications with the DLL. The DLL developer kit contains a copy of the Dynamic Link Library (DLL), a demonstration program, examples, and the USB drivers for the AI-7280.

The DLL demonstration program allows the user to experiment with each of the DLL functions through a graphical user interface without writing a line of code. Several code examples are also included which give examples of

- Generating different formats of Type I and Type II CallerID, with
    - Noise impairments
    - Intentionally corrupted checksum
- Sending a simple SMS sequence
- Uploading an playing back waveforms on the AI-7280
- Genearting common call-progress and network tones with cadences.
- Genearating distinctive ringing
- Simulating a simple North American Central Office.
- How to use some of the advanced features of the AI-7280 through the DLL functions including scripting.

1.2 What Has Changed in Rev 4.x

Rev 4.0

- Added functions Wait_For_PulseDial and Get_PulseDial_Stats to make pulse dialing detection simpler
- Added Create_OSI function so that Open Switching Intervals (OSIs) can be generated with more precise timing
- Added the Wait_For_LineFlash function to make the detection of a line flash simpler

Rev 4.1

- Restricted the range of the MaxTime parameter in the Wait_For_HookState and Wait_For_DTMF function to a maximum of 100 seconds to prevent internal timeout errors.
- Fixed minor bug in Get_MeterPulseCount which would erroneously return a "parameter out of range" error.

Rev 4.1b

- Updated documentation to highlight .Net issues
- Added more examples for VB.net and restructured examples directory

Rev 4.2

- Added timing function which allow the user to determine when DTAS and FSK were sent during a CallerID sequence

Rev 4.4

- Fixed initialization bug which prevented connection to AI-7280 when serial number was specified on USB
- Resolved timing issue which could cause communication errors on slower PCs connecting using COM port
- Slight change in directory structure for installed program
- Added a complete Visual Studio 6 C++ project example complete with updated examples
- Minimum system requirements now require Windows 2000 SP4 or greater

Rev 4.5

- Added Set_ BNCOutGain function
- Fixed minor documentation errors

Rev 4.6

- Removed limit checking on TelintFeed Voltage and Current limits to support updated firmware and extended operation range.

1.3 Minimum System Requirements

Computer/Processor:	PC with Pentium 200 MHz or greater
Memory:	256 MB of RAM
Hard Disk Space:	10 MB
Operating System:	Microsoft Windows XP SP2 or greater

1.4 Installing the DLL Developer's Kit

The AI-7280 interface DLL and all related programs, documentation, and sample code can be installed on your PC by executing the file 7280DLLKit.exe. This file is a self-extracting setup utility that will install the files into a directory of your choice. Follow the on screen instructions and the setup program will guide you through the installation process. Once the installation is complete you may install the USB drivers (see following section).

The DLL Developer's Kit will:

- Install the latest version of the DLL into the windows system32 directory
- Install the demonstration program, manual, drivers, and examples into C:\Program Files\Advent\AI-7280 DLL Kit (by default)

1.5 Connecting to the PC

The AI-7280 can be connected to the PC by either its RS-232 port or USB 2.0 compliant full speed (12 Mbit/second) port. Both connectors are present on the rear panel of the unit.

If using the TRsSim software or the Windows DLL to control the AI-7280, the USB connection is preferred because of the higher data rate. It incurs less latency when sending commands and data to the AI-7280. However either connection method will operate with the TRsSim software or the Windows DLL.

1.5.1 USB Driver

The proper drivers must be installed before a USB connection can be established between the AI-7280 and the PC. The WHQL USB driver files are automatically installed with the DLL installation package described above and no further steps should be required.

For detailed step-by-step instructions on installing the driver, see Appendix A: USB Driver Installation. The following is a summary of the installation procedure described in the Appendix.

1. Download the USB driver installation package from our website or from the CDrom.

2. Double click on the self extracting installer. This normally requires using the **administrator** privileges in Vista and Windows 7

3. The installer will decompress the files and prompt you through the steps to install the driver files.

5. A message should be displayed indicating the successful installation of the two driver files.

1.5.2 RS-232

The rear panel 9 pin connector of the AI-7280 is configured as a DCE (Data Communications Equipment) port. As the PC is normally configured as a DTE (Data Terminal Equipment) port, a straight-though cable is required for the connection. Do not use a cross-over cable between the AI-7280 and the PC serial port.

The AI-7280's RS-232 data transmissions are fixed at 8 data bits with 1 stop bit and no parity. The baud rate is adjustable from a default (at power up) setting of 9600 bps to a maximum of 115200 bps. If a break signal is detected by the AI-7280, it will reset its baud rate to 9600 bps. Only three signals are required for communication. These are Tx Data, Rx Data, and Ground. The Request-to-Send (RTS) and Clear-To-Send (CTS) signals have no effect on communication. On the AI-7280's 9 pin RS-232 connector, the Data-Terminal-Ready (DTR) pin is directly wired to the Data-Set-Ready (DSR) pin. As such, if the PC asserts DTR, it can read back that DSR is asserted as well.

1.6 Files Included in the Developer's Kit

By default the developers kit installs into C:\Progam Files\Advent\AI-7280 DLL Kit. The following sections describe the contents of the installation.

Directory	Description
[Installation Directory]	**AI-7280 DLL Demo Program** – This is a small executable that demonstrates each of the DLL function calls through a simple graphical interface.
.\DLL	**7280 DLL** – This directory contains a copy of the AI-7280 DLL and the .lib file required to import this DLL into Microsoft C++ compilers
.\Manual	**DLL Programmers Guide** – This directory will contain the latest version of this document.
.\Drivers	**USB Drivers** – This directory contains a copy of the AI-7280 USB drivers which can be installed through the procedure specified in Section 1.5
.\Examples	**DLL Programming Examples** – This directory contains several examples of how to use this DLL in several different programming environments

1.7 Examples

We have included several simple examples of how to program using the DLL in the **<u>Examples sub-directory of the installation path</u>**. These examples include programming examples in C++, VB6, VB.Net, and Excel (VBA).

These examples collectively demonstrate how to:

- Setup and send TypeI and Type II CallerID messages using multiple signaling types
- Send SMS sequences
- Send DTMF
- Measure Flash timging
- Simulate a simple Central Office(CO)
- Playback waveforms on the AI-7280

If you require assistance with your particular development environment please contact technical support and we will be happy to assist you.

2 Low Level DLL Function Reference

2.1 DLL Function Usage

The AI-7280 DLL is a collection of functions that allows the calling software to control multiple AI-7280 devices through a common interface. All of the DLL functions return a standard error code that indicating if any errors occurred during the function call. See the section on Error Codes.

Before any AI-7280 device functions can be called, communications with an AI-7280 device must be established using the Open_Device() function. This function returns a "deviceid" handle that is used to reference the connected device for all future function calls. Additional calls to the Open_Device function can open connections to other AI-7280 devices and the function will return a unique "deviceid" for each new device.

Once communications are established, the AI-7280 units can be controlled with the interface functions using the "deviceid" parameter to reference the device. Finally, once all the tasks are complete, communications can be terminated using the Close_Device function.

2.1.1 Data Type Summary

The AI-7280 interface DLL accepts and returns several different data types. They are summarized below:

C Data Type	Description / Memory Width
long	32-bit integer value
float	32-bit floating point value
char []	NULL (0) terminated string of characters. Each character is stored as one byte ASCII code. This data type is passed as the 32-bit address of the first byte in the string. All strings must be NULL (0) terminated and contain only printable ASCII characters (32 and above).
float []	32-bit array of floating point values. These values are stored in consecutive locations in memory. This data type is passed as a 32-bit address of the first element of the array.
long []	32-bit array of integer values. These values are stored in consecutive locations in memory. This data type is passed as a 32-bit address of the first element of the array.

2.1.2 Language Compatibility

It is important to note that different programming languages represent fundamental data types with different resolutions. The following table outlines a few subtle differences

Data Type	C/C++	Visual Studio 6	.Net
32-bit Integer	long	Long	Integer
32-bit floating point	float	Single	Single

IMPORTANT NOTE FOR .Net Programmers:

Micorosft .Net defines a 32-bit integer value as an “Integer” not “Long” as it was in Visual Studio 6. For .net programs you will have to declare all the DLL prototypes with Integer parameters for your project to work correctly! The example file AI7280DRV.vb has the appropriate adjustments made for VB.net projects.

Note: Special care must be taken when using string arguments for functions. See the function definition as to the required initialization size before calling the function. Also, the DLL functions will not write any characters past the first NULL in the output string. Therefore you must initialize each string to have a NULL only at the end of the string. For example:

```
char Response[200];

memset(Response,' ',sizeof(Response)-1);    // fill the string with spaces

Response[sizeof(Response)-1]='\0';          // NULL terminate the string

Send_TextCommand( deviceid, "?HS2",1,Response);          // return the string
```

2.1.3 Thread Safety

The AI-7280 DLL is not thread safe. All calls to the AI-7280 DLL must be from the same thread.

2.2 System Functions

2.2.1 Open_Device

Description:

The Open_Device function initializes communications with an AI-7280 device and returns a deviceid identifier that is used to reference the device in later function calls. The Open_Device function can be used to connect to devices on a COMM port or USB depending on the value of the port parameter. If the serial number is specified then this function will only connect to the unit with the matching serial number. If the serial number is not specified (zero length string) then this function will connect to the first AI-7280 unit found on the specified communications channel.

Function Prototype:

```
long Open_Device(long * deviceid, long port, char serial[])
```

Function Parameters:

deviceid The deviceid parameter returns a device communications handle, which is used to reference this device in all further function calls.

port The port parameter specifies which communication channel to use to connect to the AI-7280 device. The acceptable values for the port parameter are shown in the table below

Port Value	Communication Channel
0	USB
1	COM1
2	COM2
3	COM3
4	COM4

serial The serial parameter is a NULL terminated string. If this string is zero length then the serial parameter is ignored and this function connects to the first AI-7280 device located on the communications channel. If the string is non-zero length, it should contain an 8-character serial number of the AI-7280 device to connect to. This string must be in the format “SN12XXXX” (where X represents a digit between 0 and 9). When the serial number is specified, this function will only connect to a unit with the matching serial number.

2.2.2 Close_Device

Description:

The Close_Device function terminates communications with an AI-7280 device and releases the associated communications resources. Close_Device can only close a communications channel that was opened with Open_Device. When the Close_Device function is called the AI-7280 is reset to clear all system settings back to their defaults.

Function Prototype:

```
long CloseDevice(long *deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to terminate communications with. This value must be a valid deviceid returned by a call to Open_Device.

2.2.3 Get_DLLVer

Description:

The Get_DLLVer function returns the major and minor revision codes for the DLL. These values are used to track changes in the DLL software. For example if dllmajor=1 and dllminor=23 then the DLL is version 1.23

Function Prototype:

```
long Get_DLLVer(long *dllmajor,  long *dllminor)
```

Function Parameters:

dllmajor	The dllmajor parameter returns the value of the major revision code for this DLL.
dllminor	The dllmajor parameter returns the value of the minor revision code for this DLL.

2.2.4 Get_ErrorDesc

Description:

The Get_ErrorDesc function translates the error codes returned by the functions in this DLL into a string containing a description of the error that occurred. See the Error Codes section for more information on the possible error codes that can be returned from the DLL.

Function Prototype:

```
long Get_ErrorDesc (long errorcode,  char description[200])
```

Function Parameters:

errorcode	The errorcode parameter should contain the value of an error code returned from one of the functions in this DLL.
description	The description parameter returns a NULL terminated string that will be loaded with the error code description. **Note: Before calling this function insure that this string is allocated to be at least 200 characters long.**

2.2.5 Get_DeviceVer

Description:

The Get_DeviceVer returns the AI-7280 hardware and software version information and serial number.

Function Prototype:

```
long GetDeviceVer(long device, char softversion[64], char hardversion[64] ,
char serialnum[64], char unitid[64])
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
softversion	The softversion parameter is a NULL terminated string that will be loaded with the software version string when this function is called. **Note: Before calling this function insure that the description string is allocated to be at least 64 characters long. See** Data Type Summary **for initialization details.**
hardversion	The hardversion parameter is a NULL terminated string that will be loaded with the hardware version string from the unit when this function is called. **Note: Before calling this function insure that the description string is allocated to be at least 64 characters long. See** Data Type Summary **for initialization details.**
serialnum	The serialnum parameter is a NULL terminated string that will be

	loaded with the serial number string when this function is called. **Note: Before calling this function insure that the description string is allocated to be at least 64 characters long. See** Data Type Summary **for initialization details.**
unitid	The unitid parameter is a NULL terminated string that will be loaded with the unit identification string for the AI-7280 device. **Note: Before calling this function insure that the unitid string is allocated to be at least 64 characters long. See** Data Type Summary **for initialization details.**

2.2.6 Set_Timer

Description:

The Set_Timer function sets the current time value of the slow timer in the AI-7280. The slow timer is used internally in the AI-7280 for timing most signals on the telephone line. Note: this timer is reset to zero when Open_Device is called.

Function Prototype:

```
long Set_Timer(long deviceid, float time)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
time	The time parameter specifies the time value (in seconds) to load into the slow timer.

2.2.7 Get_Timer

Description:

The Get_Timer function returns the value of the slow timer in the AI-7280 in seconds.

Function Prototype:

```
long Get_Timer(long deviceid, float * time)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
time	The time parameter returns the value of the slow timer in the AI-7280 in that location.

2.2.8 Waitms

Description:

The Waitms function pauses for the specified number of milliseconds. Note: this function uses the internal windows timer as a timing reference. Therefore, timing resolution may vary depending on computer hardware and the version of Windows.

Function Prototype:

```
long Waitms (long milliseconds)
```

Function Parameters:

milliseconds	The milliseconds parameter specifies the number of milliseconds for this function to wait. Note: the minimum resolution of the wait may vary depending on the computer hardware and the version of Windows.

2.2.9 Get_Error

Description:

The Get_Error function returns any error conditions internal to the AI-7280. If the errorcode value returned is non-zero then an error has been detected in the AI-7280. Multiple error codes can be read out through multiple calls to the Get_Error function. If a zero value is returned then it indicates there are no further error conditions present. See the AI-7280 documentation for details on these error codes.

Function Prototype:

```
long Get_Error(long deviceid, long * errorcode)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
errorcode	The errorcode parameter returns an AI-7280 internal error code. If this value is non-zero it indicates that an error has been detected. See the AI-7280 documentation for more details.

2.2.10 Get_NumUSBDevices

Description:

The Get_NumUSBDevices function returns the number of AI-7280 devices connected to the PC using the USB connection. The serial number of each of these units can be read using the Get_USBSerial function. Note: This function doesn't count devices that are in use. If a communications are already established using this DLL or any other program then the device is not included in the count.

Function Prototype:

```
long Get_NumUSBDevices(long *numdevices)
```

Function Parameters:

numdevices	The numdevices parameter returns the number of AI-7280 devices connected via USB.

2.2.11 Get_USBSerial

Description:

The Get_USBSerial function returns the serial number of an AI-7280 unit connected to the USB bus.

Function Prototype:

```
long Get_USBSerial(long devicenum, char serial[9])
```

Function Parameters:

devicenum	The devicenum parameter indexes which device's serial number to retrieve. The devicenum should range from 1 to numdevices where numdevices is the number of devices connected on the USB bus.
serial	The serial parameter is NULL terminated string that is loaded with the serial number of the AI-7280 device when the function is called. **Note: Before calling this function insure that the serial string is allocated to be at least 64 characters long. See** Data Type Summary **for initialization details.**

2.3 Telephone Interface Functions

2.3.1 Set_TelIntFeed

Description:

The Set_TelIntFeed sets the DC feed parameters for the telephone line interface. The voltage parameter sets the nominal on-hook DC feed voltage (in units of Vdc). The current parameter sets the off-hook loop current (in mA) when the unit is in constant current mode. If the isvoltfeedmode parameter is set to a non-zero value then the unit is configured in constant voltage feed mode and only the resistance of the telephone interface and the CPE load limits the DC loop current.

Function Prototype:

long Set_TelIntFeed(long deviceid, float voltage, float current, long isvoltfeedmode)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
voltage	The voltage parameter sets the nominal on-hook DC line voltage for the telephone line interface (in units of Vdc)
current	The current parameter sets the off-hook loop current (in units of mA) when the telephone line is in constant current mode.
isvoltfeedmode	The isvoltfeedmode parameter specifies the DC feed mode of the telephone line interface. If the isvoltfeedmode parameter is set to 0 then the telephone line interface will operate in constant current mode. If this parameter is non-zero then the telephone line interface will operate in constant voltage mode.

2.3.2 Set_TelIntImped

Description:

The Set_TelIntImped function sets the AC impedance parameters for the telephone line interface. The impedance parameter sets the AC output impedance of the telephone line interface and the balance parameter sets the AC load impedance setting for the trans-hybrid balance network.

Function Prototype:

long Set_TelIntImped(long deviceid, long impedance, long balance)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
impedance	The impedance parameter sets the AC output impedance of the telephone line interface. The allowed values are listed in the table below

impedance setting	AC Output Impdance
0	600Ω
1	900Ω
2	TBR-21
3	Optional Impedance

balance	The balance parameter sets the AC load impedance setting for the trans-hybrid balance network. This setting should be set to match the impedance of a load on the telephone line to allow the best performance of the hybrid network. The balance parameter accepts the same values as the impedance parameter.

2.3.3 Set_TelIntPolarity

Description:

The Set_TelIntPolarity function sets the line polarity for the telephone line interface. If the value of the isreversed parameter is non-zero then the telephone line is set to reversed polarity. If the isreversed parameter is zero then the telephone line is set to normal polarity.

Function Prototype:

```
long Set_TelIntPolarity(long deviceid, long isreversed)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
isreversed	The isreversed parameter sets the telephone line polarity. A zero value sets the telephone line interface to normal polarity. A non-zero value sets the telephone interface to reversed polarity.

2.3.4 Set_TelIntDisconnect

Description:

The Set_TelIntDisconnect function disconnects or connects the telephone line interface circuitry from the telephone line connector on the front of the AI-7280. If the disconnect parameter is zero then the telephone line interface is connected to the telephone jack. If the parameter is non-zero then the line is disconnected. Note: signals can still be measured at the telephone jack when the telephone line interface is disconnected.

Function Prototype:

long Set_TelIntDisconnect(long deviceid, long disconnect)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
disconnect	The disconnect parameter specifies whether to connect or disconnect the telephone line interface to the telephone jack on the front of the AI-7280. If the value is zero then the telephone line interface is connected. If the value is non-zero then the telephone interface circuitry is disconnected from the telephone jack.

2.3.5 Set_TelIntMeasPoint

Description:

The Set_TelIntMeasPoint function sets the source and measurement range of the telephone line measurements. This function also sets the speed of the filter on the DC voltage and current measurements.

Function Prototype:

long Set_TelIntMeasPoint(long deviceid, long measpoint, long measrange, long dcspeed)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
measpoint	The measpoint parameter specifies the source of the telephone line measurements. If measpoint is zero then the measurements are made on the inner pair of the telephone jack (connected to the telephone line interface). If measpoint is non-zero then the measurements are made on the outside pair of the telephone jack.

measrange	The measrange parameter specifies the measurement range for the telephone line measurements. If measrange is zero then high gain is selected and the maximum signal level is 5Vrms before clipping occurs. If the measrange value is non-zero then low gain is selected for measuring large voltage signals such as ringing.
dcspeed	The dcspeed parameter specifies the speed of the filtering performed on the DC line voltage and current measurements. The values for dcspeed and the associated filter speed are shown in the table below. Note: the rejection specified refers to the filters AC signal rejection performance at a specified frequency.

dcspeed	Filter Accuracy/Speed
0	No filtering (instantaneous samples)
1	40dB rejection at 100Hz (0.5 second settling time)
2	40dB rejection at 30Hz (2 second settling time)
3	40dB rejection at 10Hz (5.5 second settling time)

2.3.6 Get_TelIntHookStatus

Description:

The Get_TelIntHookStatus function returns the hook switch status of the AI-7280 and also indicates whether or not the unit has gone into protected mode due to high unbalanced current. When the AI-7280 goes into protected mode it will generate an internal error code and the display will change to show this error (indicated by the flashing status LED). To clear the display and return the LEDs to normal operation use the Get_Error function to read the error code and the display will reset and resume displaying the telephone line status.

Function Prototype:

```
long Get_TelIntHookStatus(long deviceid, long *offhook, long *isunbalanced)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
offhook	The offhook value returns the hook switch detection status. The offhook parameter will be set to a non-zero value if the AI-7280 is off hook and a zero value if the unit is on-hook.
isunbalanced	The isunbalanced parameter returns a non-zero value if the AI-7280 has entered a protected mode due to high unbalanced current flow. If the isunbalanced parameter is zero then the AI-7280 is operating normally.

2.3.7 Ramp_TelIntVoltage

Description:

The Ramp_TelIntVoltage function ramps the DC line voltage from the current voltage setting to a specified destination voltage at a given slew rate. The voltage can either ramp to a destination voltage of the same polarity, or slew to a destination voltage of opposite polarity.

Function Prototype:

long Ramp_TelIntVoltage(long deviceid, float vdestination, float ramprate, long wait)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
vdestination	The vdestination parameter specifies the destination voltage (in Vdc) that the voltage will ramp to. If the voltage is positive then the voltage ramps to the destination without changing polarity. If the voltage is negative then the line voltage will ramp down to 0V and continue to the destination voltage of the opposite polarity of the starting voltage.
ramprate	The ramprate parameter specifies the slew rate for the ramp in V/ms. This parameter can be between the values 0-20. Positive values cause the voltage to ramp at the desired ramp rate. A zero value causes no-change or causes a ramp in progress to stop.
wait	The wait parameter specifies whether the DLL should wait for the ramp to complete before returning control to the calling program. If wait is non-zero then the DLL waits for the ramp to complete before returning control to the calling program. If zero then the function returns control immediately after initiating the ramp.

2.3.8 Set_TelIntHookThreshold

Description:

The Set_TelIntHookThreshold function sets the current threshold used for hook switch detection in mA. Any loop currents excepting this threshold will cause the AI-7280 to go off hook and begin regulating the DC loop current (in constant current mode).

Function Prototype:

long Set_TelIntHookThreshold(long deviceid, float hookthresh)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
hookthresh	The hookthresh parameter sets the current threshold (in mA) for hook switch detection. This parameter can range from 5 to 25mA.

2.3.9 Get_TimeStamps

Description:

The Get_TimeStamps function gets the time stamp of the last on-hook and off-hook transitions in the telephone line state.

Function Prototype:

long Get_TimeStamps(long deviceid, float *onhooktime, float *offhooktime)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
onhooktime	The onhooktime returns the value to the timestamp (in seconds) of the last transition in line state from off to on hook. This value can also be set using the Set_OnHookTime function to assist in timing detection.
offhooktime	The offhooktime returns the value to the timestamp (in seconds) of the last transition in line state from on to off hook. This value can also be set using the Set_OffHookTime function to assist in timing detection.

2.3.10 Set_OnHookTime

Description:

The Set_OnHookTime function sets the onhooktime value - that can be read using the Get_TimeStamps function. The onhooktime is the timestamp (in seconds) of the last time the telephone line state transitioned from off-hook to on-hook. This function is intended to assist in detecting new on-hook transistions. For example: If set the onhooktime to 0 then when you read back a non-zero value it indicates that a new on-hook transition has occurred.

Function Prototype:

```
long Set_OnHookTime(long deviceid, float onhooktime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
onhooktime	The onhooktime parameter specifies the time value (in seconds) to write into the onhooktime property of the AI-7280.

2.3.11 Set_OffHookTime

Description:

The Set_OffHookTime function sets the offhooktime value that can be read using the Get_TimeStamps function. The offhooktime is the timestamp of the last time the telephone line state transitioned from on-hook to off-hook. This function is intended to assist in detecting new off-hook transitions. For example: If set the offhooktime to 0 then when you read back a non-zero value it indicates that a new off-hook transition has occurred)

Function Prototype:

```
long Set_OffHookTime(long deviceid, float offhooktime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
offhooktime	The offhooktime parameter specifies the time value (in seconds) to write into the offhooktime property in the AI-7280.

2.3.12 Wait_For_HookState

Description:

This function waits for a specified amount of time for the hook switch state to change to the desired state. If the hookswitch changes to the desired state within the timeout then IsChanged will return a non-zero value. If the telephone line interface is already in the desired state when this function is called it will return immediately.

Function Prototype:

long Wait_For_HookState (long deviceid, long MaxTime, long ExpectedHookState, long *IsChanged)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
MaxTime	This parameter specifies the maximum time (ms) to wait for the hookswitch to change states. This value may set to positive values up to 100,000 ms (100 seconds).
ExpectedHookState	This parameter specifies the expected hook state. Pass 0 for on-hook, and the value 1 for off hook
IsChanged	When the function completes this value will be non-zero if the hook switch state changed to the desired state. If the function timed out then this value will be set to zero.

2.3.13 Wait_For_LineFlash

Description:

This function waits for a specified amount of time for a line flash to occur and returns the timing of the line flash (if detected). The line flash must satisfy the minimum and maximum duration requirements and must start before MaxTime has expired. The line state must be off-hook before this function is called. If the phone line is on-hook this function will return immediately.

Function Prototype:

long Wait_For_LineFlash (long deviceid, long MaxTime , long MinDuration, long MaxDuration, long *Detected, float *StartTime, float *StopTime)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
MaxTime	This parameter specifies the maximum time (ms) to wait for the line flash to start
MinDuration	This parameters specifies the minimum duration (ms) requirement for a valid line flash

MaxDuration	This parameter specifies the maximum duration (ms) requirement for a valid line flash.
Detected	If a valid line flash is detected, this will return a non-zero value
StartTime	This will return the value of the timer (seconds) when the valid line flash started.
StopTime	This will return the value of the timer (seconds) when the valid line flash ended.

2.3.14 Create_OSI

Description:

This function generates an Open Switching Interval (OSI) with precise timing. This function minimizes the effects of communication delays which can affect the duration of an OSI which is created manually through separate DLL function calls. This function returns when the OSI is completed.

Function Prototype:

```
long Create_OSI(long deviceid, long Duration)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
Duration	This parameter specifies the duration (ms) of the OSI to be generated.

2.4 Routing Functions

2.4.1 Set_MeterSource

Description:

The Set_MeterSource function selects the signal source for the AC level meter in the AI-7280.

Function Prototype:

long Set_MeterSource(long deviceid, long source)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
source	The source property specifies the signal source for the AC level meter in the AI-7280. The possible values for the source value are listed in the following table.

source value	AC Level Meter Source
0	None – Level meter is off
1	Telephone interface RX
2	Telephone interface hybrid RX
3	BNC input
4	Generator
5	Telephone interface TX

2.4.2 Set_AnalyzerSource

Description:

The Set_AnalyzerSource function selects the signal source for the analyzer functions of the AI-7280 (DTMF detector, FSK decoder, and AC capture)

Function Prototype:

long Set_AnalyzerSource(long device, long source)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
source	The source property specifies the signal source for the analyzer functions of the AI-7280 (DTMF detector, FSK decoder, etc.). The possible values for the source value are listed in the following table.

source value	AC Level Meter Source
0	None – Level meter is off
1	Telephone interface RX
2	Telephone interface hybrid RX
3	BNC input
4	Generator
5	Telephone interface TX

2.4.3 Set_BNCOutSource

Description:

The Set_BNCOutSource function selects the signal source for the BNC output on the back of the AI-7280.

Function Prototype:

long Set_BNCOutSource(long device, long source)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
source	The source property specifies the signal source for BNC output on the AI-7280. The possible source values are listed below:

source value	BNC Output Source
0	None
1	Telephone interface RX
2	Telephone interface hybrid RX
3	BNC input
4	Generator
5	Telephone interface TX
6	Input to AC level meter following optional filter
7	Input to secondary level meter following notch filters
8	Output of analyzer LPF (input to DTMF detector and FSK decoder)

2.4.4 Set_BNCOutGain

Description:

This function sets the gain applied to signals routed to the BNC output.

Function Prototype:

long Set_BNCOutGain(long device, float gain)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
Gain	The gain (volts/volt) applied to the all the signal generators before they are applied to the BNC output

2.4.5 Set_TelIntGenGain

Description:

The Set_TelIntGenGain function sets the gain applied to all the signal generators before they are applied to the telephone line. This function allows the simultaneous muting, attenuation, or amplification of all signal generators before the signals are applied to the telephone line.

Function Prototype:

```
long Set_TelIntGenGain(long device, float gain)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
gain	The gain (volts/volt) applied to the all the signal generators before they are applied to the telephone line

2.4.6 Set_BNCInGain

Description:

The Set_BNCInGain function sets the gain applied to the BNC input signal before it is mixed in with the other signal generators and applied to the telephone line.

Function Prototype:

```
long Set_BNCInGain(long device, float gain)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
gain	The gain property sets the gain applied to the BNC input signal before it is mixed in with the other signal generators and applied to the telephone line.

2.5 Measurement Functions

2.5.1 Set_MeterSpeed

Description:

The Set_MeterSpeed function sets the settling time and low frequency accuracy of the AC level meter. The meter speed should be selected to suit the signal frequency as well as the timing requirements of the particular application. In general, the longer the settling time, the greater the low frequency accuracy.

Function Prototype:

long Set_MeterSpeed(long device, long acspeed)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
acspeed	The acspeed parameter specifies the settling speed and low frequency accuracy of the AC level meter.

acspeed	Meter Settling Time
0	**Very Fast.** 0.025 second settling time ±0.1dB accuracy down to 500Hz
1	**Medium.** 0.15 second settling time and ±0.1dB accuracy down to 100Hz
2	**Slow** 0.4 second settling time and ±0.1dB accuracy down to 30 Hz
3	**Very Slow**. 1.1 second settling time and ±0.1dB accuracy down to 10Hz

2.5.2 Get_MeterReading

Description:

The Get_MeterReading function returns the latest level and frequency reading from the AC level meter and frequency counter in the AI-7280.

Function Prototype:

```
long Get_MeterReading(long deviceid,  float *level, float *freq)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
level	The level property returns the latest level reading from the AC level meter when the Get_MeterReading function is called.
freq	The freq parameter returns the latest frequency measurement from the frequency counter when the Get_MeterReading function is called.

2.5.3 Get_THDReading

Description:

The Get_THDReading function calculates the Total Harmonic Distortion plus Noise (THD+N) for the signal on the telephone line. This function assumes that any band limiting filter and the notch filters have already been set up for the signal on the line. This function measures the main AC level and the AC level after the notch filters and returns the ratio of the two.

Function Prototype:

```
long Get_THDReading(long device, float *thd)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
thd	The thd parameter returns the ratio of the notched AC level and the main AC level. If the notch filters are set to the center frequencies of a tone on the telephone line then the value returned by this function will reflect the THD+N of the signal on the telephone line.

2.5.4 Get_TelIntLineVolt

Description:

The Get_TelIntLineVolt function returns the latest measurement of the DC voltage on the telephone line interface. The settling time of this measurement is dependent on the dcspeed parameter, which can be set in the Set_TelIntMeasPoint function.

Function Prototype:

```
long Get_TelIntLineVolt(long device, float *voltage)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
voltage	The voltage parameter returns the latest DC line voltage measurement. The settling time and low frequency accuracy of this measurement can be adjusted by specifying the dcspeed in the Set_TelIntMeasPoint function.

2.5.5 Get_TelIntLoopCurrent

Description:

The Get_TelIntLoopCurrent function returns the current measurement of the DC loop current on the telephone line. The settling time of this measurement is dependent on the dcspeed parameter, which can be set in the Set_TelIntMeasPoint function.

Function Prototype:

```
long Get_TelIntLoopCurrent(long device, float *current)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
current	The current parameter returns the latest measurement of the DC loop current on the telephone line in mA.

2.5.6 Get_TelIntUnbalCurrent

Description:

The Get_TelIntUnbalCurrent function returns the latest measurement of the unbalanced current flowing out of the telephone interface. Note: excessive unbalanced current draw can cause the AI-7280 to go into a protected mode.

Function Prototype:

long Get_TelIntUnbalCurrent(long device, float *unbalcurrent)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
unbalcurrent	The unbalcurrent returns the latest measurement of the unbalanced current flowing out of the telephone interface in mA.

2.5.7 Set_Filter

Description:

The Set_Filter function sets the filter type and corner frequencies for the main filter bank in the AI-7280 unit. This filter bank conditions the signals before the AC level meter and frequency counter. The signal source for the filter bank and meter can be selected using the Set_MeterSource function.

Function Prototype:

long Set_Filter(long device, long type, float lfreq, float hfreq)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
type	The type parameter specifies the type of filter loaded in the main filter bank. The possible filter types are shown in the table below.

type	Filter Type
0	No filter
1	Fourth order Butterworth low-pass filter. The corner frequency for this filter is set by the lfreq parameter
2	Fourth order Butterworth high-pass filter. The corner frequency for this filter is set by the hfreq parameter

3	Fourth order Butterworth low-pass and high-pass filter. The high and low corner frequencies are set by the hfreq and lfreq parameter respectively
4	Fourth order bandpass filter. The center frequency for this filter type is set by the lfreq parameter
5	Single notch filter. The center frequency for this filter is set by the lfreq parameter
6	Dual notch filter. The center frequencies for these filters are set by the lfreq and hfreq parameters
7	DTMF low pass filter
8	DTMF high pass filter
9	C-Message filter

lfreq	The lfreq parameter specifies the low corner frequency (Hz) for the filter specified. This parameter can range from 20 to 10000Hz.
hfreq	The hfreq parameter specifies the high corner frequency (Hz) for the filter specified. This parameter can range from 20 to 10000Hz.

2.5.8 Set_NotchFilter

Description:

The Set_NotchFilter function sets the number of notch filters in the notch filter bank and the center frequencies for each notch filter. These filters can be used to calculate the THD+N for tones on the telephone line.

Function Prototype:

```
long Set_NotchFilter(long device, long numnotch, float freq1, float freq2)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numnotch	The numnotch parameter specifies the number of notch filters in the notch filter bank. There can be up to two notch filters loaded at one time. If the number of notches is set to zero then no filtering is performed by the notch filter bank.
freq1	The freq1 parameter specifies the center frequency (Hz) of the first notch filter loaded into the notch filter bank This parameter can range from 20 to 10000Hz
freq2	The freq2 parameter specifies the center frequency (Hz) of the second notch filter loaded into the notch filter bank This parameter can range from 20 to 10000Hz

2.6 Tone Generation Functions

2.6.1 Set_Tone

Description:

The Set_Tone function sets the level and frequency of one or more of the tone generators on the AI-7280. This function can be called at any time to change the level or frequency of any of the tone generators. The startphase parameter specifies the starting phase of the tone generators in degrees (0-360). This function call will not affect the frequency or level of other signal generators (FSK,DTMF,MF, AM etc) if this function is called while any of these generators are active.

Function Prototype:

```
long Set_Tone(long device, long tonemask, float level, float freq, float
startphase)
```

Function Parameters:

deviceid The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

tonemask The tonemask parameter is a bit mask that indicates which of the four tone generators to modify. The tonemask value can range from 0 to 15. The bit assignments in the bit mask are listed in the following table.

tonemask bit	Tone Generator
0	Tone A
1	Tone B
2	Tone C
3	Tone D

All tone generators with the corresponding bit set in the tonemask will be updated with the new level and frequency.

level The level parameter specifies the level of the tone generator(s) in Vrms. The level parameter can range from 0 to 4 Vrms.

freq The freq parameter specifies the frequency of the tone generator(s) in Hz. The freq value can range from 20 to 18000Hz.

startphase The startphase parameter specifies the starting phase of the tone generator in degrees. This is the starting phase for every tone burst produced by the tone generator. This parameter can range from 0 to 360 degrees.

2.6.2 Set_ToneShape

Description:

The Set_ToneShape function sets the wave shape of one or more of the tone generators. The tone shape can be sine wave, triangle wave, square wave, or user-defined. This function can be called at any time to change the wave shape of any tones being generated. This function call will not affect the wave shape of other signal generators (FSK,DTMF,MF, AM etc) if this function is called while any of these generators are active.

Function Prototype:

long Set_ToneShape(long device, long tonemask, long shape)

Function Parameters:

deviceid The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

tonemask The tonemask parameter is a bit mask that indicates which of the four tone generators to modify. The tonemask value can range from 0 to 15. The bit assignments in the bit mask are listed in the following table.

tonemask bit	Tone Generator
0	Tone A
1	Tone B
2	Tone C
3	Tone D

All tone generators with the corresponding bit set in the tonemask will be updated with the new wave shape

shape The shape parameter specifies the wave shape to be used by the tone generator(s) selected by the tone mask. The allowed values for the shape parameter are listed in the function below.

shape	Ringing Waveshape
0	Sine wave
1	Triangle wave
2	Square wave
3	User defined shape. This shape can be set using the Load_UserWaveShape function

2.6.3 Start_Tone

Description:

The Start_Tone function starts one or more tone generators on the AI-7280 in one of three modes. If the usepattern parameter is non-zero then this function starts the tone generator with the pattern information specified using the Set_TonePattern function. If the usepattern parameter is zero and the duration is positive then this function generates a tone burst for the length of time specified by the duration parameter. If the usepattern parameter is zero and the duration parameter is negative then the tone generator is enabled for an indefinite period of time. If the wait parameter is non-zero then this function waits for the completion of the tone pattern or burst before returning control to the calling program. If an infinite duration is specified (duration < 0) then the wait parameter is ignored and control is returned immediately.

This function can produce one tone pattern or single duration burst at a time on the AI-7280. Calling Start_Tone with a pattern or duration specified will stop any other pattern in progress and start the new pattern. However, tones can be turned on with infinite duration or disabled during a tone pattern without disturbing the pattern timing.

The tone generators in the AI-7280 are also allocated to be used for other resources in addition to the simple tone generator functions (FSK,AM,DTMF,MF). If any of the tone generators specified in the Start_Tone functions are in use by another signal generator this function will return an error code indicating a resource conflict.

Function Prototype:

long Start_Tone(long device, long tonemask, long usepattern, float duration, long wait)

Function Parameters:

deviceid The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

tonemask The tonemask parameter is a bit mask that indicates which of the four tone generators to modify. The tonemask value can range from 0 to 15. The bit assignments in the bit mask are listed in the following table.

tonemask bit	Tone Generator
0	Tone A
1	Tone B
2	Tone C
3	Tone D

All tone generators with the corresponding bit set in the tonemask will be updated with the new level and frequency

usepattern The usepattern parameter specifies whether a tone pattern is to be generated. If this value is non-zero then a tone pattern is generated. If this value is zero the tone pattern is not generated.

duration The duration parameter specifies the duration of a single tone burst (in milliseconds) if no tone pattern is used. If this value is positive then a tone burst is produced for the specified number of milliseconds. If the value of duration is negative then tone is turned on indefinitely.

wait The wait parameter specifies whether this function should wait for the completion of the tone burst(s) before returning control to the calling program. If an infinite duration is specified (usepattern=0 duration < 0) then this parameter is ignored and control is returned immediately.

2.6.4 Stop_Tone

Description:

The Stop_Tone function stops the specified tone generators. If the tone generator is being used in a tone pattern or single duration burst, then only the tone generators specified will be stopped – any other tones used in the pattern or single duration burst will continue to generate the pattern or burst.

Function Prototype:

long Stop_Tone(long device, long tonemask)

Function Parameters:

deviceid The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

tonemask The tonemask parameter is a bit mask that indicates which of the four tone generators to modify. The tonemask value can range from 0 to 15. The bit assignments in the bit mask are listed in the following table.

tonemask bit number	Tone Generator
0	Tone A
1	Tone B
2	Tone C
3	Tone D

All tone generators with the corresponding bit set in the tonemask will be updated with the new waveshape

2.6.5 Set_TonePattern

Description:

The Set_TonePattern function sets the timing parameters used when generating tone patterns. The ontime parameter specifies the duration of a tone burst in milliseconds. The offtime parameter specified the time interval between tone bursts. The cycles parameter specifies the number of tone bursts to generate.

Function Prototype:

```
long Set_TonePattern(long device, float ontime, float offtime, long cycles)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
ontime	The ontime parameter specifies the duration of each tone burst (in milliseconds). This value must be greater than zero.
offtime	The offtime parameter specifies the time interval between each tone burst (in milliseconds). This value must be greater than zero.
cycles	The cycles parameter specifies the number of tone bursts to generate. This value must be greater than zero.

2.6.6 Get_ToneStatus

Description:

The Get_ToneStatus function returns the current level and frequency setting of a single tone generator and also indicates if the tone generator is currently in use. This function only returns the status of the tone generator corresponding to the lowest bit set in the tone mask. ie if the tone mask = 14 then this function returns the status of ToneB.

Function Prototype:

long Get_ToneStatus(long device, long tonemask, float *level, float *freq, long * enabled)

Function Parameters:

deviceid The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

tonemask The tonemask parameter is a bit mask that indicates which of the four tone generators to read. The tonemask value can range from 1 to 15. The bit assignments in the bit mask are listed in the following table.

tonemask bit	Tone Generator
0	Tone A
1	Tone B
2	Tone C
3	Tone D

level The level parameter returns the tone generator level in units of Vrms

freq The freq parameter returns the tone generator frequency setting in units of Hz

enabled The enabled property returns a value indicating whether the tone generator is in use. If this value is zero then the tone generator is inactive. If this value is non-zero then the tone generator is active. If a pattern is being generated then the tone generator will always return an active status - even in the time intervals between tone bursts.

2.7 AM Modulation Functions

2.7.1 Set_AMMod

Description:

The Set_AMMod function sets the signal parameters for the AM modulator. This function specifies the level, frequency, and wave shape for a carrier tone as well as the frequency, wave shape, and modulation depth for the modulating tone. This function can be called at any time to update the AM modulator and will not modify the levels or frequencies of any other active signal generators.

Function Prototype:

long Set_AMMod(long deviceid, float carrierlevel, float carrierfreq, long carriershape, float modfreq, long modshape, float depth)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
carrierlevel	The carrier level parameter specifies the level of the carrier tone in Vrms. The carrierlevel value can range from 0 to 4Vrms.
carrierfreq	The carrierfreq parameter specifies the frequency of the carrier tone in Hz. The carrierfreq value can range from 20 to 18000Hz.
carriershape	The carriershape parameter specifies the wave shape of the carrier tone. The possible values for the shape are shown in the table below.

carriershape	Carrier WaveShape
0	Sine wave
1	Triangle wave
2	Square wave
3	User defined shape. This shape can be set using the Load_UserWaveShape function

modfreq	The modfreq parameter specifies the frequency of the modulating tone in Hz. The modfreq value can range from 20 to 18000 Hz.
modshape	The modshape parameter specifies the wave shape of the modulating tone. The modshape has the same possible values as the carriershape parameter.
depth	The depth parameter specifies the depth of the AM modulation in percent. The depth value can range from 0 to 100%

2.7.2 Start_AMMod

Description:

The Start_AMMod function starts the AM modulator on the AI-7280. Since the AM modulator uses tone generators A and B, calling this function will halt any tones or tone patterns being generated on either of these tone generators. The AM modulator cannot be started if the FSK generator is active since it also uses tone A.

Function Prototype:

```
long Start_AMMod(long device)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.7.3 Stop_AMMod

Description:

The Stop_AMMod function stops the AM modulator on the AI-7280.

Function Prototype:

```
long Stop_AMMod(long device)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.7.4 Get_AMModStatus

Description:

The Get_AMModStatus function indicates if the AM modulator is active.

Function Prototype:

```
long Get_AMModStatus(long device, long *isactive)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

isactive	The isactive parameter returns a non-zero value if the AM modulator is active. The isactive parameter returns zero if the AM modulator is inactive.

2.8 Ringing Functions

2.8.1 Set_Ring

Description:

The Set_Ring function sets the level (Vrms) and frequency (Hz) and DC voltage of the ring generator on the AI-7280. This function can be called at any time. If this function is called while ringing is active then the level and frequency of the ringing signal will immediately change to the new settings.

Function Prototype:

```
long Set_Ring(long device, float level, float freq, float dc)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
level	The level parameter specifies the ringing level in Vrms. The ringing level can range from 0 to 80Vrms.
freq	The freq parameter specifies the frequency of the ringing signal in Hz. The ringing frequency can range from 10 to 100Hz.
dc	The dc parameter specifies the dc voltage during ringing in volts DC. This parameter only affects the DC voltage during ringing; when ringing is inactive the DC line voltage reverts to the line voltage set in the Set_TelIntFeed function.

2.8.2 Set_RingPattern

Description:

The Set_RingPattern function sets the timing parameters used when generating a ringing pattern. The ontime parameter sets the duration of each ringing burst (in milliseconds). The offtime parameter sets the time interval between ringing bursts (in milliseconds). The cycles parameter sets the number of ringing bursts to generate

Function Prototype:

```
long Set_RingPattern(long device, float ontime, float offtime, long cycles)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device

	function.
ontime	The ontime parameter specifies the duration of each ringing burst (in milliseconds). This value must be greater than zero.
offtime	The offtime parameter specifies the time interval between each ringing burst (in milliseconds). This value must be greater than zero.
cycles	The cycles parameter the number of ringing bursts to generate. This value must be greater than zero.

2.8.3 Set_RingShape

Description:

The Set_RingShape function sets the waveshape for the ringing signal. The ring generator supports sine wave, triangle wave, square wave, and user defined wave shapes.

Function Prototype:

```
long Set_RingShape(long device, long shape)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
shape	The shape parameter specifies the wave shape to be used by the ring generator. The allowed values for the shape parameter are listed in the function below.

shape	Ringing Waveshape
0	Sine wave
1	Triangle wave
2	Square wave
3	User defined shape. This shape can be set using the Load_UserWaveShape function

2.8.4 Start_Ring

Description:

The Start_Ring function starts the ring generator on the AI-7280 in one of three modes. If the usepattern parameter is non-zero then this function starts ringing with the pattern information specified. If the usepattern parameter is zero and the duration is positive then this function generates a ring burst for the length of time specified by the duration parameter. If the usepattern parameter is zero and the duration parameter is negative then the ringing generator is started for an indefinite period of time. The startphase parameter specifies the starting phase of the ring generator in degrees (0-360). If the wait parameter is non-zero then this function waits for the completion of the ringing pattern or burst before returning control to the calling program. If an infinite duration is specified (duration < 0) then the wait parameter is ignored and control is returned immediately.

If the telephone line goes off hook while the AI-7280 is generating a ringing pattern or a fixed length burst, the ringing will stop and this function will return control immediately.

When the ringing generator is started all other functions (FSK, MF, DTMF, tones, noise, etc.) are stopped. While ringing is enabled, no other signal generators can be started.

Function Prototype:

```
long Start_Ring(long device, long usepattern, float duration, float startphase,
long wait)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
usepattern	The usepattern parameter specifies whether a ringing pattern is to be generated. If this value is non-zero then a ringing pattern is used. If this value is zero the ringing pattern is not used.
duration	The duration parameter specifies the duration of the ringing burst (in milliseconds) if no ringing pattern is used. If this value is positive then a ringing burst is produced for the specified number of milliseconds. If the value of duration is negative then ringing is turned on indefinitely.
wait	The wait parameter specifies whether this function should wait for the completion of ringing before returning control to the calling program. If an infinite duration is specified (usepattern=0 and duration < 0) then this parameter is ignored and control is returned immediately.

2.8.5 Stop_Ring

Description:

The Stop_Ring function stops the ring generator.

Function Prototype:

```
long Stop_Ring(long device)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.8.6 Get_RingStatus

Description:

The Get_RingStatus function returns the current level and frequency setting of the ring generator and also indicates if the ring generator is currently active.

Function Prototype:

```
long Get_RingStatus(long device, float * level, float *freq, long *enabled)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
level	The level parameter returns the ring generator level when this function is called. The level value is in units of Vrms
freq	The freq parameter returns the ring generator frequency setting when this function is called. The freq value is in units of Hz
enabled	The enabled property returns a value indicating whether the ring generator is active. If this value is zero then the ring generator is inactive. If this value is non-zero then the ring generator is active. If a pattern is being generated then the ring generator will always return an active status (even in the time intervals between ringing bursts)

2.8.7 Set_RingCadence

Description:

This function allows the generation of more advance ringing patterns than the Set_RingPattern function. Using this function, each ringing burst can have between 1 and 3 ringing pulses with programmable durations and off-times.

Function Prototype:

long Set_RingCadence (long deviceid, long Pulses, float OnTime1, float OffTime1, float OnTime2, float OffTime2, float OnTime3, float OffTime3, long Cycles)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numpulses	This parameter sets the number of pulses (1 to 3) to generate in each ringing burst.
ontime1	This parameter sets the duration (ms) of the first ringing pulse
offtime1	This parameter sets the time interval from the first to second ringing pulse. If the number of pulses is set to 1 then this parameter sets the time interval from the ringing pulse to the next ringing burst.
ontime2	This parameter sets the duration of the second ringing pulse. If the number of pulses is set to less than two then this parameter is ignored.
offtime2	This parameter sets the time interval between the second and third ringing pulse. If the number of ringing pulses is set to less than 2 then this parameter is ignored. If 2 pulses are to be generated then this parameter sets the time from the second pulse to the start of the next ringing burst.
ontime3	This parameter sets the duration of the third ringing pulse. If the number of ringing pulses is set to less than 3 then this parameter is ignored.
offtime3	This parameter sets the time interval between the third ringing pulse and the next ringing burst. If the number of ringing pulses is set to less than 3 then this parameter is ignored.
cycles	This parameter specifies the number of ringing bursts to generate

2.9 Echo Functions

2.9.1 Set_Echo

Description:

The Set_Echo function sets the gain and delay values for a tap in the echo generator. The AI-7280 supports up to three different echos (three taps) which can each have independent gain and delay. Each delay can range from 0 to 25ms and the gains can range from –100 to 100.

Function Prototype:

```
long Set_Echo(long device, long tapindex, float delay, float gain)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
tapindex	The tapindex parameter specifies which tap in the echo generator to modify. This value can range from 1 to 3.
delay	The delay parameter specifies the delay (in milliseconds) for the specified tap in the echo generator. The delay for each tap can range from 0 to 25 milliseconds.
gain	The gain parameter specifies the linear gain (V/V) for the specified tap in the echo generator. The gain for each tap can range from –100 to 100.

2.9.2 Get_Echo

Description:

The Get_Echo function returns the delay and gain settings for a tap in the echo generator.

Function Prototype:

long Get_Echo(long device, long tapindex, float *delay, float *gain)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
tapindex	The tapindex parameter specifies which tap in the echo generator to read from. This value can range from 1 to 3.
delay	The delay parameter returns the delay value (in milliseconds) for the tap in the echo generator. The delay for each tap can range from 0 to 25 milliseconds.
gain	The gain parameter returns the linear gain (V/V) for the tap in the echo generator. The gain for each tap can range from –100 to 100.

2.9.3 Set_EchoRingDisable

Description:

The Set_EchoRingDisable function sets whether echo is automatically disabled during ringing.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

long Set_EchoRingDisable(long deviceid , long Value)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
value	If value is non-zero then the echo generator will be disabled while ringing is active. If value is zero then the echo generator will operate during ringing.

2.9.4 Get_EchoRingDisable

Description:

The Get_EchoRingDisable function returns a value indicating whether echo is automatically disabled during ringing.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

```
long Get_EchoRingDisable (long deviceid , long *Value)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
value	If value is non-zero then the echo generator will be disabled while ringing is active. If value is zero then the echo generator will operate during ringing.

2.10 FSK Generator Functions

2.10.1 Set_FSKGen

Description:

The Set_FSKGen function sets the signal level, frequency, and timing settings required by the FSK generator.

Function Prototype:

long Set_FSKGen (long device, float marklevel, float spacelevel, float markfreq, float spacefreq, float marktime, float spacetime)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
marklevel	The marklevel parameter specifies the level (Vrms) of the mark tone for the FSK generator. The marklevel parameter can range from 0 to 4Vrms.
spacelevel	The spacelevel parameter specifies the level (Vrms) of the space tone for the FSK generator. The spacelevel parameter can range from 0 to 4Vrms.
markfreq	The markfreq parameter specifies the frequency (Hz) for the mark tone. This parameter can range from 20 to 10000Hz.
spacefreq	The spacefreq parameter specifies the frequency (Hz) for the space tone. This parameter can range from 20 to 10000Hz.
marktime	The marktime parameter specifies the bit time (in seconds) for the mark bits. This parameter can range from 0.00025 to 1.0 seconds.
spacetime	The spacetime parameter specifies the bit time (in seconds) for the space bits . This parameter can range from 0.00025 to 1.0 seconds.

2.10.2 Clear_FSKGen

Description:

The Clear_FSKGen function clears all the data out of the transmit buffer of the FSK generator.

Function Prototype:

```
long Clear_FSKGen (long deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.10.3 Add_FSKMarkBits

Description:

The Add_FSKMarkBits function appends the specified number of mark (logic 1) bits to the FSK transmit buffer. This function does not modify the FSK checksum value.

Function Prototype:

```
long Add_FSKMarkBits (long deviceid, long numbits)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numbits	The numbits parameter specifies the number of mark bits to add to the FSK transmit buffer.

2.10.4 Add_FSKSpaceBits

Description:

The Add_FSKSpaceBits function appends the specified number of space (logic 0) bits to the FSK transmit buffer. This function does not modify the FSK checksum value.

Function Prototype:

long Add_FSKSpaceBits (long deviceid, long numbits)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numbits	The numbits parameter specifies the number of space bits to add to the FSK transmit buffer.

2.10.5 Add_FSKAltBits

Description:

The Add_FSKAltBits function appends the specified number of alternating mark and space bits to the FSK transmit buffer. The first bit added is always a space bit. This function does not modify the FSK checksum value.

Function Prototype:

long Add_FSKAltBits (long deviceid, long numbits)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numbits	The numbits parameter specifies the number of alternating mark - space bits to add to the FSK transmit buffer. The first bit added is always a space.

2.10.6 Add_FSKByte

Description:

The Add_FSKByte function appends a byte to the FSK transmit buffer. The number of bits added depends on the current parity and stop bits settings. (See the Set_FSKFormat function for more details). This function will update the FSK checksum using the byte value (if the checksum is enabled).

Function Prototype:

```
long Add_FSKByte (long deviceid, long bytevalue)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
bytevalue	The bytevalue parameter specifies the value of the byte to add to the FSK transmit buffer. The number of bits added depends on the current parity and stop bit settings. This value must be between 0 and 255.

2.10.7 Add_FSKString

Description:

The Add_FSKString function appends a string of values to the FSK generator. The number of bits added to the transmit buffer for each character depends on the current parity and stop bits setting. This function will update the FSK checksum with each character value (if the checksum is enabled).

Function Prototype:

```
long Add_FSKString (long deviceid, char fskstring[])
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
fskstring	The fskstring parameter is a NULL terminated string which contains up to 64 characters to be added to the FSK transmit buffer.

2.10.8 Add_FSKHexString

Description:

The Add_FSKHexString function appends a series of hex values to the FSK generator data.

Function Prototype:

long Add_FSKHexString (long deviceid, char hexstring[])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
hexstring	The fskstring parameter is a NULL terminated string which contains up to 64 hex characters to be added to the FSK transmit buffer. The string will be interpreted as 32 byte values (one character per nibble)

2.10.9 Put_FSKBinaryData

Description:

The Put_FSKBinaryData function inserts a specified number of bits into the FSK transmit buffer from an array of 32-bit integer values. This function overwrites any data already in the FSK buffer. Bit values are inserted into the buffer from the LSB to MSB of each 32-bit integer starting at location 0.

Function Prototype:

long Put_FSKBinaryData (long deviceid, long numbits, long data[])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numbits	The numbits parameter specifies the number of bits to insert into the FSK transmit buffer. These bits are extracted from the data array parameter (from LSB to MSB).
data[]	The data[] parameter is an array of 32-bit integer values that must be loaded with the bit values to store in the FSK data transmit buffer. Bits values are loaded from LSB to MSB of each 32-bit word starting at index 0. **Note: Before calling this function insure that data[] is allocated to have at least ceil(numbits/32) locations.**

2.10.10 Add_FSKXSum

Description:

The Add_FSKXSum function appends the current value of the FSK checksum to the FSK transmit buffer.

Function Prototype:

long Set_FSKXSum (long deviceid)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.10.11 Set_FSKFormat

Description:

The Set_FSKFormat function sets the data format parameters for the FSK generator. This function sets the parity and stopbit settings used when adding data to the FSK buffer. This function also sets the checksum calculation type and enables or disables the checksum calculation.

Function Prototype:

long Set_FSKFormat (long deviceid, long parity, long stopbits, long xsumtype, long xsumenable)

Function Parameters:

deviceid The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

parity The parity parameter sets the parity used when adding characters to the FSK transmit buffer. The parity settings are listed in the table below.

parity Value	Parity Setting
0	No Parity – 8 bits per character
1	Odd Parity – 7 bits per characater
2	Even Parity – 7 bits per character

stopbits The stopbits parameter specifies the number of stop bits added after every byte or character value. The stopbits parameter can range from 1 to 100 bits.

xsumtype The xsumtype parameter specifies the checksum calculation method applied when adding bytes or characters to the FSK transmit buffer. The settings for the xsumtype are shown in the table below.

xsumtype Value	Checksum Type
0	Inverted Modulus 256. Conforms to Bellcore and ETSI FSK Caller ID standards
1	16 Bit CRC (x16+x12+x5+1). Conforms to NTT (Japan) FSK Caller ID standards

xsumenable When the xsumenable value is non-zero then the checksum calculation is performed whenever a byte or character is added to the FSK transmit buffer. If this value is set to zero then the checksum calculations are disabled.

2.10.12 Set_FSKXSum

Description:

The Set_FSKXSum function sets the value of the FSK checksum.

Function Prototype:

```
long Set_FSKXSum (long deviceid, long xsumvalue)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
xsumvalue	The xsumvalue parameter sets the current value of the FSK checksum.

2.10.13 Get_FSKXSum

Description:

The Get_FSKXSum function returns the current value of the FSK checksum.

Function Prototype:

```
long Get_FSKXSum (long deviceid, long *xsumvalue)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
xsumvalue	The xsumvalue parameter returns the current value of the FSK checksum.

2.10.14 Start_FSKGen

Description:

The Start_FSKGen function causes the FSK generator to begin transmitting FSK data on the telephone line. The FSK generator can be started in four different modes: single-shot, single shot and hold carrier, continuous, and external modulation. The FSK generator will begin transmitting the FSK starting at the specified bit index and with the starting phase specified. If the wait parameter is non-zero then this function waits for the completion of the FSK transmission before returning control to the calling program (note: this only applies in single shot mode)

Note: The FSK generator uses tone generator A to generate the FSK signals. Calling Start_FSKGen will override any tones or tone patterns generated on tone generator A.

Function Prototype:

long Start_FSKGen (long deviceid, long fskmode, long bitindex, float startphase, long wait)

Function Parameters:

deviceid The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

fskmode The fskmode parameter specifies the mode of operation for the FSK generator. The possible modes are listed in the table below

fskmode	FSK Transmit mode
0	*Single Shot* - The FSK data is transmitted from the starting bit index to the end of the FSK transmit buffer and then the FSK generator is disabled.
1	*Single Shot Hold Carrier* - The FSK data is transmitted from the starting bit index to the end of the FSK transmit buffer. Then the FSK generator is left active holding the tone corresponding to the last bit.
2	*Continuous* – The FSK data is transmitted from the starting index to the end of the FSK transmit buffer. The data is then continuously transmitted from index 0 to the end of the transmit buffer until the generator is stopped using the Stop_FSKGen function.
3	*External Modulation* – The FSK generator is modulated by the digital values present on digital input A. The FSK generator will continue until the Stop_FSKGen function is called.

bitindex This specifies the index of the first bit which will be transmitted. This value can be between 0 (the first bit) and NumBits-1

startphase	The startphase parameter specifies the starting phase of the FSK carrier signal in degrees. This parameter can range from 0 to 360.
wait	If the wait parameter is non-zero, this function waits for the FSK transmission to complete before returning control to the calling program. Note: this parameter is only valid for single-shot mode.

2.10.15 Stop_FSKGen

Description:

The Stop_FSKGen function stops the FSK generator.

Function Prototype:

```
long Stop_FSKGen (long deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.10.16 Set_FSKGenWaveShape

Description:

The Set_FSKGenWaveShape function sets the wave shape for the tones used by the FSK generator. The wave shape can be selected as sine wave, triangle wave, square wave, or a user defined wave shape.

Function Prototype:

```
long Get_FSKXSum (long deviceid, long shape)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
shape	The shape parameter sets the wave shape used by the FSK generator. The possible values are listed in the table below.

shape	FSK Generator Waveshape
0	Sine wave
1	Triangle wave
2	Square wave
3	User defined shape. This shape can be set using the Load_UserWaveShape function

2.10.17 Get_FSKGenStatus

Description:

The Get_FSKGenStatus function indicates if the FSK generator is active and returns the index of the bit currently being transmitted.

Function Prototype:

long Get_FSKGenStatus (long deviceid, long *isactive, long *bitindex)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
isactive	The isactive property will be set to a non-zero value if the FSK generator is active
bitindex	The bitindex property returns the index of the current bit being transmitted by the FSK generator.

2.10.18 Clear_FSKDropOut

Description:

The Clear_FSKDropOut function clears the contents of the FSK gain adjustment table. For more information on how this table is used to dynamically adjust the FSK levels, see the Set_FSKDropout function. This function cannot be called while the FSK generator is active – doing so will return an error code.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

long Clear_FSKDropOut (long deviceid)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.10.19 Set_FSKDropout

Description:

The Set_FSKDropout function is used to specify dynamic signal level changes to the FSK generator. Changes in FSK signal level can be used to simulate signal drop-outs or gain hits.

The AI-7280 can adjust the mark & space signal level up to four times during a FSK transmission. This is implemented through a table containing a column of bit index values and a column of gain values. When the FSK generator has finished transmitting the bit specified in the bit index column, it adjusts the mark & space level by the amount specified in the gain column.

For example, the following table uses all four possible rows to create a FSK level drop of 60 dB (0.001) after the 200th bit for a duration of 20 bits. After the 220th bit, the FSK level is increased by 60 dB (1000), which restores the original level. In a similar fashion, rows 3 and 4 in the table create a level drop of 20 dB starting after the 300th bit, for a duration of 10 bits.

	bit index	gain
1:	200	0.001
2:	220	1000
3:	300	0.1
4:	310	10

It is important that the bit index column contains values of increasing value (as shown in the above example).

The Set_FSKDropout function is used to modify the bit index and gain values stored in the table. Before using Set_FSKDropout, call Clear_FSKDropout once in order to reset the table structure. Once the table has been reset the Set_FSKDropout function is used to modify the table data. Do not call this function while the FSK generator is active – doing so will return an error code.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

```
long Set_FSKDropout (long deviceid, long index , long bitindex , float gain )
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
index	The index value specifies which entry in the FSK dropout table to modify. The index value must be between 1 and 4.
bitindex	The bitindex value specifies the index of the bit before the gain adjustment is to be made. For example to modify the gain starting

	at bit 4 you would set bitindex = 3. This value can range from 1 to 32767.
gain	The gain parameter specifies the gain adjustment to make on all bits after the bit specified by bitindex. This value can range from 0.0001 to 10000.0

2.10.20 Get_FSKDropout

Description:

The Get_FSKDropout function returns the BitIndex and Gain value for one entry in the FSK Dropout structure.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

long Get_FSKDropout (long deviceid, long index , long *bitindex , float *gain)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
index	The index parameter specifies which entry in the FSK dropout table to modify. The index value must be between 1 and 4.
bitindex	The bitindex parameter returns the index of the bit before the gain adjustment is to be made. For example to modify the gain starting at bit 4 you would set bitindex = 3. This value can range from 1 to 32767.
gain	The gain parameter returns the gain adjustment to make on all bits after the bit specified by bitindex. This value can range from 0.0001 to 10000.0

2.11 DTMF/MF Generator Functions

2.11.1 Set_DTMF

Description:

The Set_DTMF function sets the level and timing settings for the DTMF generator. This lolevel and hilevel parameters set the voltage (in Vrms) for the low frequency group and high frequency group respectively. The ontime parameter sets the duration (in milliseconds) of each DTMF digit, and the offtime parameter sets the inter-digit interval (in milliseconds). This function must be called before sending DTMF digits.

Note: the DTMF generator and MF generator use the same resources on the AI-7280. This function resets the first 16 symbols in the MF generator to the DTMF frequencies and the specified levels and on-times. After this function call the MF symbols “A” to “P” will be reset to DTMF digits.

Function Prototype:

```
long Set_DTMF (long deviceid, float lolevel, float hilevel, float freqadjust,
float ontime, float offtime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
lolevel	The lolevel parameter specifies the level (in Vrms) for the low frequency DTMF tones. The lolevel parameter can range from 0 to 4 Vrms.
hilevel	The hilevel parameter specifies the level (in Vrms) for the high frequency DTMF tones. The hilevel parameter can range from 0 to 4 Vrms.
freqadjust	The freqadjust parameter sets the frequency adjustment (in percent) for all DTMF digits. Setting the freqajust parameter to a non-zero value will increase or decrease all the DTMF frequencies by the specified percentage. The freqadjust parameter can range from –20 to +20%.

2.11.2 Send_DTMF

Description:

The Send_DTMF function transmits a sequence of up to 64 DTMF digits on the telephone line. The digits parameter is a string of DTMF digits to transmit. The wait parameter indicates if this function should wait for the DTMF generator to complete before returning control to the calling program.

The DTMF generator shares resources with the MF generator and also uses tone generators C and D. The Send_DTMF function will interrupt any tones being generated using tone generators C or D. The DTMF generator and the MF generator must both be inactive before calling this function.

Function Prototype:

```
long Send_DTMF (long deviceid, char symbols[], long wait)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
digits	The digits parameter should be a NULL terminated string of DTMF symbols to transmit on the telephone line. Valid DTMF digits are represented by the characters "0" to "9" and "A" to "D" and the characters "*" and "#". If the string contains any other characters this function will return an error. The length of this string must not exceed 64 characters.
wait	If the wait parameter is non-zero then this function waits for the DTMF generator to finish transmitting the digits before returning control to the calling program. Otherwise, control is returned immediately.

2.11.3 Stop_DTMF

Description:

The Stop_DTMF function immediately stops the DTMF generator.

Function Prototype:

```
long Stop_DTMF (long deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.11.4 Get_DTMFStatus

Description:

The Get_DTMFStatus function indicates if the DTMF generator is active (generating digits).

Function Prototype:

```
long Get_DTMFStatus (long deviceid, long *status)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
status	The status value is returned with a non-zero value if the DTMF generator is active (generating digits). Otherwise the status value is set to zero.

2.11.5 Set_MFSymbol

Description:

The Set_MFSymbol function sets the level, frequency, and duration for a single MF symbol in the MF generator. Up to 20 different MF symbols can be defined corresponding to the characters “A” through “T”. Each MF symbol can be defined with two tones having independent levels, frequencies, and duration. Note: the symbol characters are not case sensitive.

Note: the DTMF generator and MF generator use the same resources on the AI-7280. Changing the first 16 symbols of the MF generator “A” through “P” modify the signal parameters for the DTMF generator. Before transmitting DTMF symbols the DTMF signal parameters should be restored by calling the Set_DTMF function.

Function Prototype:

```
long Set_MFSymbol(long deviceid, char symbol[], float level1, float level2,
float freq1, float *freq2, float ontime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
symbol	The symbol parameter is a NULL terminated string containing at least one valid MF symbol in the range “A” through “T” (not case sensitive). This function sets the signal parameters for the first character in the symbol string.

level1	The level1 parameter specifies the level (in Vrms) for the first in the MF symbol. This parameter can range from 0 to 4Vrms.
level2	The level2 parameter specifies the level (in Vrms) for the second tone in the MF symbol. This parameter can range from 0 to 4Vrms.
freq1	The freq1 parameter specifies the frequency (Hz) for the first tone in the MF symbol. This parameter can range from 20 to 10000 Hz.
freq2	The freq2 parameter specifies the frequency (Hz) for the second tone in the MF symbol. This parameter can range from 20 to 10000 Hz.
ontime	The ontime parameter specifies the duration of the MF symbol in milliseconds. This parameter must be greater than zero.

2.11.6 Set_MFOffTime

Description:

The Set_MFOffTime function sets the time interval between consecutive MF symbols.

Function Prototype:

```
long Set_MFOffTime (long deviceid, float offtime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
offtime	The offtime parameter specifies the time interval between MF symbols (in milliseconds). This parameter must be greater than zero.

2.11.7 Send_MF

Description:

The Send_MF function transmits a sequence of up to 64 MF symbols on the telephone line. The symbols parameter is a string of MF symbols to transmit. The levels, frequencies, and durations for these symbols can be set using the Set_MFSymbol function. The wait parameter indicates if this function should wait for the MF generator to complete before returning control to the calling program.

The MF generator shares resources with the DTMF generator and also uses tone generators C and D. The Send_MF function will interrupt any tones being generated using tone generators C or D. The MF generator and the DTMF generator must both be inactive before calling this function.

Function Prototype:

```
long Send_MF (long deviceid, char symbols[], long wait)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
symbols	The symbols parameter should be a NULL terminated string of MF symbols to transmit on the telephone line. MF symbols consist of the digits “A” through “T” (not case sensitive). If the string contains any other characters this function will return an error. The length of this string must not exceed 64 characters.
wait	If the wait parameter is non-zero then this function waits for the MF generator to finish transmitting the digits before returning control to the calling program. Otherwise, control is returned immediately.

2.11.8 Stop_MF

Description:

The Stop_MF function stops the MF generator.

Function Prototype:

```
long Stop_MF (long deviceid, float offtime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.11.9 Get_MFStatus

Description:

The Get_MFStatus function indicates if the MF generator is active (generating symbols).

Function Prototype:

```
long Get_MFStatus (long deviceid, long *status)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
status	The status value returns a non-zero value if the MF generator is currently active (generating symbols). Otherwise the status parameter returns zero.

2.12 FSK Decoder Functions

2.12.1 Clear_FSKDecoder

Description:

The Clear_FSKDecoder function resets the FSK decoder to receive new data. This function sets the internal write pointer to the beginning of the receive buffer so the next byte decoded will be stored at index 1. This function should be called when the FSK decoder is disabled.

Function Prototype:

long Clear_FSKDecoder (long device)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.12.2 Set_FSKDecThreshold

Description:

The Set_FSKDecThreshold function sets the minimum voltage threshold for the FSK decoder (in volts-peak). Any FSK signals with peak amplitude lower than this voltage will not be detected.

Function Prototype:

long Set_FSKDecThreshold (long deviceid, float threshold)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
threshold	The threshold parameter specifies the minimum level (in volts peak) required for an FSK signal to be decoded.

2.12.3 Get_FSKDecMarkTime

Description:

The Get_FSKDecMarkTime function returns the number of seconds that a valid FSK mark signal has been detected on the telephone line.

Function Prototype:

```
long Get_FSKDecMarkTime (long deviceid, float *marktime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
marktime	The marktime parameter returns the number of seconds that a valid FSK mark signal has been detected on the telephone line.

2.12.4 Start_FSKDecoder

Description:

The Start_FSKDecoder function enables the FSK decoder on the AI-7280. After this function call the FSK decoder will decode any incoming FSK bytes and store them in the FSK decoder buffer. These bytes can be read using the Get_FSKDecByte function.

Function Prototype:

```
long Start_FSKDecoder (long deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.12.5 Stop_FSKDecoder

Description:

The Stop_FSKDecoder function disables the FSK decoder on the AI-7280.

Function Prototype:

```
long Stop_FSKDecoder (long deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.12.6 Get_FSKDecNumBytes

Description:

The Get_FSKDecNumBytes function returns the number of decoded bytes in the FSK decoder's receive buffer. These bytes can be extracted using the Get_FSKDecByte function.

Function Prototype:

```
long Get_FSKDecNumBytes (long deviceid, long *numbytes)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numbytes	The numbytes parameter returns the number of decoded FSK bytes in the FSK decoder receive buffer.

2.12.7 Get_FSKDecByte

Description:

The Get_FSKDecByte function returns a byte from the receive buffer of the FSK decoder. A status value is also returned with each byte indicating any errors that were detected while the byte was being decoded.

Function Prototype:

long Get_FSKDecByte (long deviceid, long index, long *byteval, long *bytestatus)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
index	The index parameter specifies which byte in the FSK decoder buffer to return. This index value can range from 1 to 2047.
byteval	The byteval parameter returns the value of the byte in the FSK decoder receive buffer at the location indicated by the index parameter.
bytestatus	The bytestatus parameter returns the status value for the byte being returned. The following table describes the meaning of the bits in the status value.

bytestatus bit	Meaning
bit 0	*No Stop Bit*. If this bit is set it indicates that the FSK byte was decoded with an incorrect stop bit value
bit 1	*Level Drop*. If this bit is set it indicates that the FSK level dropped below the minimum threshold while decoding the byte.

2.13 DTMF Detector Functions

2.13.1 Set_DTMFDet

Description:

The Set_DTMFDet function sets the signal detection parameters for the DTMF detector. The freqtoll property sets the frequency tolerance (in percent) for the DTMF detector. The minlevel property sets the minimum required level for a DTMF digit to be detected.

Function Prototype:

long Set_DTMFDet (long deviceid, float freqtoll, float minlevel)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
freqtoll	The freqtoll parameter sets the frequency tolerance (in percent). Any digits with frequencies that deviate from the nominal DTMF frequencies by more that this percentage will be ignored. The freqtoll parameter can range from 0 to 2 percent.
minlevel	The minlevel parameter sets the minimum level (Vrms) required for a DTMF digit to be detected. This parameter must be greater than zero.

2.13.2 Start_DTMFDet

Description:

The Start_DTMFDet function enables the DTMF detector. Once this function is called the DTMF detector will begin detecting DTMF digits on the telephone line. All detected digits will be stored with levels, frequencies, and timing information. The digits can be read using the Get_DTMFDetDigit function.

Function Prototype:

long Start_DTMFDet (long deviceid)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.13.3 Stop_DTMFDet

Description:

The Stop_DTMFDet function stops the DTMF detector. No further DTMF level or frequency measurements will be made and the current DTMF measurements will be frozen.

Function Prototype:

long Stop_DTMFDet (long deviceid)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.13.4 Wait_For_DTMF

Description:

The Wait_For_DTMF function waits for a DTMF digit to be detected before returning control to the calling program. A maximum wait time must be specified to prevent waiting forever. This maximum time can range from 0 to 100000 ms. This function avoids time consuming polling loops when a DTMF digit is expected within a fixed time interval. **Note: for this function to work correctly the DTMF detector must be started and there must be less than 31 DTMF digits recorded by the DTMF detector.** This function does not remove the digit from the detector and the details of the digit can be extracted using the Get_DTMFDetDigit function.

Function Prototype:

long Wait_For_DTMF (long deviceid, long maxtime, long *digitcode)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
maxtime	The maxtime parameter specifies the number of milliseconds to wait for a new DTMF digit to be detected. This value can range from 0 to 100,000 ms.
digitcode	The digitcode parameter returns the digit code of a detected DTMF digit. If no DTMF digits are detected then this value is set to zero.

2.13.5 Get_DTMFDet

Description:

The Get_DTMFDet function returns the current level and frequency measurements from the level meters and frequency counters in the DTMF detector. It also returns a digit code of any DTMF digit detected.

Function Prototype:

long Get_DTMFDet (long deviceid, float *lofreq, float *lolevel, float *hifreq, float *hilevel, long *digit)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
lofreq	The lofreq parameter returns the current frequency measurement (Hz) of the signals in the low frequency DTMF band.
lolevel	The lolevel parameter returns the current level measurement (Vrms) of the signals in the low frequency DTMF band.
hifreq	The hifreq parameter returns the current frequency measurement (Hz) of the signals in the high frequency DTMF band.
hilevel	The hilevel parameter returns the current level measurement (Vrms) of the signals in the high frequency DTMF band.
digit	The digit parameter returns a digit code of the DTMF digit that corresponds to the current level and frequency measurements. The digit codes are listed in the following table

digit code	DTMF Digit
0	No digit
1- 9	1 – 9
10	0
11	*
12	#
13-16	A-D

2.13.6 Get_DTMFDetNumDigits

Description:

Whenever a DTMF digit is detected, the level, frequency, and timing information is stored in a FIFO for retrieval and analysis at a later time. The Get_DTMFDetNumDigits function returns the number of DTMF digit records stored in the DTMF detector. The digit information can be read out using the Get_DTMFDetDigit function.

Function Prototype:

long Get_DTMFDetNumDigits (long deviceid, long *numdigits)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numdigits	This parameter returns the number of DTMF digit records stored in the DTMF detector.

2.13.7 Get_DTMFDetDigit

Description:

The Get_DTMFDetDigit function returns the level, frequency, and timing information from a digit record in the DTMF detector FIFO. The index specifies which DTMF digit record to read (from 0 to numdigits –1).

Function Prototype:

long Get_DTMFDetDigit (long deviceid, long index, float *lofreq, float *lolevel, float *hifreq, float *hilevel, float *starttime, float *stoptime, long *digit)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
index	The index parameter specifies which DTMF digit record in the FIFO to read. The total number of records available can be returned using the Get_DTMFDetNumDigits function. This index can range from 0 to numdigits – 1. The record at index 0 is the least recent record while the record at index numdigits-1 is the most recent record.
lofreq	The lofreq parameter returns the frequency measurement (Hz) for

the low frequency tone of the recorded DTMF digit.

lolevel	The lolevel parameter returns the level measurement (Vrms) for the low frequency tone of the recorded DTMF digit.
hifreq	The hifreq parameter returns the frequency measurement (Hz) for the high frequency tone of the recorded DTMF digit.
hilevel	The hilevel parameter returns the level measurement (Vrms) for the high frequency tone of the recorded DTMF digit.
starttime	The starttime parameter returns the time (in seconds) of the beginning of the DTMF digit. Note: this value is based on the timer internal to the AI-7280.
stoptime	The stoptime parameter returns the time (in seconds) of the end of the DTMF digit. Note: this value is based on the timer internal to the AI-7280.
digit	The digit parameter returns a digit code that corresponds to the DTMF digit that was detected. The digit codes are listed in the following table.

digit code	DTMF Digit
0	No digit
1- 9	1 – 9
10	0
11	*
12	#
13-16	A-D

2.13.8 Delete_DTMFDetDigits

Description:

This function deletes a specified number of DTMF records from the DTMF detector FIFO. Records are always deleted from the top of the FIFO (low index) and new digits are always added at the end of the FIFO (high index) to prevent accidental deletion of newly received digits.

Function Prototype:

long Delete_DTMFDetDigits (long deviceid, long numdigits)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numdigits	This parameter specifies the number of DTMF digit records to delete from the DTMF detector. Records are always deleted from the top of the FIFO (starting at index 0).

2.14 Capture and Playback Functions

2.14.1 Start_ACCap

Description:

The Start_ACCap function starts recording the AC signals on the telephone line into memory with a resolution of 16 bits per sample. The capture can be started in two modes (corresponding to two different sampling rates) and record up to 229376 samples. The samples are stored sequentially in a buffer in the AI-7280 and can be read out using the Get_ACCapSamples function. If the end of the capture buffer is reached during a capture then the write index will roll-over and samples will continue to be stored starting at location 0

Function Prototype:

long Start_ACCap (long deviceid, long mode, long startindex, long numsamples)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
mode	The mode parameter sets the sample rate for the AC capture. The two modes are listed in the table below.

mode	Sample Rate of Capture
0	39062 Hz
1	19531 Hz

startindex	The startindex parameter sets the index in the AC capture buffer to start recording. This index can range from 0 to 229375.
numsamples	The numsamples parameter sets the number of samples to record. This parameter is normally set to a value between 1 and 229376. However, if this value is set to –1 then the samples will be recorded continuously until the Stop_ACCap function is called.

2.14.2 Stop_ACCap

Description:

The Stop_ACCap function stops an AC capture in progress.

Function Prototype:

```
long Stop_ACCap (long deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.14.3 Get_ACCapStatus

Description:

The Get_ACCapStatus function returns the number of samples remaining in an AC capture. If the number of samples left is 0 then the AC capture has completed. If the number of samples returned is negative, then the AC capture is continuing indefinitely until it is stopped (by calling Stop_ACCap).

Function Prototype:

```
long Get_ACCapStatus (long deviceid, long *samplesleft)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
samplesleft	The samplesleft property returns the number of samples remaining in an active AC capture. If the samplesleft property is zero then the AC capture is completed. If the samplesleft property is negative then this implies that the capture is continuing indefinitely until it is stopped using the Stop_ACCap function

2.14.4 Get_ACCapSamples

Description:

The Get_ACCapSamples function reads a block of voltage samples from the AC capture memory starting at a specified index and returns the values of the samples in the samples[] array.

Function Prototype:

long Get_ACCapSamples (long deviceid, long startindex, long numsamples, float samples[])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
startindex	The startindex property specifies the starting index of the block of samples to read from the AC capture buffer. This index can range from 0 to 229375.
numsamples	The numsamples property specifies the number of samples to be read from the AC capture buffer. This value can range from 1 to 229376.
samples[]	The samples[] property is an array of floating point values that will return the values of the samples in the AC capture buffer. The samples will be stored sequentially from location 0 to numsamples – 1. **Note: Before calling this function insure that the samples[] array has been dimensioned to have at least numsamples elements.**

2.14.5 Put_ACCapSamples

Description:

The Get_ACCapSamples function transfers the array of voltage samples (in samples []) into the AC capture buffer. The numsamples property specifies the number of samples to be transferred and the startindex specifies the starting location in the AC capture buffer to store the samples.

Function Prototype:

long Put_ACCapSamples (long deviceid, long startindex, long numsamples, float samples[])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
startindex	The startindex property specifies the starting index in the AC capture buffer to begin storing samples at. This index can range from 0 to 229375.
numsamples	The numsamples property specifies the number of samples to be transferred into the AC capture buffer. This value can range from 1 to 229376.
samples[]	The samples[] property is an array of floating point values that hold the values of the AC samples to be transferred into the AC capture buffer. The samples will be read sequentially from location 0 to numsamples – 1. **Note: Before calling this function insure that the samples[] array has been dimensioned to have at least numsamples elements.**

2.14.6 Get_ACCapIndex

Description:

The Get_ACCapIndex function returns the location in the AC Capture buffer where the next sample will be stored. This index value can be used to determine the position of a captured AC signal in memory when the indefinite capture mode is used.

Function Prototype:

long Get_ACCapIndex (long deviceid, long *index)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
index	The index property returns the index into the AC capture buffer where the next sample will be stored.

2.14.7 Play_ACCap

Description:

The Play_ACCap function begins playing back a sequence of samples from the AC capture buffer onto the telephone line. Note: The samples will be played back on the telephone line at a sample rate of 39062Hz.

Function Prototype:

long Play_ACCap (long deviceid, long startindex, long numsamples)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
startindex	The startindex parameter specifies the starting index of the samples to be played back on the telephone line
numsamples	The numsamples parameter specifies the number of samples to be played back on the telephone line. Normally this parameter is between the values 1 and 229376, however, if this value is set to –1 then the playback continues indefinitely until the Stop_Playback function is called.

2.14.8 Get_PlaybackStatus

Description:

The Get_PlaybackStatus function returns the number of samples remaining in an AC capture playback. If the value returned is –1 then this indicates that the playback is continuing indefinitely.

Function Prototype:

long Get_PlaybackStatus (long deviceid, long *samplesleft)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
samplesleft	The samplesleft returns the number of samples remaining in the AC playback. If the returned value is –1 then the playback is continuing indefinitely.

2.14.9 Stop_Playback

Description:

The Stop_Playback function stops the playback of an AC capture.

Function Prototype:

long Stop_Playback (long deviceid)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.14.10 Start_DCCap

Description:

The Start_DCCap function starts sampling and recording the DC voltage and loop-current into memory. The voltage and current sample are stored with 12-bit resolution at a sample rate of 1000Hz. The DC capture buffer can hold a maximum of 8192 voltage and current samples. If the end of the capture buffer is reached during a capture then the write index will roll-over and samples will continue to be stored starting at location 0.

Function Prototype:

long Start_DCCap (long deviceid, long mode, long startindex, long numsamples)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
startindex	The startindex parameter sets the index in the DC capture buffer to start recording. This index can range from 0 to 8191.
numsamples	The numsamples parameter sets the number of samples to record. This parameter is normally set to a value between 1 and 8192. However, if this value is set to –1 then the samples will be recorded continuously until the Stop_DCCap function is called.

2.14.11 Stop_DCCap

Description:

The Stop_DCCap function stops a DC capture in progress.

Function Prototype:

long Stop_DCCap (long deviceid)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.14.12 Get_DCCapStatus

Description:

The Get_DCCapStatus function returns the number of samples remaining in a DC capture. If the number of samples left is 0 then the DC capture has completed. If the number of samples returned is negative, then the DC capture is continuing indefinitely until it is stopped using the Stop_DCCap function.

Function Prototype:

```
long Get_DCCapStatus (long deviceid, long *samplesleft)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
samplesleft	The samplesleft property returns the number of samples remaining in an active DC capture. If the samplesleft property is zero then the DC capture is completed. If the samplesleft property is negative then this implies that the capture is continuing indefinitely until it is stopped using the Stop_DCCap function

2.14.13 Get_DCCapSamples

Description:

The Get_DCCapSamples function reads a block of voltage and current samples from the DC capture memory starting at a specified index and returns the values of the samples in the voltage[] and current[] arrays.

Function Prototype:

long Get_DCCapSamples (long deviceid, long startindex, long numsamples, float voltage[], float current[])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
startindex	The startindex property specifies the starting index of the block of voltage and current samples to read from the DC capture buffer. This index can range from 0 to 8191.
numsamples	The numsamples property specifies the number of voltage and current samples to be read from the DC capture buffer. This value can range from 1 to 8192.
voltage[]	The voltage [] property is an array of floating point values that will return the values of the line voltage samples in the DC capture buffer. The samples will be stored sequentially from location 0 to numsamples – 1. **Note: Before calling this function insure that the voltage [] array has been dimensioned to have at least numsamples elements.**
current[]	The current[] property is an array of floating point values that will return the values of the loop current samples in the DC capture buffer. The samples will be stored sequentially from location 0 to numsamples – 1. **Note: Before calling this function insure that the voltage [] array has been dimensioned to have at least numsamples elements.**

2.14.14 Get_DCCapIndex

Description:

The Get_DCCapIndex function returns the location in the DC Capture buffer where the next sample will be stored. This index value can be used to calculate the location of captured DC signals in memory when the indefinite capture mode is used.

Function Prototype:

```
long Get_DCCapIndex (long deviceid, long *index)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
index	The index property returns the index into the DC capture buffer where the next sample will be stored.

2.15 Digital I/O Functions

2.15.1 Get_Din

Description:

The Get_Din function gets the logic level of the two digital inputs on the AI-7280.

Function Prototype:

```
long Get_Din(long device, long * dinA, long *dinB)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
dinA	The dinA parameter returns the logic level (0 or 1) of digital input A.
dinB	The dinB parameter returns the logic level (0 or 1) of digital input B.

2.15.2 Set_DoutA

Description:

The Set_DoutA function sets the logic level or special function mode for the digital output A. If the value parameter is set to 0 or 1 the logic level of digital output A is set to the corresponding logic level. If value is set to 2 then the logic level of output A will track the hook switch status of the AI-7280.

Function Prototype:

long Set_DoutA(long device, long value)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
value	The value parameter specifies the output logic level or special function of digital output A.

value	Digital Output A
0	Logic 0 (0V)
1	Logic 1 (5V)
2	Hook Status. Off hook = Logic 1 On hook = Logic 0

2.15.3 Set_DoutB

Description:

The Set_DoutB function sets the logic level or special function mode for the digital output B. If the value parameter is set to 0 or 1 the logic level of digital output B is set to the corresponding logic level. If value is set to 2 then the logic level of output B will track the current FSK decoder bit value.

Function Prototype:

long Set_DoutB (long device, long value)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
value	The value parameter specifies the output logic level or special function of digital output B.

value	Digital Output B

0	Logic 0 (0V)
1	Logic 1 (5V)

2	FSK decoder bit value
	Mark = logic 1
	Space = logic 0

2.15.4 Set_DoutC

Description:

The Set_DoutC function sets the logic level for the digital output C. If the value parameter is set to 0 or 1 the logic level of digital output C is set to the corresponding logic level.

Function Prototype:

long Set_DoutC (long device, long value)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
value	The value parameter specifies the output logic level of digital output C.

value	Digital Output C
0	Logic 0 (0V)
1	Logic 1 (5V)

2.16 Noise Generator Functions

2.16.1 Set_Noise

Description:

The Set_Noise function sets the level of the white noise generator in Vrms.

Function Prototype:

```
long Set_Noise(long device, float noiselevel)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
noiselevel	The noise level property sets the level (in Vrms) of the noise generator.

2.16.2 Start_Noise

Description:

The Start_Noise function starts the white noise generator on the AI-7280. The voltage level of the noise generator can be set using the function Set_Noise.

Function Prototype:

```
long Start_Noise(long device)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.16.3 Stop_Noise

Description:

The Stop_Noise function stops the white noise generator on the AI-7280.

Function Prototype:

```
long Stop_Noise(long device)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.17 Metering Pulse Functions

2.17.1 Set_MeterPulse

Description:

The Set_MeterPulse function sets the parameters for the meter pulse generator.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

```
long  Set_MeterPulse (long deviceid, float freq, float level, float duration ,
          float repeat)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
freq	The freq parameter specifies the frequency (Hz) for the metering pulses. This value can range from 20 to 18000Hz.
level	The level parameter specifies the voltage level (Vrms) for the metering pulses. This value can range from 0 to 4Vrms.
duration	The duration parameter specifies the duration (ms) of the metering pulses. This value can range from 1 to 1000000 ms.
repeat	The repeat parameter specifies the time interval (ms) between metering pulses. This value can range from 1 to 1000000 ms

2.17.2 Get_MeterPulse

Description:

The Get_MeterPulse function returns the parameters for the meter pulse generator.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

long Get_MeterPulse (long deviceid, float *freq, float *level, float *duration , float *repeat)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
freq	The freq parameter returns the frequency (Hz) for the metering pulses.
level	The level parameter returns the voltage level (Vrms) for the metering pulses.
duration	The duration parameter returns the duration (ms) of the metering pulses.
repeat	The repeat parameter returns the time interval (ms) between metering pulses.

2.17.3 Start_MeterPulse

Description:

The Start_MeterPulse function starts the meter pulse generator.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

long Start_MeterPulse (long deviceid)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.17.4 Stop_MeterPulse

Description:

The Stop_MeterPulse function stops the meter pulse generator.

Note: This function is only available if the AI-7280 is running software version 2.0 or greater!

Function Prototype:

```
long  Stop_MeterPulse (long deviceid)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

2.17.5 Start_MeterPulseWithCount

Description:

This function causes the AI-7280 to generate a specific number of metering pulses

Note: This function is only available if the AI-7280 is running software version 2.12 or greater!

Function Prototype:

```
long Start_MeterPulseWithCount (long Deviceid, long Count)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
count	This value should be set to the number of pulses to generate. A negative value will cause metering pulses to be generated indefinitely.

2.17.6 Get_MeterPulseCount

Description:

This function returns the number of remaining metering pulses to generate

Note: This function is only available if the AI-7280 is running software version 2.12 or greater!

Function Prototype:

long Get_MeterPulseCount (long deviceid, long *Count)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
count	This value retursns the number of pulses remaining to generate. A negative value will cause metering pulses to be generated indefinitely.

2.18 Miscellaneous Functions

2.18.1 Load_UserWaveShape

Description:

The Load_UserWaveShape function allows the user to upload a custom wave shape into the AI-7280. This user-defined wave shape can then be used as a tone shape or as a ringing shape.

Function Prototype:

long Load_UserWaveShape(long deviceid, long numsamples, float samples[])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
numsamples	The numsamples parameter specifies the number of samples in the user defined waveshape. The number of samples can range from 2 to 256.
samples	The samples parameter is a pointer to an array of 32-bit floating point values that contain the samples for the user defined wave shape. The array must contain the number of samples specified by the numsamples parameter.

2.19 Script and Command Functions

2.19.1 Send_TextCommand

Description:

The Send_TextCommand function transmits a text command to the AI-7280 and returns the response from the device. *The commands are formatted specifically for the AI-7280 and are not documented in this document; This function has been included for custom application requirements please contact technical support for more information on this function.*

Function Prototype:

long Send_TextCommand (long deviceid, char command[64], long expectresponse, char response[64])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
command	This parameter should contain the specially formatted command to send to the AI-7280. Please contact technical support for information regarding command formatting.
expectresponse	If non-zero, this function will wait for and return a response from the device. Otherwise this function will terminate without waiting for a response. Caution: most commands do require a response and setting this value to 0 can cause errors.
response	This will return the response from the device. **Note: Before calling this function insure that this string is allocated to be at least 300 characters long. See** Data Type Summary **for initialization details.**

2.19.2 Get_ScriptMemAvail

Description:

The Get_ScriptMemAvail function returns the maximum length of a user composed script that can be loaded into the AI-7280. This value only reflects the maximum length allowed by the DLL implementation; this function doesn't reflect the number of bytes remaining in memory!

Function Prototype:

long Get_ScriptMemAvail (long *maxbytes)

Function Parameters:

maxbytes	This parameter returns the maximum length of a user defined script that can be loaded into the AI-7280 in this version of the DLL.

2.19.3 Get_FirstAvailGlobalReg

Description:

The Get_FirstAvailGlobalReg function returns the first global data pool location not allocated to the DLL internal functions. User scripts should not access any global registers below this value as it could cause unpredictable results.

Function Prototype:

```
long Get_FirstAvailGlobalReg  (long *firstlocation)
```

Function Parameters:

firstlocation	This parameter returns the first global data pool location not used by the DLL. No user scripts should access any global registers below this location.

2.19.4 Run_UserScript

Description:

The Run_UserScript function loads and runs a user compiled script onteh AI-7280. The script must be first compiled using AI-Workbench. Please contact technical support for more information on the specifics of the scripting language and features.

Important notes:

- Script lengths should not exceed the length returned by Get_ScriptMemAvail
- Calling this function will stop any currently executing user scripts
- User scripts should never access global data locations below the value returned by Get_FirstAvailGlobalReg as this may cause unpredictable results

Function Prototype:

```
long Run_UserScript (long deviceid, char userscript[ ])
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
userscript	The userscript parameter should contain the compiled object code generated by AI-Workbench.

2.19.5 Control_Script

Description:

The Control_Script function will stop, halt, resume, or single stop a user script loaded in the AI-7280.

Function Prototype:

```
long Control_Script (long deviceid, long  action )
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
action	This value determines the action performed on the script processor. The allowed values are: 0 = stop (cannot be resumed) 1 = halt (can be resumed) 2 = resume 3 = single step

2.19.6 Get_ScriptStatus

Description:

The Get_ScriptStatus function returns the current status of the user script executing on the AI-7280.

Function Prototype:

```
long Get_ScriptStatus (long deviceid, long * scriptstatus)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
scriptstatus	This value indicates the current status of the user script on the AI-7280. The normal values are 0 = stopped (cannot be resumed) 1 = running 2 = halted (can be resumed) 3 = single step mode 4 = waiting for interrupt If any other value is returned it indicates that an error has occurred while executing the script. Please contact technical support if you encounter such and error code.

2.19.7 Get_ScriptVariable

Description:

The Get_ScriptVariable function returns the value of a variable in the AI-7280's memory space. Values can be fetched from the script's local data pool, the global data pool, or from a processor register. Please contact technical support for more information on the memory structure and usage.

Function Prototype:

long Get_ScriptVariable (long deviceid, long varnum, long variabletype, long isstring, char value[64])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
varnum	This value indicates the variable location in the selected data pool to access
variabletype	This value specifies which data pool to read the variable from. The allowed values are: 0 = Script processor data pool 1 = Global data pool 2 = Processor Register
isstring	If non-zero then the location is read as a string
value	The value parameter returns the value at the specified location in string format.**This string must be initialized to at least 200 characters before calling this function. See** Data Type Summary **for initialization details.**

2.19.8 Put_ScriptVariable

Description:

The Put_ScriptVariable function sets the value of a variable in the AI-7280 script memory space. Values can be fetched from the script's local data pool, the global data pool, or from a processor register. Please contact technical support for more information on the memory structure and usage.

Note: Global Data Pool locations cannot be accessed below the value returned by Get_FirstAvailGlobalReg!

Function Prototype:

long Put_ScriptVariable (long deviceid, long varnum, long variabletype, long isstring, char value[64])

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
varnum	This value indicates the variable location in the selected data pool to access
variabletype	This value specifies which data pool to write the variable to. The allowed values are: 0 = Script processor data pool 1 = Global data pool 2 = Processor Register
isstring	If non-zero then the location is written as a string
value	The value parameter should contain the value to be written (in string form)

2.20 Pulse Dialing Functions

2.20.1 Wait_For_PulseDial

Description:

This function waits for a single pulse dialing sequence to occur within a specific time window and returns the number of pulses detected and basic timing information of the detected sequence. This function reports the number of valid pulses detected up until the first pulse that violates the timing requirements specified (which is usually the end of the dialing sequence). More specific timing information can be extracted using the Get_PulseDial_Stats function.

Function Prototype:

```
long Wait_For_PulseDial (long deviceid, long MaxTime, long MaxBreak,
long MaxMake, long *NumPulses, float *StartTime, float *StopTime)
```

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
MaxTime	This specifies the maximum amount of time (ms) to wait for a pulse dialing sequence to begin.
MaxBreak	This value specifies the maximum length of a break interval (on-hook) during a pulse dialing sequence. A break interval longer than this value will cause the function to return immediately, and report the number of pulses detected up until that time.
MaxMake	This value specifies the maximum length of a make interval (off-hook) during a pulse dialing sequence. A make interval longer than this value will cause the function to return immediately, and report the number of pulses detected up until that time.
NumPulses	This returns the number of valid pulses detected in the pulse dialing sequence. If no pulse dialing was detected this will return 0. If a single pulse violates the maximum timing requirements MaxBreak or MaxMake then this may report fewer pulses than were actually present since the function returns immediately when such a timing violation occurs.
StartTime	If a valid pulse dialing sequence was detected, this will return the time when the first break interval started (first on-hook transition.)
StopTime	If a valid pulse dialing sequence was detected, this will return the time when the last break interval ended (last off-hook transition).

2.20.2 Get_PulseDial_Stats

Description:

This function returns timing statistics for the last pulse dialing sequence detected by the Wait_For_PulseDial function.

Function Prototype:

long Get_PulseDial_Stats (long deviceid, long *NumPulses, float *AvgBreak, float *MaxBreak, float *MinBreak, float *AvgMake, float *MaxMake, float *MinMake)

Function Parameters:

deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
NumPulses	This returns the number of pulses detected in the last call to Wait_For_PulseDial. If this value is 0 then the remaining statistics must be ignored.
AvgBreak	This returns the average duration (seconds) of the break intervals in the pulse dialing sequence.
MaxBreak	This returns the duration (seconds) of the longest break interval in the pulse dialing sequence
MinBreak	This returns the duration (seconds) of the shortest break interval in the pulse dialing sequence.
AvgMake	This returns the average duration (seconds) of the make intervals in the pulse dialing sequence.
MaxMake	This returns the duration (seconds) of the longest make interval in the pulse dialing sequence.
MinMake	This returns the duration (seconds) of the shortest make interval in the pulse dialing sequence.

3 CallerID Functions

The following DLL functions have been added to the AI-7280 DLL to allow the user to easily generate and send CallerID sequences without calling the lower-level AI-7280 functions. Unlike the normal AI-7280 DLL function calls these functions are global and affect all connected AI-7280 devices.

3.1 Physical Layer Function Calls

The physical layer function calls set up the signal levels, frequencies, etc. for the signals used in the CallerID sequences. Note: The settings for the physical layer function calls do not take effect until the CallerID sequence is sent. Since the CallerID functions are global, changing the physical layer settings affects the CallerID sequences for all connected AI-7280s.

3.1.1 CID_Set_FSK

Description:

This function sets the levels, frequencies, and baud rate for the FSK data sent in the CallerID sequence. Note: these settings do not take effect until a CallerID sequence is started on an AI-7280 using the CID_Send function.

Function Prototype:

long CID_Set_FSK (float MarkLevel, float MarkFreq, float SpaceLevel, float SpaceFreq, float Baud)

Function Parameters:

marklevel	This sets the level (Vrms) of the mark tone used to in the FSK data portion of the CallerID sequence
markfreq	This parameter sets the frequency (Hz) of the mark tone used in the FSK data portion of the CallerID sequence
spacelevel	This parameter sets the level (Vrms) of the space tone used in the FSK data portion of the CallerID sequence
spacefreq	This parameter sets the frequency (Hz) of the space tone used in the FSK data portion of the CallerID sequence
baud	This parameter sets the baud rate used in the FSK data portion of the CallerID Sequence

3.1.2 CID_Set_DTAS

Description:

This function sets the levels, frequencies, and timing of the Dual Tone Alerting Signal (DTAS) signal used in the CallerID sequence. (Note: this is refered to as CAS in the Belcore terminology). Note: these changes are not applied until the CallerID sequence is started using the CID_Send function.

Function Prototype:

long CID_Set_DTAS ALIAS (float Freq1, float Freq2, float Level1, float Level2, float OnTime)

Function Parameters:

freq1	This parameter sets the frequency (Hz) of the first tone in the DTAS signal
freq2	This parameter sets the frequency (Hz) of the second tone in the DTAS signal
level1	This parameter sets the level (Vrms) of the first tone in the DTAS signal.
level2	This parameter sets the level (Vrms) of the second tone in the DTAS signal.
ontime	This parameter sets the duration (ms) of the DTAS signal

3.1.3 CID_Set_Ring

Description:

This function sets the level, frequency, etc for the ringing signal applied during the CallerID sequences. Note: These changes do not take effect until the CallerID sequence is started using the CID_Send function.

Function Prototype:

long CID_Set_Ring (float Freq, float Level, float DC, long WaveShape, float StartPhase)

Function Parameters:

freq	This parameter sets the frequency (Hz) of the ringing pulse
level	This parameter sets the level (Vrms) of the ringing pulse
dc	This parameter sets the DC offset to be applied during the ringing pulse
waveshape	This parameter sets the waveshape for the RPAS signal. This parameter accepts the same values as the waveshape parameter in the Set_Ring function.
startphase	This parameter sets the starting phase for each ringing burst generated in the CallerID sequence. Note the RPAS starting phase can be set independently.

3.1.4 CID_Set_RingCadence

Description:

This function sets the ringing cadence settings for generating more complicated ringing patterns. This function allows each ringing burst to contain up to 3 different ringing pulses each of which can be programmed to different lengths and inter-burst times.

Function Prototype:

long CID_Set_RingCadence (long NumPulses, float OnTime1, float OffTime1, float OnTime2, float OffTime2, float OnTime3, float OffTime3)

Function Parameters:

numpulses	This parameter sets the number of pulses (1 to 3) to generate in each ringing burst.
ontime1	This parameter sets the duration (ms) of the first ringing pulse
offtime1	This parameter sets the time interval from the first to second ringing pulse. If the number of pulses is set to 1 then this parameter sets the time interval from the ringing pulse to the next ringing burst.
ontime2	This parameter sets the duration of the second ringing pulse. If the number of pulses is set to less than two then this parameter is ignored.
offtime2	This parameter sets the time interval between the second and third ringing pulse. If the number of ringing pulses is set to less than 2 then this parameter is ignored. If 2 pulses are to be generated then this parameter sets the time from the second pulse to the start of the next ringing burst.
ontime3	This parameter sets the duration of the third ringing pulse. If the number of ringing pulses is set to less than 3 then this parameter is ignored.
offtime3	This parameter sets the time interval between the third ringing pulse and the next ringing burst. If the number of ringing pulses is set to less than 3 then this parameter is ignored.

3.1.5 CID_Set_NumRings

Description:

This function sets the total number of ringing bursts to generate during the CallerID sequence (if ringing is a valid signal in the selected signaling type)

Function Prototype:

long CID_Set_NumRings (long NumRings)

Function Parameters:

NumRings	This parameter sets the number of ringing bursts to generate during the CallerID sequence

3.1.6 CID_Set_DTMF

Description:

This function sets the frequencies, levels, and timing for the DTMF portion of CallerID sequences. These parameters only affect DTMF callerID sequences.

Function Prototype:

```
long CID_Set_DTMF (float RowLevel, float ColumnLevel, float OnTime,
float OffTime, float FreqOffSet)
```

Function Parameters:

RowLevel	This parameter sets the level (Vrms) for the tone corresponding to the row DTMF frequencies
ColumnLevel	This parameter sets the level (Vrms) for the tone corresponding to the column DTMF frequencies
OnTime	This parameter specified the duration (ms) for each of the DTMF digits in the CallerID sequence
OffTime	This parameter specifies the inter-digit timing (ms) for the DTMF digits in the CallerID sequence
FreqOffset	This parameter sets the frequency offset (%) for the DTMF digits in the CallerID sequence

3.1.7 CID_Set_RPAS

Description:

This function sets the level, frequency, and timing of the Ringing Pulse Alerting Signal (RPAS) signal used in the CallerID sequence.

Function Prototype:

```
long  CID_Set_RPAS(float Freq, float Level, float DC, float OnTime, long
WaveShape, float StartPhase)
```

Function Parameters:

freq	This parameter sets the frequency (Hz) of the ringing pulse
level	This parameter sets the level (Vrms) of the ringing pulse
dc	This parameter sets the DC offset to be applied during the ringing pulse

waveshape	This parameter sets the waveshape for the RPAS signal. This parameter accepts the same values as the waveshape parameter in the Set_Ring function.
startphase	This parameter sets the starting phase for the RPAS signal

3.2 CallerID Timing/Signalling Functions

The following functions specify the timing parameters for the CallerID signaling sequences.

3.2.1 CID_Set_TimingToRing

Description:

This function sets the timing from the previous signaling element to the start of the first ringing burst.

Function Prototype:

```
long CID_Set_TimingToRing (float  ms)
```

Function Parameters:

ms	This parameter sets the time from the previous signaling element to the start of the first ringing burst

3.2.2 CID_Set_TimingToData

Description:

This function sets the timing from the previous signaling element to the start of the CallerID data.

Function Prototype:

```
long CID_Set_TimingToData (float ms)
```

Function Parameters:

ms	This parameter sets the time from the previous signaling element to the start of the CallerID data

3.2.3 CID_Set_TimingToDTAS

Description:

This function sets the timing from the previous signaling element to the start of the Dual Tone Alerting Signal (DTAS) signaling ellement.

Function Prototype:

```
long CID_Set_TimingToDTAS (float ms)
```

Function Parameters:

ms	This parameter sets the time from the previous signaling element to the start of the DTAS

3.2.4 CID_Set_TimingToLineReverse

Description:

This function sets the timing from the previous signaling element to the line reversal signaling ellement.

Function Prototype:

```
long  CID_Set_TimingToLineReverse (float ms)
```

Function Parameters:

ms	This parameter sets the time from the previous signaling element to the line reversal

3.2.5 CID_Set_HookTimeout

Description:

This function sets the maximum time to wait for detecting the hookswitch in the relavent CallerID signalling type.

Function Prototype:

```
long CID_Set_HookTimeout  (float HookTimeout)
```

Function Parameters:

HookTimeout	This parameter specifies the time (ms) to wait for a change in the hook-switch state

3.2.6 CID_Set_ACKDetector

Description:

This function sets the timing and detection parameters for the ACK detector used in the TypeII CallerID signaling.

Function Prototype:

long CID_Set_ACKDetector (long ACKTimeout, char *ACKDigits, float MinLevel, float FreqTol)

Function Parameters:

ACKTimeout	This parameter sets the maximum time to wait (ms) for the reception of the ACK signal.
ACKDigits	This string contains a digit for each of the acceptable DTMF digits that are acceptable ACK signals. Typically this string should be set to “AD”
MinLevel	This parameter sets the minimum level (Vrms) for the DTMF detector used when detecting the ACK signal.
FreqTol	This parameter sets the allowable frequency deviation of the ACK signal (%)

3.2.7 CID_Set_OSI_Duration

Description:

This function specifies the duration of the OSI generated during the CallerID sequence.

Function Prototype:

long CID_Set_OSI_Duration (float Duration)

Function Parameters:

Durationt	This parameter specifies the duration of the OSI generated during the CallerID signalling sequence.

3.3 CallerID Message Functions

The following functions build the contents of the CallerID messages and transmit the data with different signalling types.

3.3.1 CID_Set_MessageFormat

Description:

This function sets the formatting options for the FSK data sent during the CallerID sequence.

Function Prototype:

long CID_Set_MessageFormat (long CSBits, long MarkBits,long MarkOutBits, long StopBits)

Function Parameters:

CSBits	This parameter sets the number of channel seizure bits sent before the FSK data.
MarkBits	This parameter sets the number of mark bits sent after the channel seizure and before the FSk data
MarkOutBits	This parameter sets the number of mark bits sent after the FSK data
StopBits	This parameter sets the number of stop bits for each byte

3.3.2 CID_Set_MessageParity

Description:

This function sets the parity for the character portions of the CallerID messages. See the CallerID message generation functions to see which arguments have parity applied.

Function Prototype:

long CID_Set_MessageParity (long MsgParity)

Function Parameters:

MsgParity This parameter controls the parity setting for the character arguments in the CallerID messages. The allowed settings are as follows.

parity Value	Parity Setting
0	No Parity – 8 bits per character
1	Odd Parity – 7 bits per characater
2	Even Parity – 7 bits per character

Check function documentation for which arguments have the parity setting applied.

3.3.3 CID_ClearMessage

Description:

This function clears the CallerID data from memory. This function only affects the contents of the CallerID data, this function does not affect any of the physical parameters.

Function Prototype:

```
long CID_ClearMessage ( )
```

3.3.4 CID_SDMF_Number

Description:

This function creates an SDMF calling number CallerID message based on the CallingNumber and DateTime parameters.

Function Prototype:

```
long  CID_SDMF_Number  (char DateTime[ ], char CallingNumber[ ])
```

Function Parameters:

DateTime	This parameter specifies the date and time of the CallerID message in an 8 character string formatted as follows: **MMDDHHmm** **MM**: is the month represented as an ASCII number string with valid range from "01" to "12" **DD**: is the day represented as an ASCII number string with valid range from "01" to "31" **HH**: is the hour represented as an ASCII number string with valid range from "00" to "23" **mm**: is the minute represented as an ASCII number string with valid range from "00" to "59" **This parameter will have the message parity setting applied**
CallingNumber	This parameter is a string (typically up to 24 digits) which contains the calling telephone number. ie "5551234" **This parameter will have the message parity setting applied**

3.3.5 CID_SDMF_Absence

Description:

This function creates an SDMF callerID message with the calling number absent. The message contains a reason for the absence of the calling number.

Function Prototype:

long CID_SDMF_Absence (char DayTime[ ], char Reason[ ])

Function Parameters:

DateTime	This parameter specifies the date and time of the CallerID message in an 8 character string formatted as follows: **MMDDHHmm** **MM**: is the month represented as an ASCII number string with valid range from "01" to "12" **DD**: is the day represented as an ASCII number string with valid range from "01" to "31" **HH**: is the hour represented as an ASCII number string with valid range from "00" to "23" **mm**: is the minute represented as an ASCII number string with valid range from "00" to "59" **This parameter will have the message parity setting applied**
Reason	This parameter specifies the reason for the absence of the calling number. Typically this parameter is set to "P" to indicate the number is private, or "O" to indicate the number is unavailable. **This parameter will have the message parity setting applied**

3.3.6 CID_SDMF_VMWI

Description:

This function creates an SDMF Visual Message Waiting Indication (VMWI) message.

Function Prototype:

long CID_SDMF_VMWI (long Activate)

Function Parameters:

Activate	If non-zero then the visual message-waiting indicator is activated. If zero then the visual message-waiting indicator is de-activated.

This parameter will have the message parity setting applied

3.3.7 CID_MDMF_Set_MessageType

Description:

This function sets the message type byte for an MDMF CallerID message.

Function Prototype:

long CID_MDMF_Set_MessageType (long MessageType)

Function Parameters:

MessageType	This value should contain the message type for the MDMF message. See the appropriate CallerID standard document for allowed values.

3.3.8 CID_MDMF_Add_Parameter

Description:

This function insterts a generic MDMF parameter to the current CallerID data. The parameter consists of a parameter type and a series of bytes. The function should be passed an address to the start of the byte data (indicated here as a character pointer) and the number of bytes contained in that location. This function does not apply the parity setting to any of the arguments.

Function Prototype:

long CID_MDMF_Add_Parameter (long ParamType, char *ByteContents, long NumBytes)

Function Parameters:

ParamType	This parameter should contain the value of the MDMF parameter type for the specified argument.
ByteContents	This parameter should point to the first byte in MDMF parameter data. Note: This parameter can contain non-printable and NULL characters since the number of bytes to be copied is specified by the NumBytes parameter.
NumBytes	This parameter specifies the number of bytes in the MDMF parameter.

3.3.9 CID_MDMF_Add_DateTime

Description:

This function adds the Date and Time parameter to the current MDMF message.

Function Prototype:

long CID_MDMF_Add_DateTime (char DateTime[ ])

Function Parameters:

DateTime	This parameter specifies the date and time of the CallerID message in an 8 character string formatted as follows: **MMDDHHmm** **MM**: is the month represented as an ASCII number string with valid range from “01” to “12” **DD**: is the day represented as an ASCII number string with valid range from “01” to “31” **HH**: is the hour represented as an ASCII number string with valid range from “00” to “23” **mm**: is the minute represented as an ASCII number string with valid range from “00” to “59” **This parameter will have the message parity setting applied**

3.3.10 CID_MDMF_Add_CallingNum

Description:

This function adds the Calling Number (Calling Line Identity) parameter to the current MDMF message.

Function Prototype:

long CID_MDMF_Add_CallingNum (char CallingNumber[ ])

Function Parameters:

LineIdent	This parameter should contain an ASCII string containing the telephone number of the calling party (ie. “5551234”) **This parameter will have the message parity setting applied**

3.3.11 CID_MDMF_Add_NumAbsence

Description:

This function adds the Reason for Absence of Calling Number (Calling Line Identity) parameter to the current MDMF message.

Function Prototype:

long CID_MDMF_Add_NumAbsence (char Reason[ ])

Function Parameters:

Reason	This parameter should contain a code indicating the reason for the absence of the calling number. Typically the character “O” indicates that the number is unavailable, and the character “P” indicates that the number is private. **This parameter will have the message parity setting applied**

3.3.12 CID_MDMF_Add_CallingName

Description:

This function adds the Calling Name parameter to the current MDMF message.

Function Prototype:

long CID_MDMF_Add_CallingName (char CallingName[ ])

Function Parameters:

CallingName	This parameter should contain an ASCII string indicating the name of the calling party. (ie. “John Smith”) **This parameter will have the message parity setting applied**

3.3.13 CID_MDMF_Add_NameAbsence

Description:

This function adds the Reason for Absence of Calling Name to the current MDMF message.

Function Prototype:

long CID_MDMF_Add_NameAbsence (char Reason[ ])

Function Parameters:

Reason	This parameter should contain a code indicating the reason for the

absence of the calling party name. Typically “P” indicates that the name is private, and “O” indicates the name is unavailable.

This parameter will have the message parity setting applied

3.3.14 CID_MDMF_Add_VisualInd

Description:

This function adds the Visual Indicator parameter to the current MDMF message.

Function Prototype:

long CID_MDMF_Add_VisualInd (long Activate)

Function Parameters:

Activate	If non-zero, then the visual indicator will be activated. If this parameter is set to zero then the visual indicator will be deactivated.

3.3.15 CID_Send

Description:

This function sends the current CallerID using a specified signallling type. The signaling type defines the signals and sequence for delivering the CallerID information.

Function Prototype:

long CID_Send (long DeviceId, long SignalType, long *DataSent)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
SignalType	This parameter specifies the signaling type for delivering the CallerID data. See the section 3.4 for more information.
DataSent	This parameter returns a non-zero value if the CallerID data was transmitted successfully. This is usefull for determining if the CallerID sequence was successful when the signalling type requires certain conditions before sending data.

3.3.16 CID_Get_ACK_Info

Description:

If a DTMF ACK digit was detected during the CallerID sequence then this function will return all the detected information on the digit. This function will return the digit information even if the digit was not one of the qualified ACK digits.

Function Prototype:

long CID_Get_ACK_Info (char *DetectedDigit, float *RowFreq , float * ColumnFreq , float *RowLevel, float *ColumnLevel, float *StartTime, float * StopTime)

Function Parameters:

DetectedDigit	This parameter will return a single ASCII character representing the ACK digit detected. If no ACK is detected then this parameter will be set to space (decimal 32)
RowFreq	This returns the frequency (Hz) of the row tone in the ACK signal
ColumnFreq	This returns the frequency (Hz) of the column tone in the ACK signal
RowLevel	This returns the level (Vrms) of the row tone in the ACK signal
ColumnLevel	This returns the level (Vrms) of the column tone in the ACK signal
StartTime	This returns a time stamp of the start of the ACK digit
StopTime	This returns a time stamp of the end of the ACK digit

3.3.17 CID_Get_NumMessageBytes

Description:

This function returns the number of bytes in the current FSK based CallerID message.

Function Prototype:

long CID_Get_NumMessageBytes (long *NumBytes)

Function Parameters:

Numbytes	This parameters returns the number of bytes in the current FSK based CallerID message.

3.3.18 CID_Get_MessageByte

Description:

This function returns the value of a byte in the current FSK based CallerID message.

Function Prototype:

long CID_Get_MessageByte (long Index, long *Value)

Function Parameters:

Index	This parameter indicates which byte to read from the FSK data. This value can range from 1 to NumBytes.
Value	This parameter returns the value of the indexed byte in the FSK data

3.3.19 CID_Set_MessageByte

Description:

This function sets the value of a byte in the current FSK based CallerID message.

Function Prototype:

long CID_Set_MessageByte (long Index , long Value)

Function Parameters:

Index	This parameter indicates which byte to write in the FSK data. This value can range from 1 to NumBytes.
Value	This value (0-255) is stored in the indexed location in the FSK data.

3.3.20 CID_Get_CheckSum

Description:

This function gets the value of the checksum for the currently generated FSK CallerID message.

Function Prototype:

long CID_Get_CheckSum (long Value)

Function Parameters:

Value	The parameter is set to the value of the checksum for the currently generated FSK message.

3.3.21 CID_Set_CheckSum

Description:

This function sets the value of the checksum for the currently generated FSK CallerID message.

Function Prototype:

```
long CID_Set_CheckSum  (long Value)
```

Function Parameters:

Value	The parameter sets to the value (0 to 255) of the checksum for the currently generated FSK message.

3.3.22 CID_DTMF_Set_StopCode

Description:

This function sets the value of the stop-code for the DTMF based CallerID Message. This stopcode is appended to the end of the DTMF parameters.

Function Prototype:

```
long CID_DTMF_Set_StopCode (char StopCode[ ] )
```

Function Parameters:

StopCode	This parameter should be set to the DTMF character used to terminate the DTMF CallerID message.

3.3.23 CID_DTMF_AddParameter

Description:

This function adds a parameter to the DTMF CallerID message. Each Parameter consists of a StartCode and parameter consisiting of a string of DTMF digits.

Function Prototype:

```
long CID_DTMF_AddParameter (char StartCode[ ], char Param[ ] )
```

Function Parameters:

StartCode	This parameter should be set to a DTMF character used as a start code for the current parameter.
Param	This parameter should contain a string of DTMF digits containing the information for the DTMF CallerID parameter.

3.4 CallerID Signalling Types

The CID_Send function supports transmission of CallerID information using a variety of different signalling types. The supported signalling types are:

Signal Type Value	Description
0	**Load Data Only -** This signalling type simply loads the generated FSK message into the AI-7280. Additional code will be required to transmit the CallerID message.
1	**Send Data -** This signalling type simply transmits the FSK/DTMF data with no other signals generated.
2	**Send Data after Ring -** This signalling type will generate a single ring, transmit the FSK/DTMF data, and then continue ringing (if enabled)
3	**Send Data after OSI -** This signalling type will generate an Open Switching Interval (OSI), transmit the FSK/DTMF data, and then generate ringing (if enabled).
4	**Send Data after DTAS -** This signalling type will generate the Dual Tone Alerting Signal (DTAS), transmit the FSK/DTMF data, and then generate ringing (if enabled)
5	**Send Data after Line Reverse -** This signalling type will reverse the telephone line, send the FSK/DTMF data, return the telephone line to the normal polarity, and then generate ringing (if enabled)
6	**Send Data after Line Reverse, DTAS -** This signalling type will reverse the telephone line polarity, send the DTAS signal, transmit the FSK/DTMF CallerID, return the telephone line to the normal polarity, and then generate ringing (if enabled)
7	**Send Data after RPAS -** This signalling type will generate Ringing Pulse Alerting Signal (RPAS), transmit the FSK/DTMF data, and then generate ringing (if enabled).
8	**Send Data after DTAS, ACK** -This signaling type will generate the DTAS signal, wait for the ACK signal, and then transmit the FSK/DTMF data. This signalling type does not generate ringing.
9	**Send Data after Line Reverse, Off Hook -** This signalling type will reverse the telephone line polarity, wait for the CPE/TE to go off-hook, transmit the FSK/DTMF data, wait for the CPE/TE to go back on-hook, return the telephone line to normal polarity, and then generate ringing (if enabled).

The timing relationships between each of the generated signals in the signalling types can be set using the CallerID timing function defined in Section 3.2. Each of the timing functions specifies the time from the end of the previous signal to the start of the defined signal.

3.5 CallerID Transmission Timing Fuctions

The following functions are valid after a call to CID_Send and return time stamps associated with the start of signals sent during a CallerID transmission. These are especially useful when determining delays between DTAS, ACK, and FSK data.

3.5.1 CID_Get_DTAS_StartTime

Description:

This function returns a time stamp corresponding to the start of the DTAS signal sent in the last CallerID transmission. This is valid only after a successful call to CID_Send where the signaling type generates DTAS.

Function Prototype:

```
float CID_Get_DTAS_StartTime()
```

Return Value

Time stamp corresponding to the start of DTAS in the last CallerID transmission or a negative value if no such signal was sent.

3.5.2 CID_Get_FSK_StartTime

Description:

This function returns a time stamp corresponding to the start of the FSK signal sent in the last CallerID transmission. This is valid only after a successful call to CID_Send where data is sent using FSK (not DTMF)

Function Prototype:

```
float CID_Get_FSK_StartTime ()
```

Return Value

Time stamp corresponding to the start of FSK in the last CallerID transmission or a negative value if no such signal was sent.

4 SMS Functions

4.1 Physical Layer/Timing Functions

4.1.1 SMS_Set_FSK

Description:

This function sets the levels and frequencies for the FSK data in the SMS messages.

Function Prototype:

long SMS_Set_FSK (float MarkLevel, float MarkFreq, float SpaceLevel, float SpaceFreq, float Baud)

Function Parameters:

MarkLevel	This sets the level (Vrms) of the mark tone used to in the SMS transmissions
MarkFreq	This parameter sets the frequency (Hz) of the mark tone used in the SMS transmissions
SpaceLevel	This parameter sets the level (Vrms) of the space tone used in the SMS transmissions
SpaceFreq	This parameter sets the frequency (Hz) of the space tone used in the SMS transmissions
Baud	This parameter sets the baud rate used in the SMS transmissions

4.1.2 SMS_Reset_Timer

Description:

This function resets the timer used for timing the SMS transmissions and SMS receiver

Function Prototype:

long SMS_Reset_Timer(long Deviceid)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned Open_Device.

4.2 SMS Message TX/RX Functions

4.2.1 SMS_Set_TXMessageFormat

Description:

This function sets some of the FSK message properties for transmiting an SMS message.

Function Prototype:

```
long SMS_Set_TXMessageFormat ( long CSBits, long MarkBits, long
MarkOutBits, long StopBits)
```

Function Parameters:

CSBits	This value sets the number of channel seizure bits to send before the message
MarkBits	This value sets the number of mark bits transmitted before the data portion of the message
MarkOutBits	This value sets the number of mark bits that are transmitted after the data portion of the message
StopBits	This value sets the length of each stop bit in the FSK data.

4.2.2 SMS_Set_RXProtocol

Description:

This function sets the SMS protocol used by the SMS receiver. If protocol 1 is specified then the receiver simply looks for mark followed by at least one byte of data. If protocol 2 is specified then requires channel seizure, followed by a mark signal, optionally followed by one or more bytes of data.

Function Prototype:

```
long SMS_Set_RXProtocol  (long Protocol )
```

Function Parameters:

Protocol	This parameter sets the protocol for the SMS receiver. This value can be set to 1 or 2 for protocol 1 and 2 respectively

4.2.3 SMS_Set_TXMessage

Description:

This function specifies the contents of the SMS message to transmit. Note: This function sets the data to be transmitted to all connected AI-7280 devices.

Function Prototype:

```
long SMS_Set_TXMessage ( long DLLMsgType, char *TLData, long
NumTLBytes)
```

Function Parameters:

DLLMsgType	This parameter sets the Data Link Layer (DLL) message type for the SMS message.
TLData	This parameter should point to the first location in an array of bytes containing the contents of the Transfer Layer (TL) message.
NumTLBytes	This parameter specifies the number of bytes to be copied from the TLData location

4.2.4 SMS_Get_NumTXMessageBytes

Description:

This function returns the total number of FSK data bytes to be transmitted in the current SMS message.

Function Prototype:

```
long  SMS_Get_NumTXMessageBytes (long NumBytes )
```

Function Parameters:

NumBytes	This parameter returns the number of bytes in the currently generated SMS message

4.2.5 SMS_Get_TXMessageByte

Description:

This function returns a single byte from the currently generated SMS message.

Function Prototype:

```
long SMS_Get_TXMessageByte(long Index, long *ByteValue )
```

Function Parameters:

Index	This parameter specifies the byte to return from the currently generated SMS message. This value should range from 1 to NumBytes
ByteValue	This parameter returns the value at the specified index in the currently generated SMS message

4.2.6 SMS_Set_TXMessageByte

Description:

This function sets a single byte in the currently generated SMS message.

Function Prototype:

```
long SMS_Set_TXMessageByte(long index,long ByteValue)
```

Function Parameters:

Index	This parameter specifies the byte to modify in the currently generated SMS message. This value should range from 1 to NumBytes
ByteValue	This byte value is stored into the index location in the currently generated SMS message

4.2.7 SMS_Get_TXDLLXSum

Description:

This function returns the Data Link Layer checksum from the currently generated SMS message.

Function Prototype:

```
long SMS_Get_TXDLLXSum (long *XSum)
```

Function Parameters:

XSum	This parameter returns the checksum from the generated SMS message

4.2.8 SMS_Set_TXDLLXSum

Description:

This function sets the Data Link Layer checksum in the currently generated SMS message.

Function Prototype:

```
long SMS_Set_TXDLLXSum (long XSum)
```

Function Parameters:

XSum	This parameter sets the checksum in the currently generated SMS message

4.2.9 SMS_Send

Description:

This function transmits the currenly generated SMS message (generated using the SMS_Set_TXMessage function). The message can be transmitted with a specific time delay from the previous SMS transmission or receiption using the RelativeTimeDelay parameter.

Note: The minimum time delay between SMS receive and transmit events is 80ms and requires a USB connection. This minimum timing can also be increased by many factors including processor speed, loading, and the number of devices on the USB bus.

Function Prototype:

```
long SMS_Send  (long Deviceid, float RelativeTimeDelay, long Wait)
```

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
RelativeTimeDelay	This value specifies the relative time delay between the transmission of this SMS message and the end of the previous SMS transmission or receiption. If the specified time has already passed then the SMS message will be transmitted immediately. Note: See notes above for additional information about the minimum timing.
Wait	If this parameter is non-zero then control will not be returned to the calling program until the SMS message is completely transmitted.

4.2.10 SMS_Set_MinFSKLevel

Description:

This function sets the minimum FSK level required for the SMS message receiver. All incoming SMS messages below this level will be ignored.

Function Prototype:

```
long SMS_Set_MinFSKLevel (float MinFSKLevel )
```

Function Parameters:

MinFSKLevel	This parameter sets the minimum FSK level (Vrms) for the incoming SMS messages. All incoming SMS messages below this level will be ignored.

4.2.11 SMS_Receive

Description:

This function starts the SMS receiver algorithm. If wait is set to zero then this function will return control immediately (before a message has been received**). In this case, the user MUST repeatedly call SMS_Get_RXStatus in order to determine when the receiver is finished and to download the received message.** Note: Setting wait to a non-zero value results in the fastest receiver timing.

Function Prototype:

long SMS_Receive (long Deviceid , float RXTimeout, long Wait, long * MsgReceived , long *TimedOut)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
RXTimeout	This parameter sets the number of millseconds to wait for the complete receiption of an SMS message. In other words, this parameter specifies the maximum allowed time from the function call to the end of the incoming SMS message.
Wait	If non-zero then this function will not return until an SMS message has been received or the timeout has been reached.
MsgReceived	If wait is set to a non-zero value, then when the function completes this parameter will be set to a non-zero value if a message was received
TimedOut	If wait is set to a non-zero value, then when the function completes this parameter will be set to a non-zero value if the SMS receiver timedout.

4.2.12 SMS_Stop_Receive

Description:

This function allows the user to prematurely stop the SMS receiver before the timeout has been reached. This function is only applicable if the SMS_Receive function is called with the wait parameter set to zero.

Function Prototype:

long SMS_Stop_Receive (long Deviceid)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

4.2.13 SMS_Get_RXStatus

Description:

This function returns the status of the SMS receiver. This function must be called if SMS_Receive is called with the wait parameter set to 0. This function contains the mechanism to download the received SMS packet.

Function Prototype:

```
long SMS_Get_RXStatus  (long Deviceid, long *IsActive,long
*MsgReceived, long *TimedOut )
```

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
IsActive	This parameter indicates if the SMS receive algorithm is active. Once this value returns a zero value, the user should check the MsgReceived and TimedOut parameters.
MsgReceived	If the IsActive property returns a zero value then the MsgReceived property will be set to a non-zero value if a new SMS message was received.
TimedOut	If the IsActive property returns a zero value then the TimedOut property will be set to a non-zero value if the receiver timed out before an SMS message was received.

4.2.14 SMS_Get_RXMessageInfo

Description:

This function returns information on the latest SMS message received. Note: before calling this function the user should check to insure that a message has been received by using the MsgReceived parameter of either the SMS_Receive function or the SMS_Get_RXStatus functions.

Function Prototype:

long SMS_Get_RXMessageInfo (long Deviceid, long *CSDetected, long * TotalBytes, long *NumTLDataBytes, float *FSKLevel, float * StartTime, float *StopTime)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
CSDetected	This parameter is set to a positive non-zero value if channel seizure was detected in the incoming SMS message
TotalBytes	This parameter returns the total number of bytes received in the SMS message (this includes the message type, length, checksum, and Tlbytes)
NumTLDataBytes	This parameter indicates the number of Transfer Layer data bytes were decoded in the SMS message. (This value does not include the message type ,length, or checksum)
StartTime	This parameter returns a timer value of when the SMS packet started
StopTime	This parameter returns a timer value of when the SMS packet ended

4.2.15 SMS_Get_RXData

Description:

This function returns the data contents of the received SMS message. Note: before calling this function the user should call the SMS_Get_RXMessageInfo function to determine the number of bytes received in the SMS message (if any). **The TLBytes parameter of this function MUST be allocated to be at least NumTLDataBytes bytes BEFORE this function is called.**

Function Prototype:

```
long SMS_Get_RXData  (long Deviceid, long *DLLMsgType, long *
DLLMsgLength, long *DLLXSum, long *XSumOK, long *NumFrameErrs,
char *ByRef TLByte,long BytesToCopy, long *BytesCopied)
```

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
DLLMsgType	This parameter returns the Data Link Layer message type from the received SMS message
DLLMsgLength	This parameter returns the Data Link Layer message length parameter from the SMS message (note: this is not neccessarity the number of bytes returned in the TLBytes parameter, as this byte could be corrupted)
DLLXSum	This parameter returns the value of the Data Link Layer checksum
XsumOK	This parameter will be set to a non-zero value if the checksum is correct
NumFrameErrs	This parameter returns the number of framing errors (incorrect stop bits) detected in the SMS message.
TLBytes	This parameter should point to the first location in an array of bytes where the Transfer Layer bytes will be stored. This array must be have a length equal to NumTLDatBytes which can be obtained from the SMS_Get_RXMessageInfo function.
BytesToCopy	This parameter sets the maximum number of bytes that will be transferred into the TLBytes array. Normally this value should be set to NumTLDatBytes as returned from the SMS_Get_RXMessageInfo function. If you insure that this value is always less than or equal to the size to the TLBytes array, then you will prevent any possible General Protection Fault errors
BytesCopied	This parameter indicates the actual number of bytes copied into the TLBytes array (in the event that the BytesToCopy value was larger than the number of bytes received in the message)

4.2.16 SMS_Get_RXAllBytes

Description:

This function returns all the bytes received in the SMS message into one array argument. These index for these byte values will correspond to the byte status values returned from the SMS_Get_RXByteStatus function.

Function Prototype:

long SMS_Get_RXAllBytes (long Deviceid, char *ByteValues, long NumBytesToCopy, long *BytesCopied)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
ByteValues	This parameter should point to the first location in an array of bytes where the message bytes will be stored. This array should have a length equal to TotalBytes, which can be obtained from the SMS_Get_RXMessageInfo function.
NumBytesToCopy	This parameter sets the maximum number of bytes that will be transferred into the *ByteValues* array. Normally this value should be set to TotalBytes as returned from the SMS_Get_RXMessageInfo function. If you insure that this value is always less than or equal to the size to the *ByteValues* array, then you will prevent accidental General Protection Fault errors
BytesCopied	This parameter indicates the actual number of bytes copied into the *ByteValues* array (in the event that the *NumBytesToCopy* value was larger than the number of bytes received in the message)

4.2.17 SMS_Get_RXByteStatus

Description:

This function returns all the status bytes for the data received in the SMS message. These index for these status values will correspond to the byte values returned from the SMS_Get_RXAllBytes function.

Function Prototype:

long SMS_Get_RXByteStatus (long Deviceid, char *ByteStats, long NumBytesToCopy, long *BytesCopied)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
ByteStats	This parameter should point to the first location in an array of bytes where the status bytes will be stored. This array should have a length equal to TotalBytes, which can be obtained from the SMS_Get_RXMessageInfo function. If any of the values are non-zero, this indicates that a framing error occurred when receiving the corresponding byte value.
NumBytesToCopy	This parameter sets the maximum number of bytes that will be transferred into the *ByteStats* array. Normally this value should be set to TotalBytes as returned from the SMS_Get_RXMessageInfo function. If you insure that this value is always less than or equal to the size to the *ByteStats* array, then you will prevent accidental General Protection Fault errors
BytesCopied	This parameter indicates the actual number of bytes copied into the *ByteStats* array (in the event that the *NumBytesToCopy* value was larger than the number of bytes received in the message)

5 Other Functions

5.1 Global Script Program Functions

The global script program functions allow the DLL to load script programs compiled for the AI-7280 directly on the AI-7280. These script programs can be generated automatically through the TRSim software package, or compiled in the AI-workbench environment. Note: when a global script is run, all normal DLL functions are suspended until the global script program is terminated.

5.1.1 Run_GlobalProgram

Description:

This function executes a global script program on the AI-7280. Once this function has successfully completed, all normal DLL functions are suspended until the global script program is terminated.

Function Prototype:

```
long Run_GlobalProgram(long Deviceid, long Processor, char UserScript[ ])
```

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
Processor	This parameter specifies which of the 6 virtual processors to execute the global script program on.
UserScript	This parameter should contain the compiled script program. This compiled code can be located in the .obc files generated by either AI-Workbench or TRSim.

5.1.2 Run_GlobalFlashProgram

Description:

This function executes a global script program stored in the flash memory on the AI-7280. Once this function has successfully completed, all normal DLL functions are suspended until the global script program is terminated.

Function Prototype:

long Run_GlobalFlashProgram (long Deviceid, long Processor, long ProgramNum)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
Processor	This parameter specifies which of the 6 virtual processors to execute the global script program on.
ProgramNum	This parameter specifies the file number of the program to execute.

5.1.3 Halt_GlobalProgram

Description:

This function halts all global script programs executing on the AI-7280 and resumes normal DLL operations. Note: all signal generation will be stopped after this call and many of the signalling setting may be modified due to the global program operation. Care must be excercised to insure that all the necessary signalling values are restored after this function call.

Function Prototype:

long Halt_GlobalProgram (long Deviceid)

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.

5.1.4 Get_GlobalProgramStatus

Description:

This function returns the status of one of the six processors that can be executing a global script program.

Function Prototype:

```
long Get_GlobalProgramStatus (long Deviceid, long Processor, long * Status)
```

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
Processor	This parameter specifies which processor's status should be returned
Status	This value returns the status of the script processor. The expected values are: 0 = Stopped 1 = Running 2 = Halted (can be resumed) 3 = Single stepping mode 4 = waiting for interrupt If the status value is not one of the above values then this indicates that an error has occurred on the processor. Please contact technical support if you encounter such a condition.

5.1.5 Get_GlobalProgramVariable

Description:

This function returns the status of one of the six processors that can be executing a global script program.

Function Prototype:

long Get_GlobalProgramVariable (long Deviceid, long Processor, long VarNum, long VariableType, long IsString, char Value[ ])

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
Processor	This parameter specifies which processor's memory pool should be accessed
VarNum	This parameter specifies the variable location in the memory pool to access
VariableType	This parameter specifies which memory pool to access. The possible values are: 0 = Processor Data Pool (for the processor specified) 1 = Global Data Pool (shared by all processors) 2 = Register Location (for the processor specified)
IsString	If this value is non-zero then the "value" parameter is treated as a string, otherwise it is assumed that "value" is a string representation of a numeric value ie. "1.0245"
Value	This parameter returns a string or numeric value from the specified data pool. **This string must be initialized to at least 200 characters before calling this function. See** Data Type Summary **for initialization details.**

5.1.6 Put_GlobalProgramVariable

Description:

This function returns the status of one of the six processors that can be executing a global script program.

Function Prototype:

long Put_GlobalProgramVariable (long Deviceid, long Processor, long VarNum, long VariableType, long IsString, char Value[ ])

Function Parameters:

Deviceid	The deviceid parameter specifies which AI-7280 to communicate with. This deviceid must be returned by the Open_Device function.
Processor	This parameter specifies which processor's memory pool should be accessed
VarNum	This parameter specifies the variable location in the memory pool to access
VariableType	This parameter specifies which memory pool to access. The possible values are: 0 = Processor Data Pool (for the processor specified) 1 = Global Data Pool (shared by all processors) 2 = Register Location (for the processor specified)
IsString	If this value is non-zero then the "value" parameter is treated as a string, otherwise it is assumed that "value" is a string representation of a numeric value ie. "1.0245"
Value	This parameter specifies a value to store in the specified location in the data pool. If IsString is set to a non-zero value then this can be any valid string up to 64 characters. If IsString is cleared then this should be a decimal representation of a numeric value to store. ie "1.023". Note: this value cannot be in scientific or exponential form.

6 Error Codes

Every function in the AI-7280 DLL returns a standard error code value that indicates if the function completed with or without error. The value of the error code returned indicates the nature of any error that occurred. The DLL function Get_ErrorDesc is included to translate these error code values into English sentences to assist in debugging.

The error codes are categorized into four categories:

- Communication errors
- Resource conflict errors
- Parameter value errors
- Internal System Errors

Communication errors occur when communications with a device is lost, communications resources are unavailable, or the device identifier in a function call is invalid. Resource conflict errors occur when the specified task cannot be performed because a required resource is already in use by another signal generator internal to the AI-7280. Parameter value errors occur when an argument in a function call is outside of the allowable range. Internal system errors are unexpected and should not occur during normal operation, however the presence of the error condition prevents further execution.

The following sections document the error code values and the associated error conditions.

6.1 Communication Errors

Error Code	Name	Description
1	Timeout	Communications with the connected device timed out. The device didn't respond to a communications request within a specified time limit. All further communications with this device will be halted.
2	Bad device ID	The deviceid parameter of the function doesn't correspond to a valid communications handle created by Open_Device
3	Bad port number	The port number specified in the Open_Device function is invalid.
4	Port not open	The communications channel for the device is not open or inactive due to an error.
5	No matching serial number	No AI-7280 devices with matching serial number (if specified) could be located on the communications channel specified
6	Bad serial number format	The format of the serial number specified is invalid. AI-7280 serial numbers must be in the format "SN12XXXX" where the X characters are replaced with the numeric digits 0 through 9.
7	Communications halted	A communications error has occurred for this device. The specified operation could not be completed because the communications with the connected device has been halted.
8	Communication channel not ready	The communications channel is either in use or unavailable.
9	Operation not supported by current AI-7280 software version	The current software version in the AI-7280 does not support the requested operation. Please check www.adventinstruments.com for the latest software version.
10	Bad Reset Script	The reset script could not be executed. Please insure that the flash file contents are not corrupted.

6.2 Resource Conflict Errors

Error Code	Name	
100	FSK conflict	Tone generator A cannot be used while the FSK generator is active. Also the SMS Receive algorithm cannot be started while FSK is active
101	AM conflict	Tone generators A and B cannot be used while the AM generator is active
102	MF/DTMF conflict	Tone generators C and D cannot be used while either the MF or DTMF generators are active.
103	Ring conflict	No other signal generation function are allowed while the ring generator is active
104	FSK already on	The FSK generator must be stopped before it can be started again.
105	MF/DTMF already on	The MF/DTMF generator must be stopped before it can be started again. This error will also occur if the DTMF generator is started while MF is running or vice versa.
106	FSK Dropout Conflict	The FSK dropout properties cannot be updated while the FSK generator is active.
107	SMS RX Conflict	The current operation cannot complete because the SMS Receive algorithm is already active.
108	Global Script Running	A global script is executing. Normal DLL operations are suspended until the global script operations are terminated.
109	No Global Script Running	No global scripts are running. The function called is not available during normal DLL operation.

6.3 Parameter Value Errors

Error Code	Name	Description
500	Parameter 1 too low	The value of parameter 1 is below the acceptable range of values
501	Parameter 1 too high	The value of parameter 1 is above the acceptable range of values.
502	Parameter 1 invalid	Parameter 1 is invalid. Check the acceptable range of values for this parameter.
503	Parameter 2 too low	The value of parameter 2 is below the acceptable range of values
504	Parameter 2 too high	The value of parameter 2 is above the acceptable range of values.
505	Parameter 2 invalid	Parameter 2 is invalid. Check the acceptable range of values for this parameter.
506	Parameter 3 too low	The value of parameter 3 is below the acceptable range of values
507	Parameter 3 too high	The value of parameter 3 is above the acceptable range of values.
508	Parameter 3 invalid	Parameter 3 is invalid. Check the acceptable

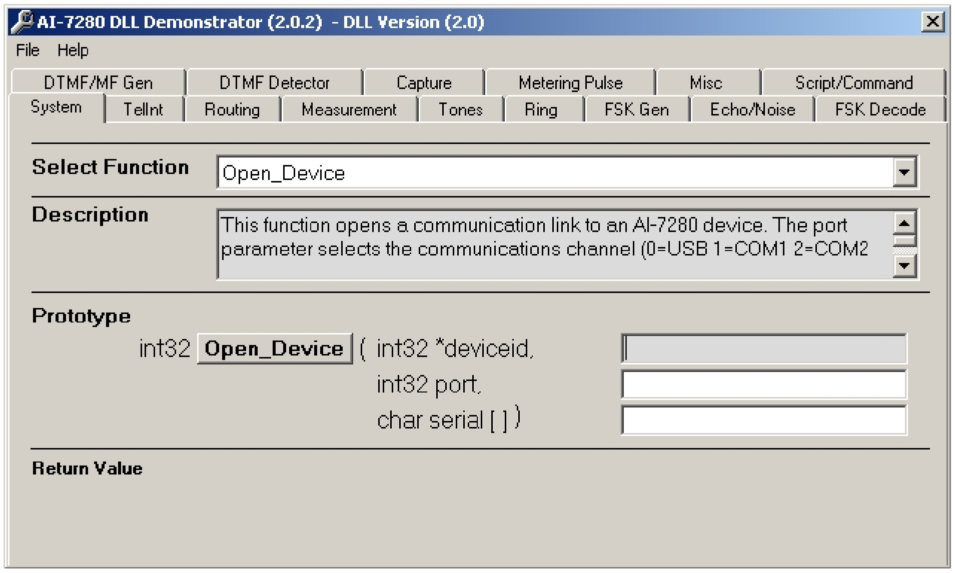
		range of values for this parameter.
509	Parameter 4 too low	The value of parameter 4 is below the acceptable range of values
510	Parameter 4 too high	The value of parameter 4 is above the acceptable range of values.
511	Parameter 4 invalid	Parameter 4 is invalid. Check the acceptable range of values for this parameter.
512	Parameter 5 too low	The value of parameter 5 is below the acceptable range of values
513	Parameter 5 too high	The value of parameter 5 is above the acceptable range of values.
514	Parameter 5 invalid	Parameter 5 is invalid. Check the acceptable range of values for this parameter.
515	Parameter 6 too low	The value of parameter 6 is below the acceptable range of values
516	Parameter 6 too high	The value of parameter 6 is above the acceptable range of values.
517	Parameter 6 invalid	Parameter 6 is invalid. Check the acceptable range of values for this parameter.
518	Parameter 7 too low	The value of parameter 7 is below the acceptable range of values
519	Parameter 7 too high	The value of parameter 7 is above the acceptable range of values.
520	Parameter 7 invalid	Parameter 7 is invalid. Check the acceptable range of values for this parameter.
521	Parameter 8 too low	The value of parameter 8 is below the acceptable range of values
522	Parameter 8 too high	The value of parameter 8 is above the acceptable range of values.
523	Parameter 8 invalid	Parameter 8 is invalid. Check the acceptable range of values for this parameter.
530	Bad MF symbol	One or more of the MF symbols specified in an argument string is invalid. Valid MF symbols range from "A" to "T" (not case sensitive)
531	Bad DTMF digit	One or more of the DTMF digits specified in an argument string is invalid. Valid DTMF digits correspond to the characters "0" to "9", "A" to "D", "*" and "#"
532	String too long	One of the string arguments in the function is too long. String arguments must be limited to 64 characters in length.
533	String contains non-printable character	One of the string parameters in the function call contains a non-printable character ie. ASCII code is less than 32
534	User script program too long	The script program specified is too large to fit into the remaining memory in the AI-7280
535	Global Data Register In Use	The global data pool register location specified is already in use by the DLL. Please select a storage location at a larger address.
536	Illegal CR or LF	An illegal Carriage Return (13) or Line Feed (10) character was detected midway through the string argument
537	Flash File Doesn't Exist	The flash file specified doesn't exist.

6.4 Internal System Errors

While undesirable, it is sometimes unavoidable that an expected error condition may occur that prevents further execution. This DLL handles these errors as internal system errors and return codes with values greater than 600. If your application causes a system error, please contact technical support with the error code number and any additional information on the conditions or code that caused the error.

7 DLL Demo Program

A demonstration program is included with the AI-7280 DLL to allow a developer to experiment with the DLL function calls through a graphical user interface. To run the demo program execute the file 7280DLLdemo.exe that was installed with the DLL developer kit. Figure 1 below shows the initial window of the demonstration program.



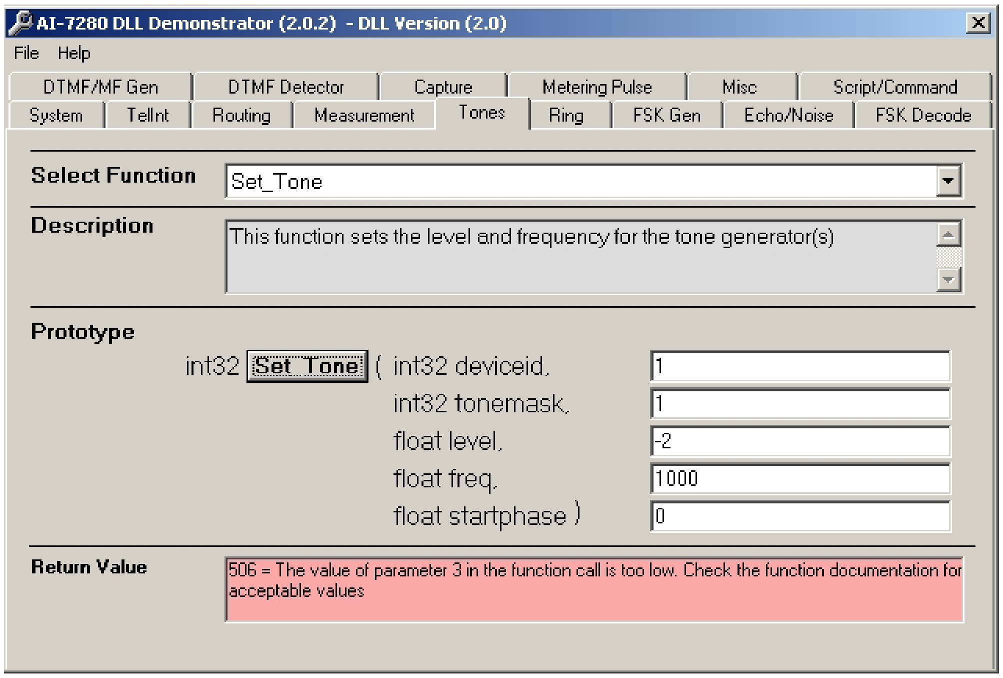
Select Function: Open_Device

Description: This function opens a communication link to an AI-7280 device. The port parameter selects the communications channel (0=USB 1=COM1 2=COM2

Prototype
int32 Open_Device (int32 *deviceid,
int32 port,
char serial [ ])

Figure 1 DLL Demonstration Program

The tabs at the top of the screen sort the functions by category to make finding the appropriate function simpler. To make a function call, select the appropriate function category by clicking on the desired tab and then select the function name using the drop-down list to the right of the ‘Select Function’ label. The function description and the function prototype will appear. Now you can insert values for each function parameter by typing either integer, floating point, or string values into the text boxes to the right of each function parameter. To get more information on a device parameter hold the mouse cursor over the name of the parameter and a short description of the parameter will appear. To call the function, simply click the button with the function name in the function parameter section. When the function completes a box will appear in the return value section with either “No Error” if the function completed normally, or a description of the error if the function returned an error code. An example of the return value display is shown in Figure 2.



Select Function

Set_Tone

Description

This function sets the level and frequency for the tone generator(s)

Prototype

```
int32 Set_Tone ( int32 deviceid,     1
                 int32 tonemask,     1
                 float level,        -2
                 float freq,         1000
                 float startphase )  0
```

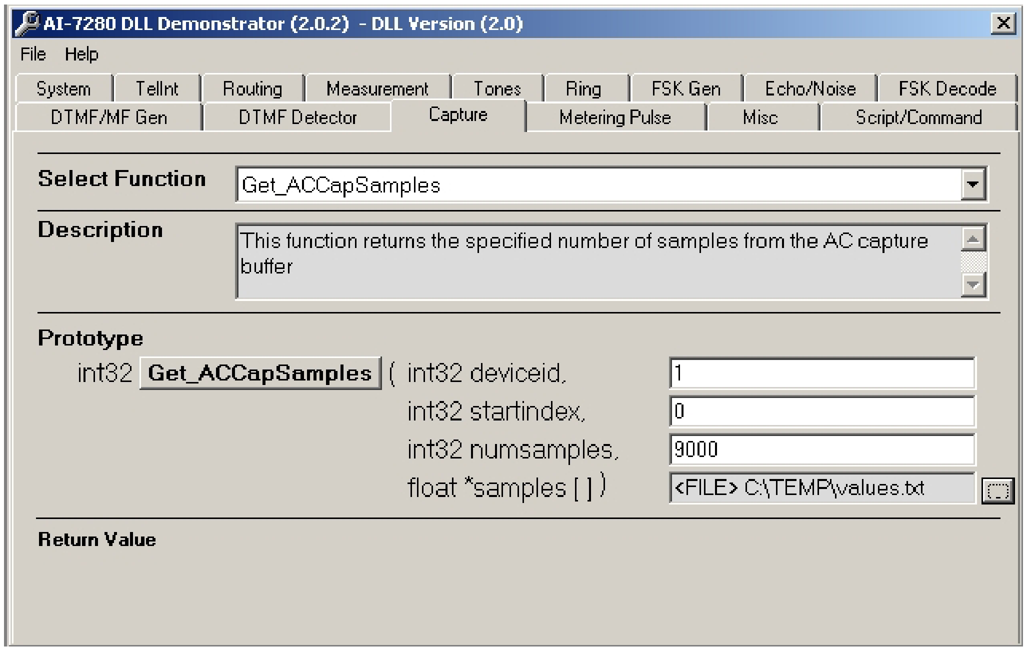
Return Value

506 = The value of parameter 3 in the function call is too low. Check the function documentation for acceptable values

Figure 2 Demo Program Return Value Display

To begin using the demo program with the AI-7280, connect an AI-7280 unit to the serial or USB port on the PC and call Open_Device (with the appropriate settings) by clicking on the button labeled “Open_Device” in the function prototype. Once this function has been successfully called, all of the other functions can be used. All the deviceid fields will be automatically completed with the deviceid of the last device connected to using Open_Device.

Some of the functions require parameters to be passed as arrays, such as the Get_ACCapSamples function. The demo program handles the arrays through text files. Any function arguments that require an array appear with a small box to the right as shown in Figure 3.



Select Function: Get_ACCapSamples

Description: This function returns the specified number of samples from the AC capture buffer

Prototype

```
int32 Get_ACCapSamples ( int32 deviceid,      1
                         int32 startindex,    0
                         int32 numsamples,    9000
                         float *samples [ ] ) <FILE> C:\TEMP\values.txt
```

Return Value

Figure 3 Array Parameter in Function Call

To specify the path of the text file click on the button and a dialog will appear that will allow you to select the file. Once the file has been selected the path of the file will appear in the gray box next to the function parameter. If the array is an input argument the values in the file will be loaded at this time. If the array is an output parameter, then the values from the function call will be saved to file when the function call is made. The text file format requires one data point per line in decimal or exponential format (ie 1.239547e-3, 345.23, 0.0001). Also, if the array parameter is of integer type you can specify hexidecimal numbers in the format 0x0F1A2. Do not pad with spaces or any other characters in the array arguments.

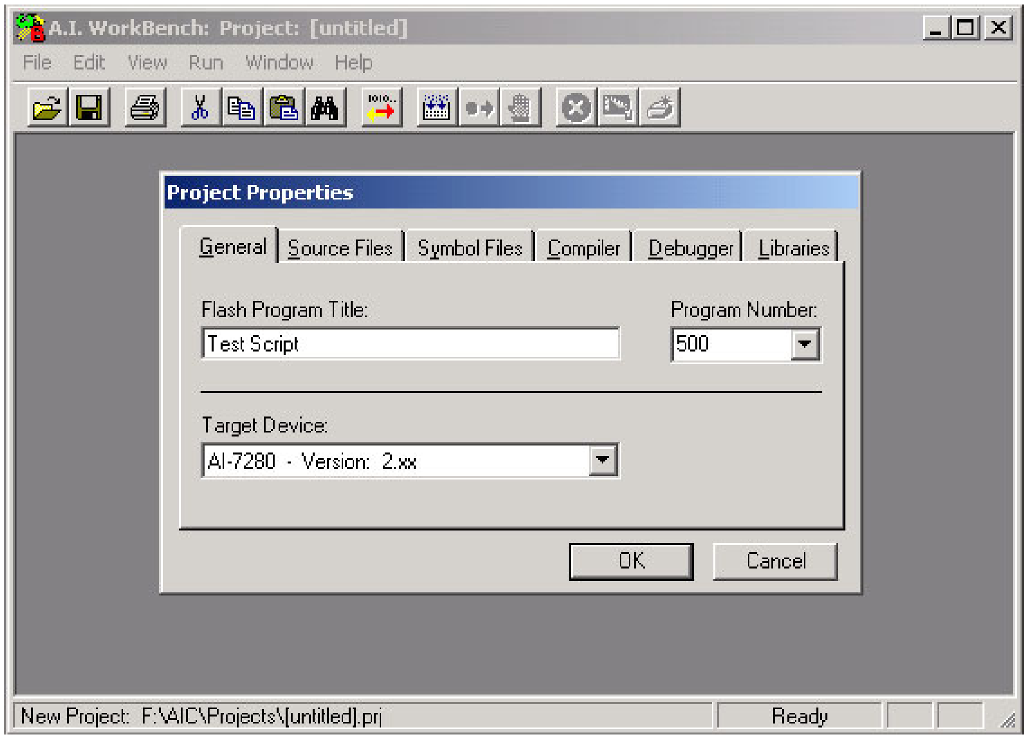
8 Using Scripting Features

The AI-7280 allows the user to gain direct control over the AI-7280 features by loading and running a script directly on the AI-7280. The script program can be run in conjunction with other DLL functions using the Run_UserScript and related functions. If the user wishes to take complete control of the unit (as may be necessary for some extremely complicated sequences) then the Run_GlobalScript and related function should be used. The following sections outlines the general procedures for generating and running scripts through the DLL interface.

8.1 Generating Scripts with AI-Workbench

AI-Workbench is the software tool used for generating all script programs for the AI-7280. AI-Workbench can be found on the CD distributed with the AI-7280 and also on the web at www.adventinstruments.com.

Once you have installed the AI-Workbench software, run the program and open a new project by selecting [File][New Project]. You will be presented with a project properties window as shown in Figure 4.

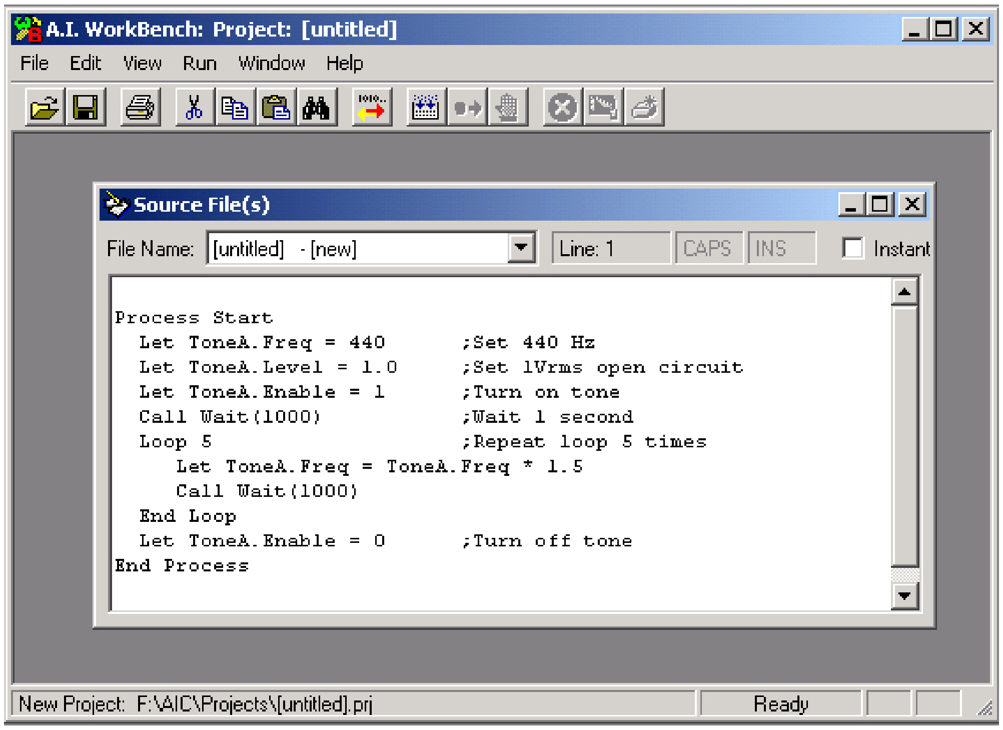


Flash Program Title: Test Script
Program Number: 500
Target Device: AI-7280 - Version: 2.xx
New Project: F:\AIC\Projects\[untitled].prj

Figure 4 AI-Workbench Project Properties

Give your project an appropriate title in the “Flash Program Title” box and insure that the Target Device dialog is selected to the correct version for your AI-7280, and then click “OK”.

A source code window will now appear where you can enter your source code. An example of a very simple program is shown in Figure 5.


A.I. WorkBench: Project: [untitled]
File
Edit
View
Run
Window
Help
Source File(s)
File Name: [untitled] - [new]
Line: 1
CAPS
INS
Instant
New Project: F:\AIC\Projects\[untitled].prj
Ready

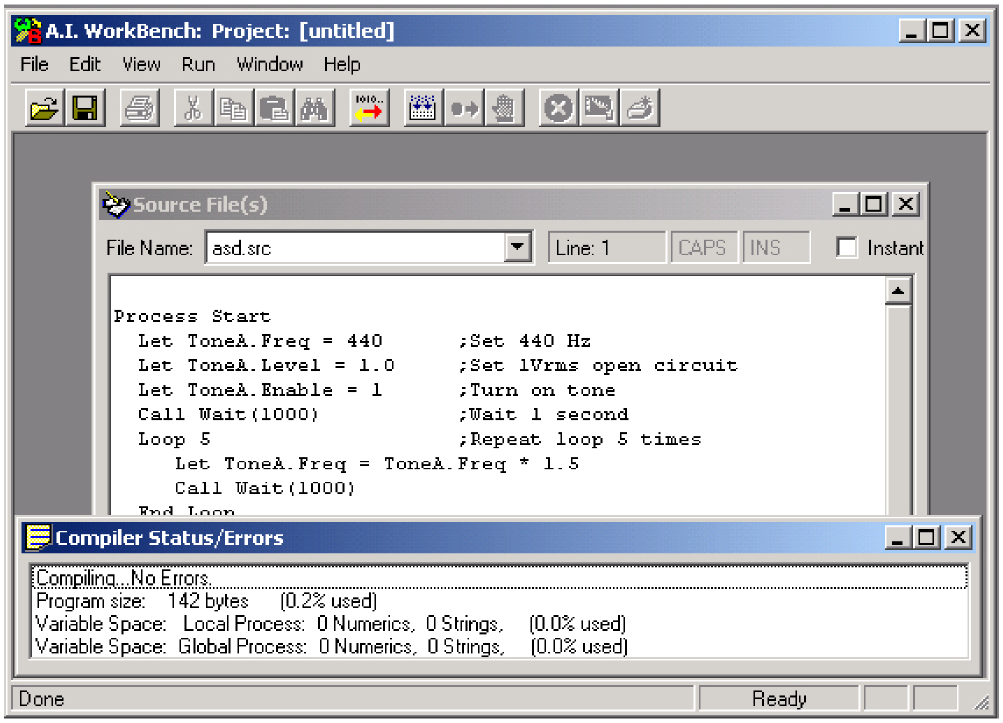
```
Process Start
  Let ToneA.Freq = 440       ;Set 440 Hz
  Let ToneA.Level = 1.0      ;Set 1Vrms open circuit
  Let ToneA.Enable = 1       ;Turn on tone
  Call Wait(1000)            ;Wait 1 second
  Loop 5                     ;Repeat loop 5 times
     Let ToneA.Freq = ToneA.Freq * 1.5
     Call Wait(1000)
  End Loop
  Let ToneA.Enable = 0       ;Turn off tone
End Process
```

Figure 5 AI-Workbench Example Script

The script shown in Figure 5 sweeps a tone through 6 starting from 440Hz increasing by 50% on each pass. The AI-Workbench syntax and structure is quite involved and is beyond the scope of this document. For more information on the scripting language you should contact technical support for the appropriate documentation. Most of the basic script structure is documented in the manual for the AI-80, however there are obvious syntax and functionality differences between the two products.

Once you have completed your script you should save the project by clicking [File][Save Project]. The software will then prompt you for the name of the project, and then the name of the source file.

Once the project is saved you can compile the project by clicking [Run][Compile]. If the compilation is successful then a project status window will appear as shown in Figure 6.



```
Process Start
  Let ToneA.Freq = 440       ;Set 440 Hz
  Let ToneA.Level = 1.0      ;Set 1Vrms open circuit
  Let ToneA.Enable = 1       ;Turn on tone
  Call Wait(1000)            ;Wait 1 second
  Loop 5                     ;Repeat loop 5 times
     Let ToneA.Freq = ToneA.Freq * 1.5
     Call Wait(1000)
  End Loop
```

```
Compiling...No Errors.
Program size:    142 bytes     (0.2% used)
Variable Space:   Local Process:  0 Numerics,  0 Strings,    (0.0% used)
Variable Space:  Global Process:  0 Numerics,  0 Strings,    (0.0% used)
```

Figure 6 AI-Workbench Compiler Status

When the compilation is successful, the compiler will generate an .obc file in the same directory as the project with the same name. So if your project name was "Test.prj" the compiler will generate a file named "Test.obc". The contents of this file are the "executable code" for the AI-7280's internal processor. The file will be in ASCII format and should be viewable through a simple text editor program like Notepad. The contents of the .obc file for the above example is:

BUF1X1TIN440HN96TIN1HN98TIN1HN95TIN0VN6TIN1000VN6WVN6TIN5VN6SJIN0ASIN1CLIN0F2SATHN96VN6AMIN1.5TVN6HN96TIN0VN6TIN1000VN6WVN6SOSRX2SQTIN0HN95ZS0

The contents of this file is the string argument that should be passed as the script argument to any of the script functions. **NOTE: Do not modify this scripts with any whitespace, CR,LF characters or it may produce unpredictable results. There should only be one carriage return in the string located at the very end.**

8.2 Designing and Running Scripts

Now that you have an idea how to write and compile AI-7280 script programs (as described in Section 8.1), you have to decide how you want the script to operate.

The AI-7280 has six processors each with its own memory space with which to run script programs. In addition to this local processor memory space, the AI-7280 has global memory space, which can be accessed by all of the processors. The DLL actually utilized five of these processors for managing many of the simultaneous activites that can be managed by the AI-7280. The sixth processor is left free so that users may generate programs to perform complicated actions that may not be possible through the normal DLL functions.

The AI-7280 DLL allows the users to execute scripts in two different fashions: User Scripts, and Global Scripts.

User Scripts are run on the sixth processor in parallel with normal DLL functions – this allows the user to supliment the functionality of the DLL without giving up the functionality of the current DLL functions. Unfortunately, since user scripts operate in conjunction with DLL scripts, very specific rules must be followed or the user may "break" the functionality of the DLL. The functions Run_UserScript, Control_Script, Get_ScriptStatus, Get_ScriptVariable, and Put_ScriptVariable all operate on user scripts.

Global Scripts are run completely independently of the DLL functions. When a global script is run, all DLL functions are suspended and all six processors are relinquished. This allows the user the most flexibility and memory useage, however none of the DLL functions will operate while a global script is run. When the global script is stopped, all normal DLL functions will continue. The TRSimm package can automatically generate global scripts from some CallerID sequences.

To run a user script that you authored using the steps in the previous section.

1. Load the contents of the scripts .obc file into a string variable (or string constant). Insure that you do not modify the contents of the string.

2. Call the Run_UserScript function and pass the script as a string parameter.

3. Once the script is running you should be able to monitor variables running on the processor using Get_ScriptVariable. Care should be taken when using Put_ScriptVariable to insure that you do not interfere with normal script operations!

3. You should periodically monitor the status of the script processor using Get_ScriptStatus until the script completes. Note: Your code should be prepared for the condition where the script does not complete or indicates a processor error. If this condition exists you should shut down the processor using Control_Script before continuing.

4. Once the script has completed (or you have shut the script down) you should be free to continue your normal DLL program.

The example code packaged with the DLL shows a simple example of how to run a global script on the AI-7280 in the function Run_GlobalProgramOn7280. If you require any further information on using these function calls or specific script information then please contact technical support.

8.3 Rules for User Scripts

Since user scripts run concurrently with the five other DLL support scripts it is vital that the programmer observe some basic rules for desiging user scripts.

1. Avoid global variables! The DLL support scripts use many global variables for communication between processes. It is unlikely that your program will require similar functionality, so it would be best to avoid using global memory space. Local processor variables cannot interfere with the DLL support scripts. If you decide to use global variables then be sure to follow the next rule!

2. Always check global variable usage! The DLL support script uses many global variables for communication between processes starting from location 0 and working up to a maximum value. The next available global variable location for user scripts can be returned using the function Get_FirstAvailGlobalReg. Note: This value may change with DLL versions since features are always being added. It would be a good idea to always design your memory map from the largest available location down to avoid conflicts. It is an even better idea to avoid global variable usage.

3. Always check the size of your script! Your user script program must share code space with the DLL support script. Always insure that your user script is shorter than the length returned by Get_ScriptMemAvail. Attempting to load a script larger than this value will fail. The maximum size of a user script will also change with DLL version.

4. Do not use the communication features! The scripting language allows the AI-7280 to send and receive characters on the COMM and USB port. Using these features while the DLL is active will almost guarantee and error as it will disrupt the normal DLL-to-script communications!

5. Do not access reasources you know to be active in the DLL! If your script attempts to access a resource (for example the ring generator) that is already being used by the DLL (say through Start_Ring) the AI-7280 may produce unexpected results.

6. Always leave things the way you found them! Since the user script program runs directly on the AI-7280 and has complete access to all of the device resources, it is possible to change device properties without updating some DLL support script status variables. It is good practice to always return system resources (tone generators, ring generators, etc) back to the original state after your user script terminates.

9 Revision History

Release 1.0

First public release

Release 1.1

- Fixed bug with the system timer
- Fixed bug with Comm port flow control

Release 2.0

- Tone generators can operate up to 18KHz
- Added support for AI-7280 Rev 2 features which includes
  - Metering Pulse Functions
  - Dropout support for FSK generator
  - Automatic echo disable during ringing
- New functions allow scripts to be loaded and run through DLL interface
- Fixed bug in Get_Echo function (previously this would only retrieve Tap 0)

Release 2.1

- A .h and .lib file has been included in the distribution to allow easy integration into Microsoft Visual C++ projects
- A small C++ example program has been added to the distribution to demonstrate the function calls.
- A correction has been made to the dll to prevent “Unprintable Character” errors from occurring with certain string arguments.
- The documentation has been corrected to indicate long types instead of int (since the size of the int data type is compiler specific. The AI-7280 DLL passes and returns 32-bit integer arguments)
- Corrected documentation for logic levels in Digital Output functions

Release 3.0

- New CallerID support functions have been added to allow the user to easily generate and send CallerID sequences without using the low level fuction calls
- New SMS support functions have been added to allow the transmission and receiption of SMS messages.
- Add_FSKHexString, and Set_RingCadence functions have been added to the DLL

- Added Start_MeterPulseWithCount and Get_MeterPulseCount functions to support generation of a specific number of metering pulses. (Available with Rev 2.12 AI-7280 firmware)
- Added global script support which allows any AI-7280 script program to be loaded, executed, and monitored.

Release 3.0a

- Corrected Close_Device parameter to to indicate pass by reference not by value
- Corrected Set_MFSymbol function to indicate Freq2 is passed by reference not by value.
- Added section to describe how to write and execute script programs on the AI-7280 using AI-Workbench and the DLL interface.

Release 3.0b

- Corrected typos in documentation

Release 3.1

- Fixed internal error which occurred when any tones were active before CID_Send was called using signaling type 4. The modifications allow tone generator B to be active through the CallerID sequence with signalling type 4.
- Improved Start_Tone behaviour such that successive calls to Start_Tone with a tone pattern will start the new tone pattern immediately.

Release 4.0

- Added functions Wait_For_PulseDial and Get_PulseDial_Stats to make pulse dialing detection simpler
- Added Create_OSI function so that Open Switching Intervals (OSIs) can be generated with more precise timing
- Added the Wait_For_LineFlash function to make the detection of a line flash simpler

Release 4.0b

- Updated Echo functions to indicate linear gain (V/V)

Release 4.0c

- Improved the documentation of the FSK dropout functions
- Corrected documentation of Start_FSKGen function to include the bitindex parameter

Release 4.0d

- Improved the documentation to better reflect the initialization requirements for strings used as output arguments.

Release 4.1

- Updated dll to trap invalid MaxTime values in Wait_For_HookState and Wait_For_DTMF functions and updated documentation to indicate the limits on MaxTime parameters
- Improved communication timeout code
- Fixed minor bug in Get_MeterPulseCount, which would erroneously return a “parameter out of range” error.

Release 4.1b

- Added documentations for VB.net users to highlight differences in the “Long” and “Integer” representations
- Added example project for VB.net users

Rev 4.2

- Added timing function which allow the user to determine when DTAS and FSK were sent during a CallerID sequence

Rev 4.4

- Fixed initialization bug which prevented connection to AI-7280 when serial number was specified on USB
- Resolved timing issue which could cause communication errors on slower PCs connecting using COM port
- Slight change in directory structure for installed program
- Added a complete Visual Studio 6 C++ project example complete with updated examples
- Minimum system requirements now require Windows 2000 SP4 or greater

Rev 4.5

- Added Set_ BNCOutGain function
- Fixed minor documentation errors

Rev 4.5a

- Updated domain names in documentation

Rev 4.6

- Mentioned removing limit checking on feed parameters.
- Updated USB driver install procedures and minor documentation errors

10 Support

For assistance in hardware setup, program installation, software operation, or general questions, please contact us in any of the following methods.

- Email: Technical Questions: techsupport @adventinstruments.com

  Sales Inquires: sales@adventinstruments.com
- In North America:

Tel:	(604) 944-4298
Fax:	(604) 944-7488
Mail:	Advent Instruments Inc.
	111 - 1515 Broadway Street
	Port Coquitlam, BC, V3C6M2
	Canada

- In Asia:

Tel:	(852) 8108-1338
Fax:	(852) 2900-9338
Mail:	Advent Instruments (Asia) Ltd.
	Unit No. 7, 9/F, Shatin Galleria
	18 - 24 Shan Mei Street
	Fotan, Shatin, N.T.
	Hong Kong

11 Appendix A: USB Driver Installation

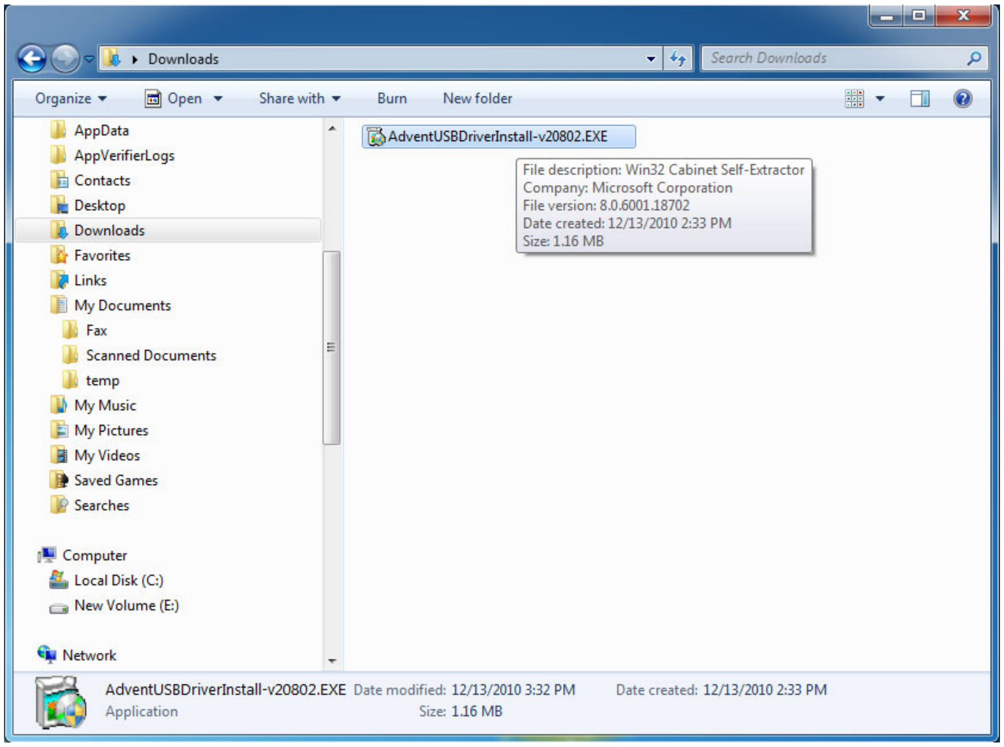
The WHQL certified driver files are automatically installed along with the TRsSim application and the following procedure should only be required if the end application will not use the TRsSim software.

The drivers may be installed manually by using the add remove hardware wizard or via the self extracting installer package. The following figures show the step by step procedure for installing the USB drivers with this installer package.

1. Download the USB installer Package

The USB installer package can be downloaded from our website at the following page

http://www.adventinstruments.com/Downloads/ or copied from the original CDrom provided with the instrument.



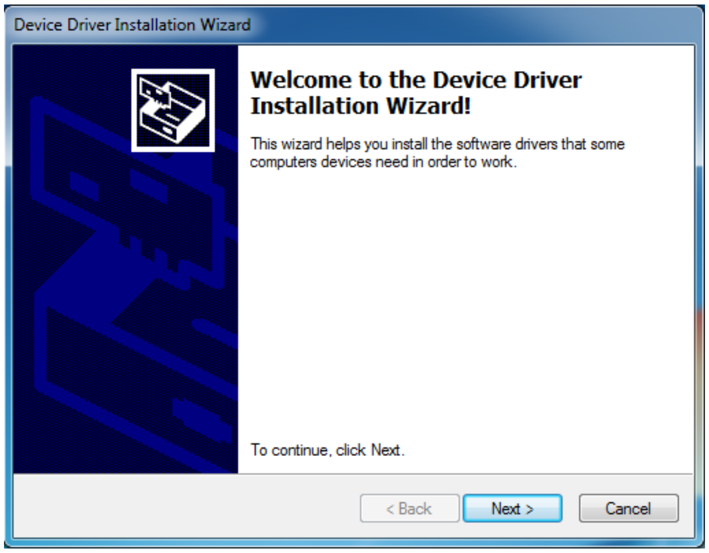
AdventUSBDriverInstall-v20802.EXE

File description: Win32 Cabinet Self-Extractor
Company: Microsoft Corporation
File version: 8.0.6001.18702
Date created: 12/13/2010 2:33 PM
Size: 1.16 MB

AdventUSBDriverInstall-v20802.EXE Date modified: 12/13/2010 3:32 PM Date created: 12/13/2010 2:33 PM
Application Size: 1.16 MB

2. Run the USB installer Package

Once the file is located on the target computer double click on the file to start the self extraction and the following window will be displayed.



Device Driver Installation Wizard

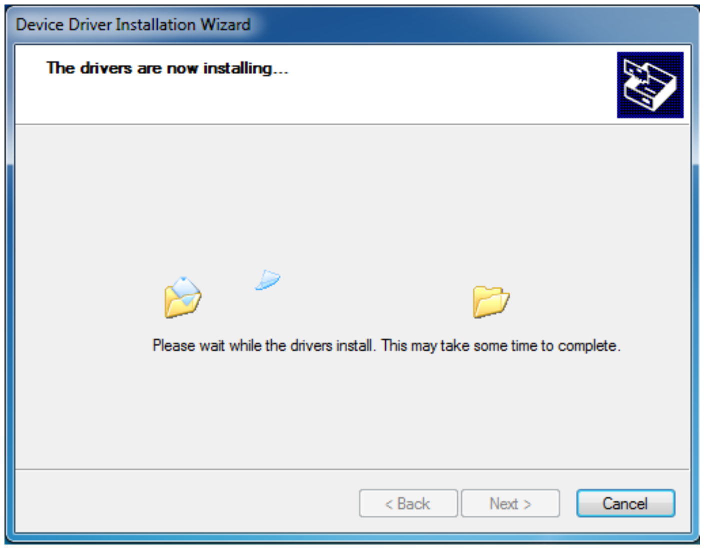
Welcome to the Device Driver Installation Wizard!

This wizard helps you install the software drivers that some computers devices need in order to work.

To continue, click Next.

3. Click on the Next button to install the driver files

Windows will now copy the driver files from the installer package to the target computer.



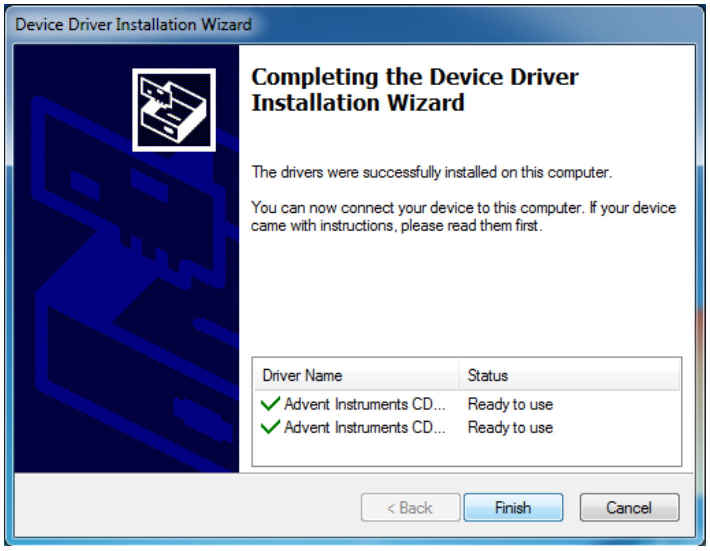
Device Driver Installation Wizard

The drivers are now installing...

Please wait while the drivers install. This may take some time to complete.

4. Confirm the Driver installed correctly

Once Windows has completed the installation of the driver files the following window will display the results. Confirm that there are two ✓ marks beside both of the listed driver names.



Device Driver Installation Wizard

Completing the Device Driver Installation Wizard

The drivers were successfully installed on this computer.

You can now connect your device to this computer. If your device came with instructions, please read them first.

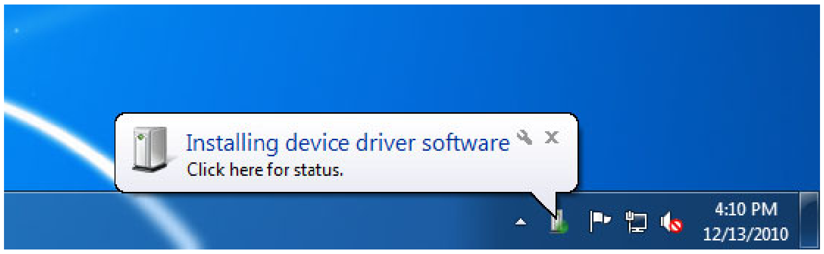
Driver Name	Status
✓ Advent Instruments CD...	Ready to use
✓ Advent Instruments CD...	Ready to use

5. Click on the Finish button to exit the installer

The driver is now installed and you may connect the instrument to the PC via a USB cable.

6. Connect the Instrument to the PC via the USB cable

The following brief popup bubble will appear indicating Windows has detected the new instrument.



Installing device driver software

Click here for status.

Then a short time later the following popup will appear and the instrument is recognized by Windows and the PC.