AI-7280

Central Office Line Simulator

Programmers Guide

Direct Control

Advent Instruments Inc.

Release 2.0a

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
1.1 Introduction 1
1.2 Firmware Overview 2
1.3 Communication Basics 4
1.3.1 Serial RS-232 Port 5
1.3.2 USB Port 5

2. Command Reference 7
2.1 Overview 7
2.2 Get & Set Commands 7
2.3 Program Control Commands 10
Clear Program Memory: PC 10
Load Program: PL"(program text)" 10
Start Program from RAM: PS(n)M 11
Start Program from Flash: PS(n)F(file#) 11
Halt Program Execution: PH(n) 11
Resume Program Execution: PR(n) 11
Stop Program Execution: PX(n) 11
Single Step: PT(n) 12
Set Program Load Address: PA(offset) 12

3. HAL Properties 13
3.1 Overview 13
3.2 ACCAP Properties 14
3.3 BULK Properties 17
3.4 COMM Properties 19
3.5 DATA Properties 21
3.6 DCCAP Properties 25
3.7 DCPROFILE Properties 27
3.8 DIO Properties 29
3.9 DTMF Properties 30
3.10 DTMFCAP Properties 32
3.11 ECHO Properties 34
3.12 FILE Properties 35
3.13 FILTER Properties 37
3.14 FSK Properties 39
3.15 FSKDROP Properties 41
3.16 MEASURE Properties 42
3.17 METERPULSE Properties 44
3.18 MFGEN Properties 46
3.19 NOISE Properties 49
3.20 RING Properties 50
3.21 SCHEDULER Properties 52
3.22 SIGNALIO Properties 54
3.23 SOURCE Properties 54
3.24 STATUS Properties 56

3.25 SYSTEM Properties 57
3.26 TELINT Properties 61
3.27 TIMER Properties 64
3.28 TONE Properties 66
3.29 TONEA Properties 67
3.30 TONEB Properties 71
3.31 TONEC Properties 72
3.32 TONED Properties 73
3.33 USB Properties 74

4. Programming 77

4.1 Overview 77
Capabilities 77
Program Development 78
4.2 Programming Example 78
What is TRsSim 79
Creating a Device Script Project 79
Direct Program Loading 84
4.3 Script Language Basics 85
Process Blocks 85
Function & Subroutine Blocks 86
Variables & Constants 87
Statements 89
Built in Routines and Properties 92

Appendix A: Firmware Revisions 97

Appendix B: Property Listing by Index 99

Appendix C: Error Codes 104

Appendix D: Support 107

Glossary of Terms 109

1. Getting Started

1.1 Introduction

The AI-7280 can be controlled from any device that provides a RS-232 serial port. By sending simple ASCII character commands, all of the low level functionality of the AI-7280 is accessible. While using this method of control provides the most flexibility, it is generally more complex than using the provided Win32 DLL or .NET Assembly. Both provide a layer of abstraction that simplifies many tasks. However for applications where the DLL or .NET assembly cannot be used (non-Windows PC), or where the DLL or .NET assembly cannot perform the specific task, this form of communication and control may be the best option.

It is possible to use the AI-7280's USB port for direct control; however, the correct USB drivers are required for this functionality. Currently drivers are available for Windows, Linux, and Apple OS-X operating systems. Please contact technical support for more information on USB driver availability if you wish to use the USB port.

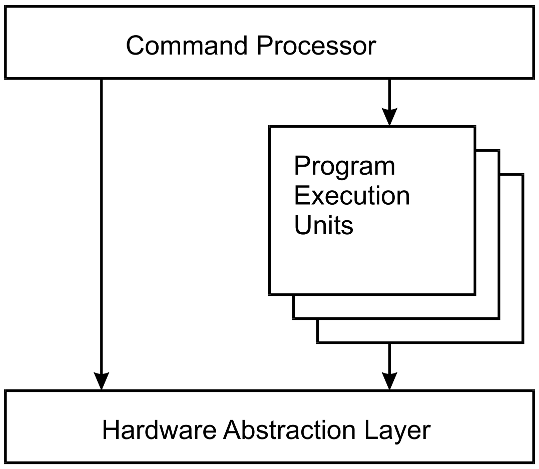
This document provides technical information on how the AI-7280 is controlled directly from the RS-232 serial port or USB port. It divided into the following sections, each describing a specific aspect of controlling the AI-7280.

- **Section** 1. Getting Started
  Provides a general overview of the AI-7280's firmware structure and communication basics.
- **Section** 2. Command Reference
  Describes the structure and function of the basic AI-7280 commands.
- **Section** 3. HAL Properties
  Provides reference information on the operation and function of all the hardware abstraction layer properties.
- **Section**

4. Programming
Provides an overview and example of how programs are developed, along with basic information on the scripting language constructs.

1.2 Firmware Overview

At its most basic level, the AI-7280 firmware can be broken down into three logical elements. They are the command processor, program execution units, and the hardware abstraction layer. While overly simplistic, the following figure shows the structure of those logical elements.



Starting from the bottom of the figure is the Hardware Abstraction Layer (HAL). This is simply a layer of software that controls all hardware related aspects of the AI-7280. The AI-7280 functionality is represented by over 200 properties. These properties are organized into groups of similar function. The groups control various settings, such as the ringing generator, tone generators, level meters, FSK modulator, FSK generator, DTMF detector, and others.

From a programming point of view, each property can be viewed as a variable or register. For example, to change the frequency of the ringing generator, a command is sent to write a value into the "Ring.Freq" property. By writing a value into a property register, the underlying hardware is immediately changed. The syntax used to define the various properties is as follows:

(group name) . (property name)

Some examples include:

Ring.Freq	(controls ring generator frequency)
TelInt.Voltage	(telephone interface line voltage)
ToneA.Level	(output level of tone generator A)
Measure.Level	(returns output of RMS level meter)

While most of the properties can be both written to and read from, some may be read-only, or write-only. For example, the **Measure.Level** property is a read-only property that cannot be written to.

The vast majority of the properties are of a numeric data type. This means that only numbers can be written, or read from that property. However, some of the properties are string types. This means that ASCII character strings may be written, or read from them.

Since all aspects of the AI-7280 are determined by the HAL property values, controlling the AI-7280 is reduced to simply reading and writing to the various properties.

There are two ways in which the property values can be manipulated. They are:

- Commands received from the RS-232 serial port, or USB port, or
- Statements executed by the program execution units.

The Command Processor block, shown in the previous figure, is responsible for interpreting characters received from either the RS-232 or USB ports. It simply waits for complete command strings to arrive from the controlling device, and then processes the command. It will either execute the desired command, or if a syntax error is detected, return an error message. The command processor operates in a strict master slave relationship with the controlling device. It acts as the slave and returns data only in response to a command.

The two commands used to access the HAL properties are set and get commands. The set command writes a value into any HAL property, while the get command returns the value stored in the HAL property.

In addition to set and get are other commands used to control the program execution units. The AI-7280 contains a number of virtual processors that can be programmed for a wide variety of tasks. These are the program execution units shown in the previous figure. The program execution units allow the AI-7280 to operate in a semi-autonomous manner. Programs running on the AI-7280 can perform various low-level timing critical tasks, while the PC (or other controlling device) performs the supervisory tasks. These programs may be stored in the AI-7280's non-volatile flash memory, or downloaded into the volatile RAM.

To aid in writing AI-7280 programs, the TRsSim software provides a simplified high-level language compiler. This software converts high-level language statements into object code executed by the AI-7280. An example of such a program is as follows:

```
Loop
    If TelInt.HookDetect = 1 Then
        Exit Loop
    End If
End Loop
Let ToneA.Freq = 440
Let ToneB.Freq = 350
Let ToneA.Level = 0.500
Let ToneB.Level = 0.500
Let ToneA.Enable = 1
Let ToneA.Enable = 1
```

The above program waits until a connected telephone device goes off-hook. Once this occurs, it enables tone generators A and B to produce the standard North American dial tone (440 & 350 Hz) at a level of 0.5 Vrms (open circuit).

This program can be compiled by the TRsSim software and stored in the AI-7280's flash memory. When needed, the PC (or other controlling device), can simply execute this program with one of the AI-7280's program execution units.

1.3 Communication Basics

All commands sent to and received from the AI-7280 use ASCII characters. A command is simply a series of ASCII characters followed by the <CR> character (ASCII 13). The <CR> character marks the end of the command. The AI-7280 only processes the command once the <CR> character is received. Optionally, a <LF> character (ASCII 10) may follow the <CR> character; however it is ignored and has no effect on the command.

For every command sent, the AI-7280 responds with either an acknowledgement, error code, or the requested data. Because ASCII characters are used for the commands, any terminal program (for example. Windows HyperTerminal) may be used to send commands. The AI-7280 does not echo any sent characters so a local echo must be enabled to see the commands in a terminal program's window. In addition, all responses sent by the AI-7280 are terminated with only a <CR> character. Not a <CR> <LF> combination.

Some command examples that can be sent from a terminal program are shown below.

Note, commands are represented by bold face text and must be followed by the <CR> character. The expected response is shown in the line underneath the command. Terminal program settings should be 9600 baud, 8 data bits, 1 stop bit, with no flow control.

1. Get the software version string:
   ```
   ?HS2
   "AI-7280 Software Version [2.12](1)"
   ```
2. Enable the ringing generator:
   ```
   >HN111=1
   OK
   ```
3. Turn off the ringing generator:
   ```
   >HN111=0
   OK
   ```
4. Measure and return the DC voltage on the telephone line:
   ```
   ?HN71
   -4.81737e1
   ```

More detailed information on the command syntax is provided in section 2. Command Reference.

Note: It is important to limit the length of any commands to less than 128 characters. This is because the AI-7280 must buffer all of the command characters prior to the <CR> character. The size of the buffer is fixed and this requires that all commands are less than 128 characters in length (including the <CR> character).

Note: AI-7280 commands are case sensitive. All commands use upper case characters exclusively.

1.3.1 Serial RS-232 Port

The AI-7280 serial port operates at one of the following baud rates: 9600, 19200, 38400, 57600, and 115200. The serial format is fixed at 8 data bits, 1 stop bit, and no parity. Flow control is not used (either hardware or software).

Upon a power up or reset, the baud rate is set to 9600 bps. It may be changed to one of the other supported baud rates by writing to the HAL property (Comm.Baud).

If the AI-7280 detects a line break (at least 11 consecutive space bits), it resets the baud rate back to the default value of 9600 bps and flushes any unprocessed commands from its internal buffer. When attempting to communicate with the AI-7280 for the first time, it is good practice to first send a line break, which ensures the baud rate is set to 9600 baud. Once communication is established, the baud rate may be set higher in order to improve throughput.

The rear panel 9 pin RS-232 connector is configured as a DCE (Data Communications Equipment) port. As such, no cross over cable is required when making a connection to a standard PC serial port. A minimum of three wires are required to make a serial connection. They are TX Data (pin 2), RX Data (pin 3), and Ground (pin 5). The AI-7280 can monitor the PC's RTS (pin 7) signal and control the state of the CTS (pin 8) signal, but does not use them to communicate. The DSR (pin 6) and DTR (pin 4) connections are internally tied together. Pins (1 and 9) have no internal connection.

Upon power up or a hardware reset, the AI-7280 outputs the following characters from the RS-232 serial port at 9600 baud.

```
ABL1<CR>
GO<CR>
```

They are generated by an internal boot loader program which loads the main firmware program of the AI-7280. Following these characters, the AI-7280 is ready to accept commands.

1.3.2 USB Port

The USB port may be used to communicate with the AI-7280 in the same manner as the RS-232 port. The command structure and syntax is identical. However in order to use the USB port, a suitable driver must be installed on the host PC. A Windows XP, Vista, 7 USB driver installation package is available at:

http://www.adventinstruments.com/Products/AI-7280/Downloads

Note: The USB drivers are automatically installed if the TRsSim software has been installed on the PC (version 3.0 and higher).

To determine if drivers are available for other operating systems, please contact technical support.

2. Command Reference

2.1 Overview

The AI-7280 supports two main classes of commands. They are:

- Get & Set Data
- Program Management

The get and set commands are used to write to or read from any of the Hardware Abstraction Layer (HAL) properties. They are the most important commands in controlling the AI-7280 settings and reading its measurements. In addition to accessing the HAL properties, the commands can access the variable memory space used by the program execution units, or registers internal to the execution units. For details on the use of these commands see the following section: 2.2 Get & Set Commands.

The second class of commands deals with program management and control the program execution units. Programs can be stored and executed from the non-volatile flash memory, or downloaded into RAM and executed. Commands are provided to start, halt, resume, and stop the program execution units. See section: 2.3 Program Control Commands for more detailed information.

It is important to note that the same command structure is used for both the RS-232 port and the USB port. There is no difference in syntax when using either communication method.

2.2 Get & Set Commands

From a programming point of view, the AI-7280 can be represented as a large collection of registers. Some of these registers control various hardware settings (like the HAL properties), while others access variable memory used by executing programs or registers controlling the state of program execution. In order to control the AI-7280, the set and get commands are used to write to and read from many different registers.

The syntax of the get and set commands are:

?(reg)	Get register contents
>(reg)=(data)	Set register contents

Where "(reg)" represents the register to access, and "(data)" represents the value to set the register to.

Registers contain data of either a numeric or character string type. In addition, registers are grouped into three different classes. They are HAL properties, program variable memory, and program execution control & status. As such, registers are defined by their class, data type, and an ID number.

(reg)	**<class> <data type> <ID number>**

Where:

<class>	'H' for HAL properties
	'G' for program variable registers
	'V' for program control & status registers
<data type>	'N' for numeric values
	'S' for strings
<ID number>	Integer value representing specific register

Some examples of registers are:

HN112	Numeric HAL property representing the frequency of the ringing generator.
HS2	String HAL property representing the software version string of the AI-7280 firmware.
VN103	Numeric status register for the first program execution unit.

In order to read these registers, get command '?' precedes the register. For example, sending the following commands reads the registers listed above.

```
?HN112
2.2e1

?HS2
"AI-7280 Software Version [2.12](1)"

?VN103
0
```

Note that each command must be terminated with a <CR> character (0Dh) before it is processed by the AI-7280. In addition, the data returned by the AI-7280 is always terminated with only a <CR> character.

In each of the get commands used above, the AI-7280 reads the contents of the register and returns its value. Reading "HN112" returns the frequency of the ringing generator, which by default is 22 Hz. Register "HS2" returns a string representing the firmware version, and "VN103" returns a value representing the program execution status (0=stopped, 1=running, 2=halted).

If the data type of the register being read is numeric, then the value is returned in scientific notation. Lower case 'e' delimits the mantissa portion from the exponent. In addition, the period '.' character is always used as the decimal symbol.

For string data types, the returned characters are enclosed in double quotation marks.

Multiple commands can be combined on the same command line. For example, the following command reads the registers for ringing frequency, level, and DC offset voltage simultaneously.

```
?HN112:?HN113:?HN116
2.2e1:6e1:4.8e1
```

Multiple commands are delimited by using the colon ':' character. In the response, the value of each register is also delimited by the colon character. In the example above, the ringing frequency is currently set at 22 Hz, ringing AC level at 60 Vrms, and ringing DC offset level at 48 Volts.

Writing data to registers requires using the set command. For example, to set the ringing frequency to 68.5 Hz, send the following command:

```
>HN112=68.5
OK
```

The AI-7280 responds with 'OK' to each valid set command. If multiple set commands are chained together, each one returns an 'OK' response. For example,

```
>HN112=22:>HN113=40
OK:OK
```

It is important to observe the following rules when using the set command:

- No space characters can be inserted between the register 'HN112', the equal sign '=', or the value passed '22', otherwise an error message will be returned.
- If setting a numeric value, the format for the data must be as follows:

  [-]n[.n] Where 'n' is one or more digits 0 to 9

  Examples of valid numeric values are:

  1

  3.1415926

  -314.159

  0.000833

  Examples of invalid numeric values are:

  3,1415926

  -3.14159e2

  .000833

- If setting a character string value, it must be enclosed in double quotation marks. To include quotation marks within the string, send two quotation marks.

  For example, to set the variable register 'GS1' to:

  He said "never", and left the room.

  Send the following command:

  ```
  >GS1="He said ""never"", and left the room."
  ```

If the AI-7280 cannot correctly interpret a command, its response is in the format of: 'ERR=x', where 'x' is an integer value representing the error condition. For a listing of all command error codes, see: Appendix C: Error Codes

In order to properly use the get and set data commands, it is important to understand the purpose of each of the AI-7280's registers. A summary of all HAL properties is listed in Appendix B: Property Listing by Index, while section 3. HAL Properties provides detailed information for each property.

2.3 Program Control Commands

An important feature of the AI-7280 is its ability to execute simple programs. These programs can be used to off load low level tasks from the PC or other controlling device. Up to six independent program execution units (PEU) can be operating at any one time. Each PEU can run a separate program, or a program can be constructed from multiple processes, which each PEU running a process.

The following commands are used to control the PEUs. If the AI-7280 correctly interprets the command, it returns the characters 'OK' in response. However, if an error is detected it returns the character string 'ERR=x', where 'x' represents an error code. See Appendix C: Error Codes for a listing of the possible error codes and their meaning.

Clear Program Memory: PC

This command clears the RAM program memory. Programs may be stored and executed out of the internal non-volatile flash memory, or the RAM. If using the RAM to load and execute programs, send the 'PC' command before loading a program with the 'PL' command.

Example:

```
PC
OK
```

Load Program: PL"(program text)"

Loads a program into the AI-7280's RAM. All programs generated by the TRsSim compiler are composed of printable ASCII characters, which must be enclosed in quotation marks when using this command. As the maximum length of any command must not exceed 128 characters, programs exceeding 123 characters should be broken up into multiple text strings. Each text string can be loaded in succession by using the 'PL' command. The maximum program size that can be loaded into the RAM is 16384 characters.

Example: this program enables the ringing generator for 2 seconds.

```
PL"TIN1HN111WIN2000TIN0HN111"
OK
```

Note: If the program text contains quotation marks, they must be duplicated before using the PL command.

Example: load the program: TIS"hello"GS1

```
PL"TIS""hello""GS1"
OK
```

Start Program from RAM: PS(n)M

Starts executing the program loaded into RAM. The value (n) must be between 1 and 6 and specifies which program execution unit (PEU) is used to execute the program. The AI-7280 can run up to 6 programs or processes at the same time.

Example: Start PEU #1 running a program loaded into RAM

```
PS1M
OK
```

Start Program from Flash: PS(n)F(file#)

Starts program execution from a file stored in the flash memory. The file to execute is identified by the integer (file#) parameter. As with the run from memory command, the value (n) specifies which PEU is used (valid range is from 1 to 6). Program files are loaded into the flash memory by using the TRsSim software.

Example: Run program #2 stored in the flash memory on PEU #1

```
PS1F2
OK
```

Halt Program Execution: PH(n)

This command halts program execution for the specified PEU. The value (n) specifies which the PEU to halt. If the value zero is passed for (n), then all PEUs are halted. To resume execution, use the program resume 'PR' command.

Example: Halt PEU #4

```
PH4
OK
```

Resume Program Execution: PR(n)

Resumes program execution. The value (n) specifies the PEU to resume. If the value zero is passed for (n), then all halted PEUs resume program execution.

Example: Resume PEU #4

```
PR4
OK
```

Stop Program Execution: PX(n)

Stops program execution. The value (n) specifies the PEU to stop. If the value zero is passed for (n), then all PEUs are stopped. Once a program is stopped, it cannot be resumed. Rather is must be started from the beginning again using the 'PS' command.

Example: Stop all programs

```
PX0
OK
```

Single Step: PT(n)

Single step program execution. The value (n) specifies the PEU to execute a single instruction. If the value zero is passed for (n), then all PEUs are single stepped. Only PEUs that have been halted can be single stepped. PEUs that are running or stopped will ignore this command.

Example: Execute a single instruction for PEU #1

```
PT1
OK
```

Set Program Load Address: PA(offset)

This command specifies a memory offset when loading programs into RAM with the 'PL' command. In situations where more than one program is to be loaded into the RAM, this command sets where additional programs are stored. The (offset) value must range from 0 to 4095 and specifies the number of 32 bit words used to offset the program memory.

Example: Load and run two programs in memory. The first program has a length of just under 200 characters.

```
PC
OK
PL"(program #1 text - first 100 characters)"
OK
PL"(program #1 text - next 100 characters)"
OK
PS1M
OK
PA50
OK
PL"(program #2 text - max. 123 characters at a time)"
OK
PS2M
OK
```

Use caution with the 'PA' command, since if used improperly it can cause programs to be overwritten in memory.

Note: While not shown in the examples above, every command sent to the AI-7280 must be terminated with the <CR> character (0Dh), and optionally may be followed by the <LF> character (0Ah). The <CR> character (0Dh) terminates every response returned by the AI-7280.

3. HAL Properties

3.1 Overview

The Hardware Abstraction Layer (HAL) properties represent a collection of registers that control all of the signal processing and hardware aspects of the AI-7280. By writing to or reading from various properties, the AI-7280 can be configured to perform a wide variety of tasks.

Each property is identified by a unique ID number and name. The ID number is required when accessing a property with the set '>' or get '?' command, while the name is used when accessing a property from within the TRsSim programming environment.

The HAL properties are organized in groups based on their function. The following table lists all of the groups with a brief description of their purpose. More detailed information on each of the properties is provided in the following sections.

Tone & Signal Generation Groups:

TONEA	Controls the generation of either a single frequency tone, FSK modulated signal, or AM modulated signal.
TONEB	Controls the generation of a single frequency tone, or can supply the modulation source if ToneA is set to AM mode.
TONEC	Controls the generation of a single frequency tone.
TONED	Controls the generation of a single frequency tone.
MFGEN	Simplifies generating DTMF or MF tones by providing access to a programmable tone table that specifies frequencies, levels, and timing for up to 20 different tones.
NOISE	Controls a broadband white noise generator.
DATA	Collection of properties used in defining a bit stream used by the FSK modulator.
FSKDROP	Controls the signal level during FSK modulation.

Telephone Line Interface Groups:

TELINT	Sets the telephone interface settings such a line voltage and loop current among others.
RING	Controls the ringing generator frequency, level, DC offset, and wave shape.
METERPULSE	Controls the metering pulse tone generator.
ECHO	Determines the level and delay of any simulated echoes for signals sent to the telephone interface.
DCPROFILE	Generates a user defined DC line voltage profile.

Signal Analysis & Measurement:

MEASURE	Collection of properties that return various measurements such as signal level, frequency, DC line voltage, and DC loop current.
FILTER	Controls the settings of optional filters applied before level and frequency measurements.
DTMF	Controls a DTMF level meter and frequency counter.
DTMFCAP	Collection of properties used to detect and capture DTMF digits.

FSK	Controls the FSK demodulator, which decodes a FSK signal into a stream of byte values.
SOURCE	Determines the signal source for measurement meters, DTMF detector, FSK decoder, and BNC output connector.

Waveform Capture & Playback & Transfer:

ACCAP	Controls the capture and playback of AC signals.
DCCAP	Controls the capture of DC line voltage and loop current measurements.
BULK	Collection of properties that simplify the transfer of AC or DC waveform data to and from the PC.

System Related Functions:

SYSTEM	Collection of properties controlling system related functions.
COMM	Controls the RS-232 serial port settings.
USB	Controls the USB port settings.
DIO	Determines the function of the rear panel digital Input/Output connector.
FILE	Collection of properties used to access files stored in the flash memory.
TIMER	Collection of properties that represent various timers used in the system.
SIGNALIO	Sets the gain of any signals routed to the BNC output connector.
STATUS	Properties returning status information on various AI-7280 activities.
SCHEDULER	Method to perform various operations at a predetermined future time.

Each property is of either a numeric or string data type. If a numeric property then its value can represent any single precision floating point value. Otherwise as a string property, its value is represented by a string of characters. For the string data types, the maximum length supported is 64 characters. The null character (0h) is used internally to mark the end of a string. As such, null (0h) cannot be used within a character string.

All of the properties in the following sections are listed in alphabetical order. For a listed by ID number, see: Appendix B: Property Listing by Index

Note: The property descriptions that follow assume an AI-7280 firmware version of at least 4.14. If an earlier version is used, some of the following properties may not be available. It is recommended that the AI-7280 be upgraded to the latest firmware version. A software package to update the firmware is available on the www.adventinsturments.com web site.

3.2 ACCAP Properties

These properties are used to capture AC signals and optionally play them back. Up to 229376 samples can be collected at either the full sample rate (39062.5 samples/s) or at the half sample rate (19531.25 samples/s) depending on the MODE property value. In either case, the samples are stored as 16 bits values representing a maximum range of +/- 10V. The samples are stored in an internal memory buffer with the INDEX property pointing to where in the buffer the next sample is to be stored. The signal source for the capture is determined by the SOURCE.ANALYZER property.

Samples stored in the buffer may be played back by using the PLAYINDEX and PLAYCOUNT properties. The MODE property affects the playback sample rate as well as the capture rate. If half rate playback is selected, then cubic interpolation is performed in order to up sample to the full system sample rate. The playback samples are summed in with the current tone and noise generators. By using the SAMPLE property, the captured samples may be read from the buffer or written into the buffer.

Note that a two sample delay is incurred for playback. For example, if playing 10 samples, the first two (at full rate) will be zero, followed by 8 samples stored in the capture buffer. When the playback stops, the last two samples are still stored in an internal buffer and not sent to the DAC. If playback is started with another 10 samples, the two remaining samples stored in the buffer are cleared.

Name:	ID:	Type:
ACCAP.COUNT	**185**	Numeric
Description: Sets the number of samples to capture. For example, setting this property to 10 captures 10 samples. As the samples are collected this property counts down to zero. Once at zero it stops the sampling process. As each sample is recorded the INDEX property increments. To stop a capture in progress, write the value 0 to this property. If the value -1 is written, the capture continues indefinitely.		

Name:	ID:	Type:
ACCAP.INDEX	**184**	Numeric
Description: Sets the position in the capture buffer where the next recorded sample is to be stored. As samples are collected, this property automatically increments by 1 for each sample. It also automatically rolls over to zero once the RECLOOPEND property value is exceeded. The maximum index value is 229375.		

Name:	ID:	Type:
ACCAP.MODE	**183**	Numeric
Description: Determines the capture and playback mode: 0 = capture and playback at full rate (39062.5 sampler/s) at 16 bits/sample 1 = capture and playback at half rate (19531.25 samples/s) at 16 bits/sample		

Name:	ID:	Type:
ACCAP.PLAYCOUNT	**189**	Numeric
Description: Sets the number of samples to playback. Writing a value greater than zero starts playback of the specified number of samples. The first sample is taken from the position in the buffer determined by PLAYINDEX. As each sample is output, this property decrements by one until it reaches zero. Once it reaches zero, playback stops. To stop a playback in progress, write the value 0. If a value if -1 is written, then playback continues indefinitely. Note that a two sample delay is incurred when playback is started.		

Name:	ID:	Type:
ACCAP.PLAYGAIN	**214**	Numeric
Description: Sets the gain applied to the waveform playback. The default value is 1.0. Using negative values result in a polarity inversion.		

Name:	ID:	Type:
ACCAP.PLAYINDEX	**188**	Numeric
Description: Sets or returns the position within the capture buffer where the next playback sample is read from. As samples are played back, this value automatically increments by 1. When it becomes greater than PLAYLOOPEND this property is reset to the value stored in PLAYLOOPSTART.		

Name:	ID:	Type:
ACCAP.PLAYLOOPEND	**213**	Numeric
Description: Sets the end position of a range of samples for playback. Once PLAYINDEX is greater than this value, PLAYINDEX is set to the PLAYLOOPSTART value. By default, PLAYLOOPEND is set to the maximum number of samples allowed in the buffer (229375).		

Name:	ID:	Type:
ACCAP.PLAYLOOPSTART	**212**	Numeric
Description: Sets the start position of a range of samples for playback. When PLAYINDEX becomes greater than PLAYLOOPEND, then PLAYINDEX is set to this value. By default, PLAYLOOPSTART is set to zero.		

Name:	ID:	Type:
ACCAP.RECLOOPEND	**213**	Numeric
Description: Sets the end position for a range of captured samples. When recording, if INDEX exceeds RECLOOPEND, then INDEX is set to 0. The default value is 229375.		

Name:	ID:	Type:
ACCAP.SAMPLE	**187**	Numeric
Description: Reading this property returns the sample value pointed to by the SAMPLEINDEX property. Writing to this property sets the value pointed to by the SAMPLEINDEX property. Either reading or writing to this property causes the SAMPLEINDEX property to increment by 1.		

Name:	ID:	Type:
ACCAP.SAMPLEINDEX	**186**	Numeric
Description:		
Sets the location in the capture buffer from where the SAMPLE property can either read or write data from/to. The range of this value is 0 to 229375.		

3.3 BULK Properties

The BULK properties are used to transfer data automatically between AC or DC waveform capture buffers and the PC via the COMM or USB ports. To initiate a transfer set the SOURCE and DEST properties to reflect the data source and destination respectively. If sending data to the PC, set the SPACE property to maintain a minimum amount of free space in the COMM or USB transmit buffers. SPACE is ignored if the COMM or USB port is not the destination. Set AUTOHALT to a non-zero value if command processing should be halted on either the COMM or USB port during the transfer. Finally, to start the transfer, set the LENGTH property to a non-zero value. As each data byte or sample is transferred from the source to the destination, the LENGTH property is decremented by one. The transfer is complete when LENGTH reads back as zero, or set to zero.

When reading or writing samples to the AC waveform capture buffers, each sample is represented by a 16 bit signed integer value. To convert between voltage and 16 bit 2's complement integer value, multiply or divide by 3276.8.

The 16 bit samples are always sent or received least significant byte (LSB) first.

The DC waveform capture buffer contains 24 bit records. Each record stores the line voltage and loop current measurements as signed 12 bit values. Bits 0 to 11 represent the 2's complement signed voltage measurement, while bits 12 to 23 represent the 2's complement signed current measurement. To convert between volts or milliamps to 12 bit 2's complement integers, multiply or divide by 10.

The 24 bit records stored in the DC waveform capture buffer are transferred least significant byte (LSB) first.

Name:	ID:	Type:
BULK.AUTOHALT	**194**	Numeric
Description:		
This property determines if normal command processing is suspended while a transfer is in progress. This is required when transferring data to the AC or DC capture buffers otherwise the command processor will interpret the incoming data bytes as commands. If AUTOHALT set to a non-zero value then when a transfer is started (by setting LENGTH to a value greater than zero), a check is made whether the COMM or USB ports are used in the transfer (either as source or destination). If either one is used (or both), then command processing is halted on that port (or both) by modifying the SYSTEM.HALTCMDS property. Once the transfer completes, the original value of SYSTEM.HALTCMDS is restored. Note, terminating a transfer by setting LENGTH to zero does not restore SYSTEM.HALTCMDS. It must be restored manually.		

Name:	ID:	Type:
BULK.DEST	**191**	Numeric
Description: Determines the destination of the data transfer as follows: 0 = AC Capture Buffer 1 = DC Capture Buffer 2 = COMM Port 3 = USB Port		

Name:	ID:	Type:
BULK.LENGTH	**192**	Numeric
Description: Sets the number of samples or bytes to transfer according to the SOURCE property. If the source is the AC capture or DC capture then this property represents the number of samples to transfer (2 bytes per AC sample, 3 bytes per DC sample). Otherwise if the source is either the COMM or USB port, then this property represents the number of bytes to transfer.		

Name:	ID:	Type:
BULK.SOURCE	**190**	Numeric
Description: Determines the source of the data transfer as follows: 0 = AC Capture Buffer 1 = DC Capture Buffer 2 = COMM Port 3 = USB Port		

Name:	ID:	Type:
BULK.SPACE	**193**	Numeric
Description: Sets the minimum amount of space to be maintained in the COMM or USB transmit data buffers. This value is only used if the COMM or USB ports are set as the data destination.		

Note: If the source is either the AC or DC capture buffers, then as the samples are read from the buffers the ACCAP.SAMPLEINDEX or DCCAP.READINDEX properties are incremented after each read. They will wrap to zero when the end of the buffer is reached. However if the destination is either AC or DC capture buffers, the ACCAP.SAMPLEINDEX or DCCAP.READINDEX properties are NOT incremented during the data transfer. This is because the source determines the number of bytes per transfer cycle, and if a COMM or USB port is the source only one byte is transferred at a time. A single byte is not a complete sample. When writing into the capture buffers, the initial index value is calculated from either the ACCAP.SAMPLEINDEX or DCCAP.READINDEX property at the moment the LENGTH property is set to a non-zero value and the transfer starts. From that point on, internal data is used to hold the address of the next byte to be transferred. Changing ACCAP.SAMPLEINDEX or DCCAP.READINDEX has no effect during the transfer.

Note: Version 2.02 of the firmware adds a timeout feature. HAL EXT integer #100 sets a timeout period (in units of 100 ms). When LENGTH is set to value greater than zero (starts a transfer), a timer is started with the timeout period. If the transfer is not complete when the timer reaches zero, the transfer is terminated. The LENGTH property is set to zero and if AUTOHALT is set to 1, the SYSTEM.HALTCMDS value is restored. This timer feature is disabled if the timeout is set to a value less than or equal to zero. By default, the timer is disabled.

3.4 COMM Properties

The following properties are used to control the RS-232 serial communications port, check its status, send data, and receive data. The system automatically performs two checks with the communication port. First, if a line break is detected it resets the baud rate to 9600 and clears the send and receive data buffers. A line break also clears bit 0 of the SYSTEM.HALTCMDS property. Second, if a parity, framing, or receive buffer overrun is detected, the entire receive buffer is flushed and the RXSTATUS property is set to a non-zero value.

Name:	ID:	Type:
COMM.BAUD	**27**	Numeric - w/o
Description: Sets the baud rate as follows: 0 = 9600 bps (default baud rate on power up or reset) 1 = 19200 bps 2 = 38400 bps 3 = 57600 bps 4 = 115200 bps The baud rate change is immediate once the property is written to. As such, using the set data '>' command to change the baud rate, causes the response 'OK' to be returned at the new baud rate.		

Name:	ID:	Type:
COMM.CTS	**34**	Numeric - w/o
Description: Controls the state of the CTS pin on the RS-232 connector. Is asserted by writing a non-zero value and de-asserted by writing zero.		

Name:	ID:	Type:
COMM.GETBYTE	**29**	Numeric - r/o
Description: Returns the next byte value (0 to 255) from the receive buffer. This property should only be read if RXCOUNT returns a value greater than zero.		

Name:	ID:	Type:
COMM.INIT	**26**	Numeric - w/o
Description: Writing any value to this property resets the communications port. The transmit and receive buffers are cleared and the baud rate is reset to 9600.		

Name:	ID:	Type:
COMM.RTS	**35**	Numeric - r/o
Description: Returns 1 if the RTS pin on the RS-232 connector is asserted. Otherwise zero is returned if the pin is not asserted.		

Name:	ID:	Type:
COMM.RXCOUNT	**28**	Numeric - r/o
Description: Returns the number of data bytes waiting in the receive buffer. Unless SYSTEM.HALTCMDS bit 0 is set, the command processor routines will automatically remove data bytes from the receive buffer.		

Name:	ID:	Type:
COMM.RXSTATUS	**32**	Numeric - r/o
Description: Returns a value representing the serial port status as follows: 0 = no error 1 = parity or framing error detected 2 = buffer overflow occurred 3 = line break detected		

Name:	ID:	Type:
COMM.SENDBYTE	**30**	Numeric - w/o
Description: Writing to this property sends a single byte to the transmit buffer. The TXFREE property should be checked prior to writing a byte, as there may not be any free space left in the buffer. Data bytes stored in the transmit buffer are transmitted at a speed determine by the BAUD property setting.		

Name:	ID:	Type:
COMM.SENDSTRING	**31**	String - w/o
Description: Similar to the SENDBYTE property, but instead sends a character string to the transmit buffer. The TXFREE property should be checked prior to sending the string, as there may not be any free space left in the buffer. Data bytes stored in the transmit buffer are transmitted at a speed determine by the BAUD property setting.		

Name:	ID:	Type:
COMM.TXFREE	**33**	Numeric - r/o
Description: This read only property returns the amount of free space left in the transmit buffer. It is important not to write any data bytes to the transmit buffer unless this property returns a value greater than the number of bytes to write.		

Note: The properties to receive data bytes should not be accessed unless the SYSTEM.HALTCMDS property has bit 0 set. Otherwise the command processor will automatically remove bytes from the receive data buffer.

3.5 DATA Properties

These properties are used to control the contents of the FSK transmit data buffer. This buffer is used when sending FSK modulated data. Any arbitrary pattern of up to 24576 bits can be constructed using these properties. See the TONEA properties for information on how to configure and enable the FSK modulator.

Name:	ID:	Type:
DATA.ADDALTERNATE	**124**	Numeric - w/o
Description: Writing to this property adds the specified number of bits to the FSK data bit buffer, in an alternating space/mark pattern. The first bit added is always a space (logic low). The valid range is from 0 to 24576.		

Name:	ID:	Type:
DATA.ADDBYTE	**125**	Numeric - w/o
Description: Writing to this property adds a serially encoded (LSB first) byte value to the FSK data buffer. The number of bits added to the buffer will be 9 plus the STOPBITS setting. The value must be in the range of 0 to 255. The PARITY setting has NO effect on this property.		

Name:	ID:	Type:
DATA.ADDCHAR	**126**	Numeric - w/o
Description: Similar to ADDBYTE, writing to this property adds a serially encoded (LSB first) byte value to the FSK data buffer. The number of bits added to the buffer will be 9 plus the STOPBITS setting. The value must be in the range of 0 to 255. Unlike the ADDBYTE property, the PARITY does affect the encoded bit pattern. If odd or even parity is enabled, the 8th data bit will be replaced by the parity bit.		

Name:	ID:	Type:
DATA.ADDHEXSTRING	**132**	String - w/o
Description: This property decodes the string of hexadecimal characters into byte values and adds them to the FSK data buffer. Each byte is encoded serially (LSB first). The PARITY setting has NO effect on the encoded bit pattern. If a non-hexadecimal character (0-9, A-F) is encountered, the encoding stops. The string must be an even number of characters; otherwise the last character is ignored. For example, writing "4AE52C" adds three bytes (left to right) with the value of 0x4A, 0xE5, and 0x2C to the FSK data buffer.		

Name:	ID:	Type:
DATA.ADDMARK	**122**	Numeric - w/o
Description: Writing to this property adds the specified number of mark bits (logic high) to the FSK data bit buffer. The valid range is from 0 to 24576.		

Name:	ID:	Type:
DATA.ADDPATTERN	**238**	Numeric - w/o
Description: Adds an arbitrary bit pattern to the FSK data buffer. Writing an integer to this property copies the value bit for bit to the FSK data buffer starting with the LSB. The number of bits copied ranges from 1 to 24 and is determined by the DATA.PATTERNLENGTH property. Note, the checksum value is not affected by writing to this property. Example 1: Data.PatternLength = 4 Data.AddPattern = 2 ‘ adds the four bits 0100 to the FSK buffer Example 2: Data.PatternLength = 8 Data.AddPattern = 1 ‘ adds the eight bits 1000 0000 to the FSK buffer		

Name:	ID:	Type:
DATA.ADDSPACE	**123**	Numeric - w/o
Description: Writing to this property adds the specified number of space bits (logic low) to the FSK data bit buffer. The valid range is from 0 to 24576.		

Name:	ID:	Type:
DATA.ADDSTRING	**127**	String - w/o
Description: Similar to ADDCHAR, writing to this property adds a string of characters to the FSK data buffer. Each character is encode with a start bit, data bits (LSB first), and one or more stop bits. The maximum number of characters that can be added is 64. As with ADDCHAR, the PARITY setting may affect the encoded bit pattern. The characters in the string are added to the FSK data buffer in a left to right order.		

Name:	ID:	Type:
DATA.ADDXSUM	**128**	Numeric - w/o
Description: Writing any value to this property adds the current checksum value to the FSK data buffer. The number of bits and format of the checksum depends on the XSUMTYPE property value.		

Name:	ID:	Type:
DATA.BITCOUNT	**240**	Numeric - r/o
Description: Returns the total number of bits stored in the FSK data buffer. Note: This property is effectively the same as TONEA.NUMBITS.		

Name:	ID:	Type:
DATA.BITINDEX	**241**	Numeric
Description: Sets an index into the FSK data buffer. The index is zero based and its value must range from 0 to the maximum number of bits supported (less one) (24576-1). The bit value accessed by the DATA.BITVALUE property uses this index property.		

Name:	ID:	Type:
DATA.BITVALUE	**242**	Numeric
Description: Sets or gets the bit value pointed to by the BITINDEX property. Writing a non-zero value sets the bit to one, while writing zero will clear the bit. If reading this property and the BITINDEX property is invalid (greater than DATA.BITCOUNT), then a value of -1 is returned. Any read or write access to this property causes the BITINDEX property to increment by one, provided BITINDEX is within zero and BITCOUNT less 1.		

Name:	ID:	Type:
DATA.CLEAR	**119**	Numeric - w/o
Description: Writing any value to this property clears the FSK data bit buffer. The number of bits in the buffer is determined by reading the TONEA.FSKNUMBITS property.		

Name:	ID:	Type:
DATA.DUPLICATE	**222**	Numeric
Description: This property is used to duplicate every bit added to the FSK data array EXCEPT for stop bits. Its value represents the number of times a non-stop bit is duplicated. For example, a setting of 2 duplicates every bit twice. Thus every call to DATA.ADD… generates three times the normal number of bits. By default, this property is set to zero. Its maximum value is 3. The purpose of this property is to support fractional stop bits. As an example, to generate 1.5 stop bits for every byte, set DUPLICATE = 1, STOPBITS = 3, and change the FSK baud rate to 2400 bps (double the nominal 1200 bps). Because every data bit is duplicated the effective baud rate is still 1200 bps for all the data bits; however the 3 stop bits become effectively 1.5 stop bits when viewed at the nominal 1200 baud rate.		

Name:	ID:	Type:
DATA.PARITY	**120**	Numeric
Description: This property controls the parity setting of any characters or strings added to the FSK bit buffer. The value range of settings is: 0 = 8 data bits per character, no parity; 1 = 7 data bits per character, odd parity; 2 = 7 data bits per character, even parity. The parity setting only affects data added to the buffer with the ADDCHAR and ADDSTRING properties.		

Name:	ID:	Type:
DATA.PATTERNLENGTH	**237**	Numeric
Description: Sets the number of bits which are added to the FSK buffer when the DATA.ADDPATTERN property is written to. The valid range is from 1 to 24 bits.		

Name:	ID:	Type:
DATA.STOPBITS	**121**	Numeric
Description: When bytes, characters, strings, or hex strings are added to the FSK bit buffer, they are encoded in a serial format which starts with a start bit (space), followed by 8 data/parity bits (LSB first), and ends with one for more stop bits (mark). The number of stop bits added is set by this property, and can range from 1 to 200.		

Name:	ID:	Type:
DATA.STOPBITVALUE	**239**	Numeric
Description: Sets the value used for stop bits. This property can be either zero or one. By default the stop bit value is 1. The stop bit value is used when writing to the ADDBYTE, ADDCHAR, ADDSTRING, ADDHEXSTR, ADDXSUM properties.		

Name:	ID:	Type:
DATA.XSUMENABLE	**129**	Numeric
Description: This property enables or disables the checksum calculations. Writing a non-zero value enables checksum calculations, while zero disables them. If enabled, the XSUMVALUE property is updated anytime the ADDBYTE, ADDCHAR, ADDSTRING, or ADDHEXSTRING properties are written too.		

Name:	ID:	Type:
DATA.XSUMTYPE	**130**	Numeric
Description: This property sets the method used to update the checksum value. Two settings are supported. They are: 0 = Inverted Modulus 256 (used with Telecordia, TIA, ETSI standards) 1 = 16 Bit CRC (x16+x12+x5+1) (used with NTT (Japan) FSK Caller ID)		

Name:	ID:	Type:
DATA.XSUMVALUE	**131**	Numeric
Description: The current value of the checksum can be read or modified by accessing this property. Depending on the XSUMTYPE settings, this value ranges from either 0 to 255 (8 bit), or 0 to 65535 (16 bit). Before using the checksum counter for a Caller ID or SMS message, set this property to zero.		

3.6 DCCAP Properties

The AI-7280 has the ability to sample the telephone interface line voltage and loop current values and then store the measurements into a data buffer. These properties are used to control the sampling process. Up to 8191 voltage and current measurements may be stored to a buffer. The sampling rate of the AI-7280 is fixed at 1000 samples per second.

Name:	ID:	Type:
DCCAP.COUNT	**168**	Numeric
Description: Sets the number of samples to collect. For example, setting this property to 10 causes the collection of 10 voltage and current readings at a rate of 1000 samples/s. As the samples are collected this property decrements down to zero. Once at zero it stops the sampling process. To stop a capture in progress, write the value 0. If the value of -1 is written to the COUNT property, the capture continues indefinitely.		

Name:	ID:	Type:
DCCAP.CURRENT	**171**	Numeric - r/o
Description: Returns the current sample (in units of mA) from the buffer location specified by READINDEX.		

Name:	ID:	Type:
DCCAP.INDEX	**167**	Numeric
Description: Sets the position in the capture buffer when the next measured sample is to be stored. This can range from 0 to 8191. As samples are collected, this property automatically increments by 1 for each sample. It will reset at 8191 to zero on the next sample captured.		

Name:	ID:	Type:
DCCAP.HEXSTRING	**172**	String - r/w
Description: Returns a 6 character hexadecimal string representing the voltage and current readings from the buffer location specified by READINDEX. The first three nibbles is the current reading while the last three nibbles are the voltage reading. Each group of three nibbles represents a 12 bit value (signed - using 2's complement). To convert the signed 12 bit integer values to volts or mA, multiply by 0.1.		

Name:	ID:	Type:
DCCAP.READINDEX	**169**	Numeric
Description: Sets where in the capture buffer the voltage and current readings are read from. The valid range is from 0 to 8191.		

Name:	ID:	Type:
DCCAP.VOLTAGE	**170**	Numeric - r/o
Description: Returns the voltage sample (in units of Volts) from the buffer location specified by READINDEX.		

3.7 DCPROFILE Properties

These properties are used to generate complex line voltage profiles. Line voltage sample points can be stored in a buffer and then played back at a specified rate. The buffer can hold up to 229376 sample points. This is the same buffer used for recording or playing AC waveforms (ACCAP properties). As such it is important to prevent conflicts when using these properties in conjunction with AC or DC signal captures.

The playback rate is adjustable between a range of 1 sample/sec to 5000 samples/sec. The resolution in sample time is 200 micro-seconds. As such, all playback rates are rounded to fit the nearest 200 micro-second interval.

To play out samples, set INDEX to the first sample offset into the buffer. Then set COUNT to a positive value representing the number of samples to output. Alternatively, set COUNT to a negative value to play out continuously. Once COUNT is written to the first sample point is read and the line voltage adjusted. The INDEX property increments and COUNT decrements (if positive) for each sample played out. As long as COUNT is non-zero, the next sample is output after the time (1/RATE) has elapsed.

During play out, if the INDEX value exceeds LOOPEND, the INDEX value is changed to LOOPSTART. If INDEX reaches the end of the buffer it automatically resets to zero.

The scaling factor used to convert between the 16 bit samples in the buffer and line voltage is 1/100. Accordingly, when writing samples into the buffer the desired voltage must be multiply by 100 in order to calculate the 16 bit sample value.

The play out of samples operates in one of two different modes as set by the MODE property. They are:

MODE = 0 (standard)

During play out, sample values read from the buffer are used to change both the TELINT.VOLTAGE and RING.DCLEVEL properties. The valid range for the samples is -72 to +72 volts. When the polarity of the samples changes (positive to negative or negative to positive) the telephone interface polarity is toggled).

When the play out of samples stops the line voltage setting and ringing offset setting are not restored to any prior values. They remain at the last sample read from the buffer.

Note, if the AI-7280 contains the Extended Line Feed Option (AI-E702) the range of voltages is increased to +/-105 volts.

MODE = 1 (high voltage / ringing)

This mode configures the AI-7280 to use its high voltage ringing amplifier for generating the DC voltage profile. This extends the output voltage from -170V to +80V, which is useful in simulating unusual line voltage profiles that including ringing.

Please note the following conditions when using this mode.

In this mode, the voltage samples stored in the buffer represent an offset to the current line voltage setting. The TELINT.VOLTAGE property is not changed in this mode of operation.

When using this mode, please note the following:

- The last sample value is always ignored because immediately after outputting the value and COUNT decrements, the line voltage is returned to the value prior to starting the play out.

- Ensure the ring generator (RING.ENABLE) stays off and remains off for the duration of play out.
- Ensure all tone generation is off (tones, noise, waveform playback) and remains off for the duration of play out.
- The off-hook detection function does not operate when the line voltage is positive.
- The ring trip detection function does not operate in this mode.
- Do not access the DCPROFILE.VOLTAGE property during play out.

Older AI-7280's will not support this mode of operation due to a hardware limitation. The MODE property will always return a zero when read. The hardware limitation can be removed if the unit is returned for calibration.

Name:	ID:	Type:
DCPROFILE.COUNT	**219**	Numeric
Description: Sets the number of samples left to play out. A positive value starts play out while a negative value enables continuous play out. Setting a value of zero stops any play out in progress. As samples are playing, this value decrements (if positive).		

Name:	ID:	Type:
DCPROFILE.INDEX	**216**	Numeric
Description: Index position used to access or play out a sample in the buffer. This value can range from zero to 229375. All read and writes to the VOLTAGE property increment the INDEX property. When play out is running (COUNT not equal to zeor) the index automatically resets to zero if the end of the buffer is reached, or is set to LOOPSTART if LOOPEND is reached.		

Name:	ID:	Type:
DCPROFILE.LOOPEND	**221**	Numeric
Description: Used to set the end of a play out range. When INDEX becomes greater than this value, it is set to LOOPSTART. By default LOOPEND is set to the maximum buffer index (229375).		

Name:	ID:	Type:
DCPROFILE.LOOPSTART	**220**	Numeric
Description: Used to set the beginning of a play out range. When INDEX becomes greater than LOOPEND, it is set to this value. By default LOOPSTART is set to zero.		

Name:	ID:	Type:
DCPROFILE.MODE	**243**	Numeric
Description: Sets the playback mode. The two modes are supported: 0: Standard 1: Ringing / high voltage playback Please note the precautions described in the beginning of this section when operating in mode 1.		

Name:	ID:	Type:
DCPROFILE.RATE	**218**	Numeric
Description: Sets the play out rate in samples per second. The valid range is from 1 to 5000 samples per second. Note the sample time is quantized to the closest multiple of 200 micro-seconds.		

Name:	ID:	Type:
DCPROFILE.VOLTAGE	**217**	Numeric
Description: Accesses the sample value at the current INDEX position in the buffer. Every read or write to this property increments INDEX. The sample value is quantized to 16-bits with a 0.01 voltage resolution.		

3.8 DIO Properties

The following properties are used to set the function of the three digital outputs (A, B, C) present at the rear panel of the AI-7280. In addition, the state of the two digital inputs (A, B) can be read by the INA and INB properties.

Name:	ID:	Type:
DIO.INA	**145**	Numeric - r/o
Description: Returns state of digital input A. 0=logic low, 1=logic high.		

Name:	ID:	Type:
DIO.INB	**146**	Numeric - r/o
Description: Returns state of digital input B. 0=logic low, 1=logic high.		

Name:	ID:	Type:
DIO.OUTA	**142**	Numeric
Description: Sets state or function of digital output A. Note that the digital output is an open collector configuration with a 5100 ohm pull up resistor to +5 volts. 0 = fixed at logic low 1 = fixed at logic high 2 = hook status (logic high if off-hook).		

Name:	ID:	Type:
DIO.OUTB	**143**	Numeric
Description: Sets state or function of digital output B. Note that the digital output is an open collector configuration with a 5100 ohm pull up resistor to +5 volts. 0 = fixed at logic low 1 = fixed at logic high 2 = FSK decoder output (logic high for mark, logic low for space).		

Name:	ID:	Type:
DIO.OUTC	**144**	Numeric
Description: Sets state or function of digital output C. Note that the digital output is an open collector configuration with a 5100 ohm pull up resistor to +5 volts. 0 = fixed at logic low 1 = fixed at logic high		

3.9 DTMF Properties

These properties are used to access the DTMF signal analyzer. If enabled, the signal analyzer divides the incoming signal into low and high frequency paths using band splitting filters. It then measures the frequency and level of each signal path. If both low and high group signal levels are above the set minimum threshold and both frequencies are within the specified tolerance of a DTMF digit, the DIGIT property is set to code representing the DTMF digit.

Note that the signal analyzer provides no de-bouncing for transient signals. Erroneous codes may be detected at the beginning and end of digits as the signal rises and falls. In addition speech and other signals may be briefly decoded as DTMF digits. Use the DTMFCAP properties to perform additional timing checks and buffer of digit detection.

The signal source used by the DTMF analyzer is determined by the SOURCE.ANALYZER property value.

Name:	ID:	Type:
DTMF.DIGIT	**148**	Numeric - r/o

Description:

Returns a code representing a detected DTMF digit. The return values are as follows:

0 = no digit detected

1-9 = DTMF digit 1 to 9 respectively

10 = DTMF digit 0

11 = DTMF digit * (star)

12 = DTMF digit # (pound)

13-16 = DTMF digit A to D respectively

In order for a non-zero digit code both the low and high group level must exceed the MINLEVEL property and the measured frequency must be within the frequency tolerance set by FREQTOL. This property is updated every time a frequency measurement is completed. The interval between frequency measurements is set by FREQTIME.

Name:	ID:	Type:
DTMF.ENABLE	**147**	Numeric

Description:

If set to a non-zero value, the DTMF detector is enabled. If zero, then no more measurements are performed and the current measurement values are frozen. The value returned by the DIGIT property is frozen as well.

Name:	ID:	Type:
DTMF.FREQTIME	**150**	Numeric

Description:

Sets the frequency measurement interval (in units of ms). The valid range is from 2 to 20. As a general rule, the longer the measurement interval the more stable and accurate the frequency measurements become.

Name:	ID:	Type:
DTMF.FREQTOL	**149**	Numeric

Description:

Sets the frequency tolerance of the DTMF detector (in percentage). The valid range is from 0 to 2.0. Both the low and high group tones must be within a frequency window set by this parameter in order for a digit to be detected.

Name:	ID:	Type:
DTMF.HIGHFREQ	**154**	Numeric - r/o

Description:

Returns the last frequency measurement for the high group (column) tone in units of Hz.

Name:	ID:	Type:
DTMF.HIGHLEVEL	**155**	Numeric - r/w
Description: Returns the current level measurement for the high group (column) tone. Units are Vrms.		

Name:	ID:	Type:
DTMF.LOWFREQ	**152**	Numeric - r/o
Description: Returns the last frequency measurement for the low group (row) tone in units of Hz.		

Name:	ID:	Type:
DTMF.LOWLEVEL	**153**	Numeric - r/o
Description: Returns the current level measurement for the low group (row) tone. Units are Vrms.		

Name:	ID:	Type:
DTMF.MINLEVEL	**151**	Numeric
Description: Sets the minimum RMS level that both the low and high group tones must exceed before a valid DTMF digit is detected. The level is specified in Vrms.		

3.10 DTMFCAP Properties

The DTMF capture properties are used to de-bounce and qualify digits detected by the DTMF signal analyzer. For all valid digits detected, it stores the frequency, level, and timing measurements into a buffer. The DTMF.ENABLE property must be set to non-zero for the capturing to occur. Up to 63 digits can be captured before the buffer becomes full. Once the buffer is full, any subsequent digits are lost.

To qualify as a valid DTMF digit, the digit's frequency must match the same DTMF digit three times consecutively. Additionally, the total signal power measured must be less than twice the low and high group tones combined. This improves the talk off performance of the detector. If these conditions are met, then the current frequency and level readings are stored to the buffer. The start time of the digit is also recorded into the buffer. The digit stop time is recorded into the buffer once the digit ends. Reading the stop time prior to the end of a digit returns a value of less than or equal to zero.

At the end of each detected digit, the NUMDIGITS property increments.

The INDEX property specifies where in the buffer digits are read from. Zero represents the first (or oldest digit) digit, while 1 the next digit, and so on. Digits can be removed from the buffer by writing to the DELETE property. Digits are always removed starting from the oldest detected digit (INDEX zero).

Name:	ID:	Type:
DTMFCAP.DELETE	**174**	Numeric - w/o
Description:		
Sets how many captured DTMF digits are to be deleted from the data buffer. Writing zero does not delete any records. To ensure that all digits stored are removed from the data buffer write a value of at least 63 (the maximum number of digits that can be stored).		

Name:	ID:	Type:
DTMFCAP.DIGIT	**176**	Numeric - r/o
Description:		
Returns the DTMF digit code for the captured digit pointed to by the INDEX property. If the digit code is zero, then no data exists. The codes are the same as defined for the DTMF.DIGIT property		

Name:	ID:	Type:
DTMFCAP.INDEX	**175**	Numeric
Description:		
Sets how far back into the buffer to read from. The buffer is setup as a FIFO. If INDEX is zero, then the first (oldest) digit captured can be read. Setting INDEX to 1 reads the next captured digit and so on. Setting INDEX beyond the number of digits in the buffer will cause all the data read to be zero. The maximum value of this property is 62.		

Name:	ID:	Type:
DTMFCAP.HIGHFREQ	**180**	Numeric - r/o
Description:		
Returns the high group tone frequency in Hz for the digit specified by the INDEX property.		

Name:	ID:	Type:
DTMFCAP.HIGHLEVEL	**178**	Numeric - r/o
Description:		
Returns the high group tone level in Vrms for the digit specified by the INDEX property.		

Name:	ID:	Type:
DTMFCAP.LOWFREQ	**179**	Numeric - r/o
Description:		
Returns the low group tone frequency in Hz for the digit specified by the INDEX property.		

Name:	ID:	Type:
DTMFCAP.LOWLEVEL	**177**	Numeric - r/o
Description:		
Returns the low group tone level in Vrms for the digit specified by the INDEX property.		

Name:	ID:	Type:
DTMFCAP.NUMDIGITS	**173**	Numeric - r/o
Description: Returns the number of DTMF digits stored in the capture buffer. This value can range from 0 to 63. Note that it is only when a DTMF digit ends that this property is incremented.		

Name:	ID:	Type:
DTMFCAP.STARTTIME	**181**	Numeric - r/o
Description: Returns the start time of the digit. The start time is defined as when both the low and high group tone signal level exceeds the minimum specified level set by DTMF.MINLEVEL. The time value is taken from the TIMER.SLOW property.		

Name:	ID:	Type:
DTMFCAP.STOPTIME	**182**	Numeric - r/o
Description: Returns the stop time of the digit. The stop time is defined as when either the low or high group tone level falls to less than 1/2 the signal power when the DTMF digit was detected. The time value is taken from the TIMER.SLOW property. If reading the buffer while the digit is still present, this value returns a value less than or equal to zero.		

3.11 ECHO Properties

The following properties are used to create transmit signal echoes. Up to three echoes can be created with independent delay and gain characteristics. If enabled, the echoes apply to any signal that is routed to the telephone interface for transmission.

Name:	ID:	Type:
ECHO.ENABLE	**205**	Numeric
Description: This property acts as a master enable or disable for the echo generator. If set to a non-zero value, then echoes are enabled. If set to zero, then echoes are disabled (regardless of the tap settings). By default, the value at power up is 1.0 (enabled), but the individual tap enables are set to zero (disabled).		

Name:	ID:	Type:
ECHO.RINGDISABLE	**206**	Numeric
Description: Controls the operation of the echo generator during ringing. If set to a non-zero value (enabled), then all echoes are disabled when ringing is turned on. Once ringing is turned off, the echoes are re-enabled. If this setting is enabled while ringing is currently active, it will not have any effect until the ringing is turned off and then turned on again. Note, this property only has an effect if the ENABLE property is non-zero.		

Name:	ID:	Type:
ECHO.TAPDELAY	**157**	Numeric
Description: Sets the echo time delay for the echo tap specified by TAPINDEX. The delay value is in units of ms with a valid range of 0 to 25.		

Name:	ID:	Type:
ECHO.TAPGAIN	**158**	Numeric
Description: Sets the echo gain (or loss) for the echo tap specified by TAPINDEX. The gain can range from -100 to +100. A setting of zero turns off the tap. Negative gains invert the polarity of the echo and gains between less than 1 or more than -1 represent a loss.		

Name:	ID:	Type:
ECHO.TAPINDEX	**156**	Numeric
Description: Sets which of the three possible echo taps are adjusted using the TAPDELAY and TAPGAIN properties. The valid range is from 1 to 3.		

3.12 FILE Properties

The FILE properties allow access to the non-volatile flash memory contained within the AI-7280. The flash memory contains a collection of files used to store system level programs and data along with user programs. Only read access to the flash memory is allowed by these properties.

Each file is referenced by a 32 bit identification number. Within each file, is a collection of items, each referenced by an integer value. These items represent file data such as numeric values or character strings. By using the following properties, the items of a file can be read.

Name:	ID:	Type:
FILE.EXIST	**21**	Numeric - r/o
Description: Returns a value of 1 if a file exists in the flash memory with the same file ID as set by the IDHIGH and IDLOW properties. If no file exists with that file ID, 0 is returned.		

Name:	ID:	Type:
FILE.IDHIGH	**20**	Numeric
Description: Sets the high 16 bits (0 to 65535) of the 32 bit flash file ID code. Normally these bits are used to determine the type of file as follows: 1 = System application file (DSP firmware) 2 = Parameter data file (contains calibration data) 3 = Script program file		

Name:	ID:	Type:
FILE.IDLOW	**19**	Numeric
Description: Sets the low 16 bits (0 to 65535) of the 32 bit flash file ID code.		

Name:	ID:	Type:
FILE.ITEMID	**22**	Numeric
Description: Used in reading a file item. Set this property to the item ID number of the desired item. Read the ITEMTYPE property to check if it exists within the file.		

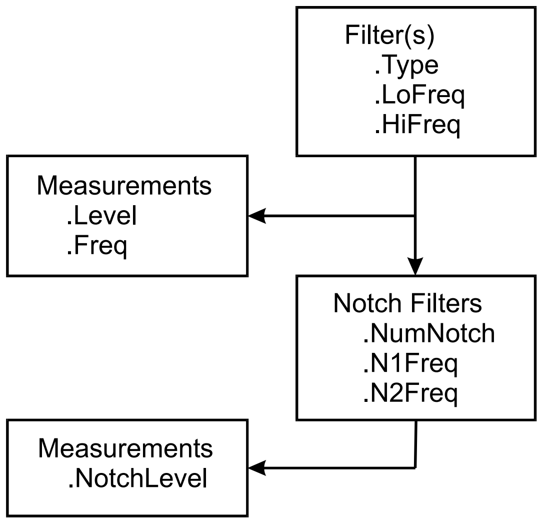
Name:	ID:	Type:
FILE.ITEMNUMBER	**24**	Numeric - r/o
Description: Returns the value of a numeric file item referenced by the ITEMID property. The value returned is only valid if the item type (as returned by ITEMTYPE) is either 1 or 2.		

Name:	ID:	Type:
FILE.ITEMSTRING	**25**	String - r/o
Description: Returns a string of characters for the file item referenced by the ITEMID property. The value returned is only valid if the item type (as returned by ITEMTYPE) is either 3 or 4. Note that for type 4 (big string), only up to the first 64 characters are returned.		

Name:	ID:	Type:
FILE.ITEMTYPE	**23**	Numeric - r/o
Description: Returns the type of item specified by the ITEMID property, for the file specified by IDLOW and IDHIGH. The possible return values are as follows: 0 = item does not exist in file 1 = item is an integer value (use ITEMNUMBER to read) 2 = item is a floating point value (use ITEMNUMBER to read) 3 = short string (limited to a max of 64 characters) (use ITEMSTRING to read) 4 = big string (unlimited number of characters) (use ITEMSTRING to read) 5 = binary data array (cannot be read)		

3.13 FILTER Properties

Optional filters may be placed before the various signal measurement meters. The following properties set the filter type, of which some may have adjustable corner or center frequencies. Up to two notch filters can be applied before the signal meter's secondary level meter. The following figure shows the relationship between the filter locations and the measurement points.



Name:	ID:	Type:
FILTER.HIFREQ	**75**	Numeric
Description: Sets the corner frequency for the 4th order high pass filter, or the center frequency for the second of two notch filters. The valid frequency range is from 20 to 10000 Hz. Note that writing to the TYPE property resets this value to 1000 Hz.		

Name:	ID:	Type:
FILTER.LOFREQ	**76**	Numeric
Description: Sets the corner frequency for the 4th order low pass filter, or the center frequency for the band pass or first of two possible notch filters. The valid frequency range is from 20 to 10000 Hz. Note that writing to the TYPE property resets this value to 1000 Hz.		

Name:	ID:	Type:
FILTER.N1FREQ	**78**	Numeric
Description: Sets the center frequency for the first of two secondary notch filters. The valid frequency range is from 20 to 10000 Hz. Note that writing to the NUMNOTCH property resets this value to 1000 Hz.		

Name:	ID:	Type:
FILTER.N2FREQ	**79**	Numeric
Description: Sets the center frequency for the second of two secondary notch filters. The valid frequency range is from 20 to 10000 Hz. Note that writing to the NUMNOTCH property resets this value to 1000 Hz.		

Name:	ID:	Type:
FILTER.NUMNOTCH	**77**	Numeric
Description: Sets the number of secondary notch filters placed before the second level meter (MEASURE.NOTCHLEVEL). This can range from 0 to 2. Writing to this property resets the N1FREQ and N2FREQ values to 1000 Hz.		

Name:	ID:	Type:
FILTER.TYPE	**74**	Numeric

Description:

Sets the type of filter to apply to all signal meters. Note that changing this property resets the LOFREQ and HIFREQ property values. The valid settings for this property are as follows:

0 = no filter

1 = 4th order Butterworth LPF (corner set by LOFREQ property)

2 = 4th order Butterworth HPF (corner set by HIFREQ property)

3 = both 1 and 2 above

4 = 4th order band pass (center set by LOFREQ property)

5 = notch (center set by LOFREQ property)

6 = dual notch (center set by both LOFREQ and HIFREQ property)

7 = DTMF low pass (rejects high group tones)

8 = DTMF high pass (rejects low group tones)

9 = C-message weighting filter

10 = ITU O.41 weighting filter

3.14 FSK Properties

The following collection of properties provides access to the FSK decoder. The decoder converts a FSK modulated data stream back into a series of bytes. The FSK signal is assumed to conform to either Bell 202 or V.23 standards with the data bytes encoded as 1 start bit, 8 data bits, and 1 or more stop bits. The data bytes must be sent LSB first. Up to 2047 data bytes can be decoded and stored in a buffer. To access the data stored in the buffer, use the INDEX, BYTEVALUE, and BYTESTATUS properties.

Name:	ID:	Type:
FSK.ACTIVE	**159**	Numeric

Description:

Enables the FSK decoder if set to a non-zero value. If set to zero all FSK decoding stops; however, any bytes stored in the buffer remain accessible.

Name:	ID:	Type:
FSK.BYTESTATUS	**165**	Numeric - r/o
Description: Returns the status value from the position in the FSK data byte buffer set by the INDEX property. The status value is an integer value from 0 to 3, defined as follows: 0 = byte received without framing or level errors 1 = byte received with a framing error (stop bit was not detected) 2 = signal level dropped during the byte (decode byte value may be incorrect) 3 = signal level dropped and framing error detected		

Name:	ID:	Type:
FSK.BYTEVALUE	**164**	Numeric - r/o
Description: Returns the byte value (0 to 255) from the position in the FSK data buffer specified by the INDEX property.		

Name:	ID:	Type:
FSK.COUNT	**162**	Numeric
Description: Returns the number of bytes decoded and stored in the FSK data buffer. After decoding a byte, the COUNT value is incremented by one. The valid range for COUNT is from 0 to 2047. Normally this value should be set to zero before enabling the FSK decoder. Once the first byte is received the COUNT value returns 1. If setting the COUNT property to N, the next byte received can be read using an INDEX value of N+1.		

Name:	ID:	Type:
FSK.INDEX	**163**	Numeric
Description: Determines from where in the FSK data buffer the decode byte value and status value are read from. The valid range is from 1 to 2047.		

Name:	ID:	Type:
FSK.LASTBYTE	**166**	Numeric - r/o
Description: Returns the last byte value decoded.		

Name:	ID:	Type:
FSK.LEVELTHRESHOLD	**160**	Numeric
Description: Sets the minimum acceptable FSK signal level in units of Volts peak. Signals below this level are not decoded.		

Name:	ID:	Type:
FSK.MARKTIME	**161**	Numeric
Description:		
Returns the number of seconds a mark tone (approximately 1200 to 1300 Hz) is present. If the signal frequency goes outside the mark window, the mark time is reset to zero. This measurement functions even when the FSK decoder is disabled.		

3.15 FSKDROP Properties

FSK generator signal level changes are supported by the FSKDROP properties. During the generation of a FSK modulated signal, the output level can be adjusted up to four times at various bit positions.

The following properties control access to an internal table storing FSK bit index's and gain values. The table contains four rows, with each row specifying when a level change will occur, and the amount of level change. To access the table, set the INDEX property to the table row to access (range is 1 to 4). Then use the BITINDEX and GAIN properties to read or write to the table data. When the FSK generator starts, a pointer is reset to the first row of the table. When the FSK generator bit index equals or exceeds the BITINDEX value in the table, then the current mark and space signal level is multiplied by the corresponding GAIN value in the table row. Following this change the pointer is incremented to the next row in the table. In this manner, up to four level changes may be programmed during the generation of a FSK modulated signal.

When using these properties, please note:

- All gain changes are made after the bit specified in the BITINDEX field has been generated. For example, a BITINDEX value of 5 means that after the 5th bit, the gain adjustment is applied. Not before. The minimum BITINDEX value is 1, thus the first bit is always generated at the default level. Level adjustments can only be made after the first bit.
- The list of BITINDEX values must be in ascending order, otherwise unexpected results may occur.
- The GAIN values must range between 0.0001 to 10000.0.
- Do not change the table entries while the FSK generator is active. This may cause unexpected results.

Name:	ID:	Type:
FSKDROP.BITINDEX	**198**	Numeric
Description:		
Sets the FSK bit position where the gain adjustment is applied. Note that the level adjustment is made AFTER the specified bit. The minimum value is 1.		

Name:	ID:	Type:
FSKDROP.CLEAR	**196**	Numeric - w/o
Description: Writing any value to this property causes a reset of the FSK drop-out table to default settings. The default values prevent any level FSK level adjustments by setting the BITINDEX values to a number much greater than the maximum number of bits allowed in a FSK generated signal.		

Name:	ID:	Type:
FSKDROP.GAIN	**199**	Numeric
Description: Sets the gain factor to be applied to the mark and space levels. The valid range is from 0.0001 to 10000.0.		

Name:	ID:	Type:
FSKDROP.BITINDEX	**197**	Numeric
Description: Sets the table row to access with the BITINDEX and GAIN properties. The valid range is from 1 to 4.		

3.16 MEASURE Properties

The MEASURE properties are used to return level, frequency, or phase measurements of a signal, or return the telephone interface line voltage and loop current measurements. Optional filters may be placed before the AC level and frequency measurements by using the FILTER properties. In addition, a secondary level measurement may be preceded by up to two programmable notch filters. These are normally used in performing THD+N measurements.

The telephone interface line voltage and loop current measurements are updated once every millisecond.

Name:	ID:	Type:
MEASURE.DCSMOOTHING	**70**	Numeric
Description: Sets the smoothing or averaging factor used for the LINEVOLT and LOOPCURR properties. The value must be between 0 and 1. A value of 0 performs no averaging and the properties return the last measurement. As the value is increased up to a maximum of 1, the measurements are progressively averaged with an exponential weighting of prior readings. A value of 1 freezes the readings.		

Name:	ID:	Type:
MEASURE.FREQ	**68**	Numeric - r/o
Description: Returns the current frequency measurement made of the signal following any filters enabled with the FILTER.TYPE property. The source of the signal is determined by the SOURCE.METER property.		

Name:	ID:	Type:
MEASURE.LEVEL	**67**	Numeric - r/o
Description: Returns the current RMS level measurement made of the signal following any filters enabled with the FILTER.TYPE property. The source of the signal is determined by the SOURCE.METER property. The SMOOTHING property sets the amount of averaging applied to this level measurement.		

Name:	ID:	Type:
MEASURE.LINEVOLT	**71**	Numeric - r/o
Description: Returns the current telephone interface line voltage measurement in units of volts. The DCSMOOTHING property sets the amount of averaging applied to this measurement.		

Name:	ID:	Type:
MEASURE.LOOPCURR	**72**	Numeric - r/o
Description: Returns the current telephone interface loop current measurement in units of mA. The DCSMOOTHING property sets the amount of averaging applied to this measurement.		

Name:	ID:	Type:
MEASURE.NOTCHLEVEL	**69**	Numeric - r/o
Description: Returns the current RMS level measurement of the signal following the secondary notch filters. The SMOOTHING property sets the amount of averaging applied to this level measurement. Normally this property is used in conjunction with the LEVEL reading when performing a THD+N measurement. See the FILTER properties for information on how to set the secondary notch filters.		

Name:	ID:	Type:
MEASURE.PHASE	**210**	Numeric - r/o
Description: Returns the phase difference (in degrees) between the reference signal and the meter's signal. The reference signal is selected by the SOURCE.PHASEREF property. The returned value ranges from -180 to +180 degrees. Positive phase values indicate that the signal is lagging the reference signal. Negative values represent leading phase. A single frequency sine wave is required for this measurement, with a frequency less than 10 kHz. The SMOOTHING property determines the amount of averaging applied to this measurement.		

Name:	ID:	Type:
MEASURE.PHASEDELAY	**211**	Numeric
Description: Sets the delay time compensation used for a phase measurement. If a fixed time delay exists between the phase reference signal and the meter's signal, this property can be used to compensate for the delay. The valid range is from -1.0 to +1.0 seconds.		

Name:	ID:	Type:
MEASURE.PHASELEVEL	**209**	Numeric - r/o
Description: Returns the RMS level of the phase reference signal, as selected by the SOURCE.PHASEREF property. The SMOOTHING property determines the amount of averaging applied to this measurement.		

Name:	ID:	Type:
MEASURE.SMOOTHING	**66**	Numeric
Description: Sets the smoothing factor used with the various RMS level meters. The valid range is from 0.5 to 0.99995. As the smoothing value is set closer to 1, the level meter's response time slows down as more exponential averaging is applied. As the response time increases the meter's readings become more stable.		

Name:	ID:	Type:
MEASURE.UNBALANCE	**73**	Numeric - r/o
Description: Returns a measurement of the telephone interface unbalance current flow in units of mA. A large unbalanced current normally indicates a ground loop problem. This measurement reading is updated once every 30 ms.		

3.17 METERPULSE Properties

The following properties control the operation of the metering pulse generator. The generator creates a sine wave signal of a specified frequency and amplitude. Its duration is programmable to millisecond resolution and is repeated with millisecond resolution. The output of this tone generator is NOT summed with the other tone/noise generators, but rather directly injected into the telephone interface transmit output path (but prior to echo generation). The generator can be programmed to produce either a specified number of pulses, or run indefinitely.

Name:	ID:	Type:
METERPULSE.COUNT	**200**	Numeric
Description: Sets how many pulses to generate. If set to a positive value, the generator begins to produce the specified number of pulses. After each pulse the COUNT value is decremented by one. When zero is reached pulse generation stops. If set to a negative value, it is clamped to -1 and pulse generation continues indefinitely.		

Name:	ID:	Type:
METERPULSE.DURATION	**203**	Numeric
Description: Sets the duration of the metering pulse tone burst in units of ms. The valid range is from 1 to 1,000,000 ms. To generate a continuous tone, set this value greater than the REPEAT value. If the DURATION is equal to the REPEAT value the generator produces a continuous tone; however, once the pulse duration has elapsed the tone generator's phase is reset to 0 degrees and the next pulse is started.		

Name:	ID:	Type:
METERPULSE.FREQ	**201**	Numeric
Description: Sets the frequency of the metering pulses between the limits of 10 Hz and 18000 Hz.		

Name:	ID:	Type:
METERPULSE.LEVEL	**202**	Numeric
Description: Sets the output level of the metering pulses in units of Vrms. The valid range is from 0 to 4 Vrms.		

Name:	ID:	Type:
METERPULSE.REPEAT	**204**	Numeric
Description: Sets the repeat interval for the metering pulses. The valid range is from 1 to 1,000,000 ms. For example, if set to 1000 ms, the metering pulses starts every 1000 ms with duration specified by the DURATION property.		

3.18 MFGEN Properties

The Multi-Frequency Generator properties implement a flexible DTMF or MF tone generator. It uses an internal data table containing frequency, level, and timing information for up to 20 arbitrary symbols. Each symbol represents either a single or dual tone signal with independent frequency, level, and timing. Symbols can be generated one at a time, or in a sequence.

Before enabling the MF generator ensure both tone generators C and D are disabled, as the MF generator takes a lower priority.

The following table shows the INDEX value required to access the frequency, level, and timing table for each of the 20 symbols.

Symbol Number	Symbol Character	Freq #1 (Hz)	Freq #2 (Hz)	Level #1 (Vrms)	Level #2 (Vrms)	Duration (ms)	Off Time (ms)
1	1	<1>	<2>	<3>	<4>	<5>	<1001>
2	2	<6>	<7>	<8>	<9>	<10>	<1002>
3	3	<11>	<12>	<13>	<14>	<15>	<1003>
4	4	<16>	<17>	<18>	<19>	<20>	<1004>
5	5	<21>	<22>	<23>	<24>	<25>	<1005>
6	6	<26>	<27>	<28>	<29>	<30>	<1006>
7	7	<31>	<32>	<33>	<34>	<35>	<1007>
8	8	<36>	<37>	<38>	<39>	<40>	<1008>
9	9	<41>	<42>	<43>	<44>	<45>	<1009>
10	0	<46>	<47>	<48>	<49>	<50>	<1010>
11	*	<51>	<52>	<53>	<54>	<55>	<1011>
12	#	<56>	<57>	<58>	<59>	<60>	<1012>
13	A	<61>	<62>	<63>	<64>	<65>	<1013>
14	B	<66>	<67>	<68>	<69>	<70>	<1014>
15	C	<71>	<72>	<73>	<74>	<75>	<1015>
16	D	<76>	<77>	<78>	<79>	<80>	<1016>
17	E	<81>	<82>	<83>	<84>	<85>	<1017>
18	F	<86>	<87>	<88>	<89>	<90>	<1018>
19	G	<91>	<92>	<93>	<94>	<95>	<1019>
20	H	<96>	<97>	<98>	<99>	<100>	<1020>

Note that the off-time column is only relevant when two or more symbols are generated. The off-time for a symbol is the time following that symbol when no signal is produced.

Name:	ID:	Type:
MFGEN.ACTIVE	**141**	Numeric

Description:

This property is used to enable or disable the MF tone generator. Writing a non-zero value starts the generator with either the last SYMBOL or last STRING value written to it. While the generator is active, reading the ACTIVE property returns the value 1. Once it has finished, the value 0 is returned.

Note, if either TONEC, TONED, or ringing is enabled, the MF generator cannot be started and will always return zero.

The MF tone generator uses the wave shape settings of tone generator C and tone generator D.

Name:	ID:	Type:
MFGEN.FREQADJUST	**136**	Numeric
Description: The FREQADJUST property provides a shortcut for setting the tone frequencies in the symbol table to the standard DTMF values. Writing to the property sets the frequencies for symbols 1 to 16 to the standard DTMF frequency, plus or minus the specified adjustment in percent. The allowable frequency adjustment range is from -20 % to +20%. The frequency table entries for symbols 17 to 20 are not affected.		

Name:	ID:	Type:
MFGEN.FREQADJUST1	**136**	Numeric - w/o
Description: Writing to the FREQADJUST1 property causes the frequency of tone #1 for the first 16 symbols to be adjusted by the specified percentage from their current value. For DTMF applications this represents the ROW tone frequencies.		

Name:	ID:	Type:
MFGEN.FREQADJUST2	**136**	Numeric - w/o
Description: Writing to the FREQADJUST2 property causes the frequency of tone #2 for the first 16 symbols to be adjusted by the specified percentage from their current value. For DTMF applications this represents the COLUMN tone frequencies.		

Name:	ID:	Type:
MFGEN.FREQOFFSET1	**136**	Numeric - w/o
Description: Writing to the FREQOFFSET1 property causes the frequency of tone #1 for the first 16 symbols to be adjusted by the specified value in units of Hz. For DTMF applications this represents the ROW tone frequencies.		

Name:	ID:	Type:
MFGEN.FREQOFFSET2	**136**	Numeric - w/o
Description: Writing to the FREQOFFSET2 property causes the frequency of tone #2 for the first 16 symbols to be adjusted by the specified value in units of Hz. For DTMF applications this represents the COLUMN tone frequencies.		

Name:	ID:	Type:
MFGEN.INDEX	**133**	Numeric
Description: This property provides access to the symbol data table. The data table consists of 120 entries (6 per symbol with 20 symbols). Each symbol has two entries for tone frequencies (in Hz), two for tone levels (in Vrms), one for the tone duration (in ms), and one for the off time following the symbol (in ms). To read or write set the INDEX property to the entry number and then use the VALUE property. See the previous table for the relationship between the INDEX property and the symbol table entries. The valid range of the INDEX property is from 1 to 100 or from 1001 to 1020.		

Name:	ID:	Type:
MFGEN.LEVEL	**135**	Numeric
Description: Writing to this property sets the level of both tones #1 and #2 for the first 16 symbols to the specified value (in units of Vrms). This acts as a shortcut to manually setting the symbol table levels to the same value. The valid range for the level is from 0 to 4.0 Vrms.		

Name:	ID:	Type:
MFGEN.LEVEL1	**230**	Numeric - w/o
Description: Writing to this property sets the level of tone #1 for the first 16 symbols to the specified value (in units of Vrms). For DTMF applications this is the ROW tone level. The valid range for the tone level is from 0 to 4.0 Vrms.		

Name:	ID:	Type:
MFGEN.LEVEL2	**231**	Numeric - w/o
Description: Writing to this property sets the level of tone #2 for the first 16 symbols to the specified value (in units of Vrms). For DTMF applications this is the COLUMN tone level. The valid range for the tone level is from 0 to 4.0 Vrms.		

Name:	ID:	Type:
MFGEN.OFFTIME	**138**	Numeric
Description: The OFFTIME property controls the time interval between generating multiple tone symbols. This only has an effect when generating multiple tones by using the STRING property. The units are in milliseconds. Writing to this property changes the off-time for all 20 symbols. If an off-time of zero is used, the phase of the tone generators is continuous at symbol transitions.		

Name:	ID:	Type:
MFGEN.ONTIME	**137**	Numeric
Description: Writing to this property sets the tone duration, for symbols 1 to 16, to the specified value, in units of milliseconds. This acts as a shortcut to manually setting the symbol table durations to a common value.		

Name:	ID:	Type:
MFGEN.RESET	**236**	Numeric - w/o
Description: Writing to this property sets both frequencies for symbols 1 to 16 to the standard DTMF values. Tone #1 is the ROW frequency, while Tone #2 is the COLUMN frequency.		

Name:	ID:	Type:
MFGEN.STRING	**140**	String
Description: The STRING property is used to generate a series of tone symbols. The 20 different symbols each correspond to a different ASCII character (0-9, *, #, A, B, C, D, E, F, G, H). After writing to this property, start the tone generator by writing a non-zero value to the ACTIVE property.		

Name:	ID:	Type:
MFGEN.SYMBOL	**139**	Numeric - w/o
Description: Writing to this property, specifies the next symbol to generate. The valid range for this property is from 1 to 20. Start the symbol by writing a non-zero value to the ACTIVE property. Note that the STRING property takes precedence over this property. Do not write to it before starting writing to ACTIVE.		

Name:	ID:	Type:
MFGEN.VALUE	**134**	Numeric
Description: The VALUE property works in conjunction with the INDEX property for access to the symbol data table. Once the INDEX value has been set to the desired table entry, the table entry value can be read or modified to by reading or writing to the VALUE property.		

3.19 NOISE Properties

A pseudo-random noise generator is controlled with the following two properties. The noise generator produces a flat spectral response up to approximately 18 kHz.

Name:	ID:	Type:
NOISE.ENABLE	**117**	Numeric
Description: The noise generator is enabled by writing a non-zero value, and disabled by writing a value of zero.		

Name:	ID:	Type:
NOISE.LEVEL	**118**	Numeric
Description: Sets the RMS output level of the noise generator. Its valid range is from 0 to 2.0 Vrms.		

3.20 RING Properties

The telephone interface ringing generator is controlled with the following properties.

It supports a wide range of programmable frequencies, levels, DC offsets, and wave shapes.

By default the ringing generator does not turn off when a connected Terminal Equipment (TE) device goes off-hook. Rather its output becomes muted. If the TE returns to the on-hook state the ringing resumes. This operation can be altered with the RING.TRIP property which causes the ring generator to turn off upon detecting an off-hook condition.

Note, the ringing generator can produce a maximum of 80 Vrms when using the "Sine" wave shape and a nominal DC offset voltage level of 48 Volts. If higher crest factor wave shapes or different DC offset voltages are selected, the ringing voltage may be clipped. The ringing amplifier output is constrained between -170 and +80 volts.

Name:	ID:	Type:
RING.DCLEVEL	**116**	Numeric
Description: Sets the DC offset voltage when ringing is enabled. When ringing is turned off the line voltage returns to the TELINT.VOLTAGE setting. This value can range from 0 to 72 volts. Note, if ringing is started with a DC offset voltage setting of 15 volts or less, the RING.TRIP property is automatically forced to 1.		

Name:	ID:	Type:
RING.ENABLE	**111**	Numeric
Description: Non-zero value enables the ringing generator. Turning on the ringing generator automatically turns off tones A, B, C, D, the MF generator, and the noise generator. When ringing is started, the waveform phase angle is automatically reset to zero. If a different starting phase is desired, set the phase immediately after setting the ENABLE property to a non-zero value. Note: Waveform playback is NOT stopped when ringing is started.		

Name:	ID:	Type:
RING.FREQ	**112**	Numeric
Description: Sets the ringing frequency from 10 Hz to 100 Hz.		

Name:	ID:	Type:
RING.LEVEL	**113**	Numeric
Description: Sets the ringing level from 0 Vrms to 80 Vrms.		

Name:	ID:	Type:
RING.PHASE	**114**	Numeric
Description: Reading this property returns the current ring generator phase angle between 0 and 360 degrees. Writing to this property sets the current phase angle. Note, if ringing is not active reading this property returns 0.		

Name:	ID:	Type:
RING.TRIP	**207**	Numeric
Description: Controls the state of the ringing generator when a connected TE goes off-hook. If zero (default value) the ringing generator is only muted when a TE is off-hook. Once the TE goes back on-hook ringing resumes. If this property is set to a non-zero value then the ringing generator turns off (RING.ENABLE=0) when an off-hook condition is detected. Note, if ringing is started with a DC offset voltage setting of 15 volts or less, this property is automatically forced to 1.		

Name:	ID:	Type:
RING.WAVESHAPE	**115**	Numeric
Description: Determines which wave shape is used for ringing. This can be set to any of the following values: 0 = sine 1 = triangle 2 = square 3 = user defined Note, the LEVEL property is only calibrated for the sine wave shape. For information on how to program the user defined wave shapes, please contact technical support.		

3.21 SCHEDULER Properties

These properties are used to specify actions performed at a future predetermined time. They manage a list of up to 50 actions where each action can either modify a HAL register or control the operation of a PEU.

The scheduler uses both the TIMER.ROLLCOUNT and TIMER.SLOW properties to determine when to trigger an action. Every action has an ATCOUNT value and an ATTIMER value. When TIMER.ROLLCOUNT is equal to ATCOUNT and TIMER.SLOW is equal to or greater than ATTIMER, then the action is triggered and it executes.

If multiple action records are created using the same ATCOUNT and ATTIMER values, the actions are executed in the order they were created.

Action records are created when writing to the ACTION property. The value written to ACTION determines the action taken. It must be either:

- A valid HAL register number. When the action triggers it performs a write to the HAL register with the value specified in the PARAMETER property.
- The value of 1000 which pauses execution of the PEU specified in the PARAMETER property.
- The value of 1001 which resumes execution of the PEU specified in the PARAMETER property.
- The value of 1002 which stops execution of the PEU specified in the PARAMETER property.

To confirm that an action was created read back the property value. If the value read back is -1 the action was not created. This can be due to:

- The value written to ACTION is not in the above list.
- The maximum number of actions has been created and no more can be added.
- The ATCOUNT property is set to a negative value.

Name:	ID:	Type:
SCHEDULER.ACTION	**245**	Numeric
Description: Writing a value to this property creates a new action using the current values of ATCOUNT, ATTIMER, and PARAMETER. A check is made to ensure: 1) The ACTION value is valid. 2) The ATCOUNT value is >= 0 3) There is space available for more actions. If any of these tests fail this property is set to -1. Reading back the property is used to confirm if the action was created.		

Name:	ID:	Type:
SCHEDULER.ATCOUNT	**246**	Numeric
Description: Sets the value that TIMER.ROLLCOUNT must be at to trigger an action. This property must be greater or equal to zero. If set to a negative value, writing to the ACTION property will not create a new action.		

Name:	ID:	Type:
SCHEDULER.ATTIMER	**247**	Numeric
Description: Sets the value that TIMER.SLOW must be greater or equal to in order to trigger an action.		

Name:	ID:	Type:
SCHEDULER.COUNT	**249**	Numeric - r/o
Description: Returns the number of actions waiting to be triggered. As actions are triggered this value counts down. It returns zero when all actions have been triggered.		

Name:	ID:	Type:
SCHEDULER.PARAMETER	**248**	Numeric
Description: Sets the parameter value used in conjunction with an action. If the action value is a HAL register, then this property specifies the value written to the HAL register when the action triggers. Otherwise if the action is to control a PEU, then this property specifies which PEU is affected. A value of zero affects all PEUs.		

Name:	ID:	Type:
SCHEDULER.RESET	**244**	Numeric - w/o
Description: Resets the scheduler. All actions that have been created and waiting for a trigger are erased. Reading COUNT returns zero.		

3.22 SIGNALIO Properties

The single property in this group sets the gain applied to any signals routed to the rear panel BNC output connector.

Name:	ID:	Type:
SIGNALIO.BNCOUTGAIN	**195**	Numeric
Description: Sets the gain of the BNC output port. It must range between -100 to +100. A negative value applies a 180 degree phase reversal to the signal. The source of the signal sent to the BNC output port is determined by the SOURCE.BNCOUT property value.		

3.23 SOURCE Properties

The following four properties select what signal source is routed to the signal meter, signal analyzer, rear panel BNC output, and the phase meter reference channel.

Name:	ID:	Type:
SOURCE.ANALYZER	**64**	Numeric
Description: Sets the source of the signal routed to the signal analyzer (DTMF, FSK Decoder, waveform capture) as follows: 0 = off 1 = telephone interface (both transmit and receive signals) 2 = telephone interface hybrid output (rejects signals transmitted by the AI-7280) 3 = rear panel BNC input connector 4 = tone generators (tone A,B,C,D, MF, and FSK generator, waveform playback) 5 = telephone interface transmit (signal applied to the telephone line by the AI-7280)		

Name:	ID:	Type:
SOURCE.BNCOUT	**65**	Numeric

Description:

Sets the source of the signal routed to the rear panel BNC output connector as follows:

0 = off

1 = telephone interface (both transmit and receive signals)

2 = telephone interface hybrid output (rejects signals transmitted by the AI-7280)

3 = rear panel BNC input connector

4 = tone generators (tone A,B,C,D, MF, and FSK generator, waveform playback)

5 = telephone interface transmit (signal applied to the telephone line by the AI-7280)

6 = input to signal meter (following optional filter)

7 = input to secondary level meter (following optional notch filters)

8 = output of signal analyzer (input to DTMF detector and FSK decoder)

9 = phase measurement reference channel (selected by SOURCE.PHASEREF)

Name:	ID:	Type:
SOURCE.METER	**63**	Numeric

Description:

Sets the source of the signal routed to the signal meter as follows:

0 = off

1 = telephone interface (both transmit and receive signals)

2 = telephone interface hybrid output (rejects signals transmitted by the AI-7280)

3 = rear panel BNC input connector

4 = tone generators (tone A,B,C,D, MF, and FSK generator, waveform playback)

5 = telephone interface transmit (signal applied to the telephone line by the AI-7280)

Name:	ID:	Type:
SOURCE.PHASEREF	**208**	Numeric

Description:

Sets the source of the reference signal when performing phase measurements. The possible selections are as follows:

0 = off

1 = telephone interface (both transmit and receive signals)

2 = telephone interface hybrid output (rejects signals transmitted by the AI-7280)

3 = rear panel BNC input connector

4 = tone generators (tone A,B,C,D, MF, and FSK generator, waveform playback)

5 = telephone interface transmit (signal applied to the telephone line by the AI-7280)

3.24 STATUS Properties

The two status properties are read-only and return integer values where each bit position represents the operational state of the AI-7280.

Name:	ID:	Type:
STATUS.A	**223**	Numeric - r/o

Description:

Returns an integer with each bit representing the following:

Bit 0: Set if noise generator enabled

Bit 1: Set if ToneA generator enabled

Bit 2: Set if ToneB generator enabled

Bit 3: Set if ToneC generator enabled

Bit 4: Set if ToneD generator enabled

Bit 5: Set if FSK Generator is active

Bit 6: Set if MF/DTMF generator active

Bit 7: Set if ringing generator is active

Bit 8: Set if AC Playback is active

Bit 9: Set if DC profile playback is active

Bit 10: Set if meter pulse generator is active

Name:	ID:	Type:
STATUS.B	**224**	Numeric - r/o
Description: Returns an integer with each bit representing the following: Bit 0: Set if system error flags register is non-zero Bit 1: Set if hook detect indicates off-hook Bit 2: Set if DC capture is active Bit 3: Set if AC capture is active Bit 4: Set if digital input A is high Bit 5: Set if digital input B is high Bit 6: Reserved – returns zero Bit 7: Reserved – returns zero Bit 8: Reserved – returns zero Bit 9: Set if PEU #1 status is NOT zero Bit 10: Set if PEU #2 status is NOT zero Bit 11: Set if PEU #3 status is NOT zero Bit 12: Set if PEU #4 status is NOT zero Bit 13: Set if PEU #5 status is NOT zero Bit 14: Set if PEU #6 status is NOT zero Bit 15: Set if any PEU is in the ERROR state		

3.25 SYSTEM Properties

The following properties perform a wide range of system related tasks. These included returning firmware version information, controlling error conditions, and managing system interrupt flags.

Name:	ID:	Type:
SYSTEM.ERRORGET	**13**	Numeric - r/o
Description: Reads a value from the system error stack. If zero is read then the error stack is empty and the READY led is returned to solid on from a possible flashing state (if serious errors were recorded). For a listing of standard error codes, see Appendix A of the AI-7280 User Guide.		

Name:	ID:	Type:
SYSTEM.ERRORSET	**12**	Numeric - w/o
Description: Writes a value to the system error stack. Depending on the value written, the READY led may start to flash indicating a serious error. For a listing of standard error codes, see Appendix A of the AI-7280 User Guide.		

Name:	ID:	Type:
SYSTEM.FFSID	**4**	String - r/o
Description: Returns a string representing the flash file format currently used by the firmware. The version numbers are enclosed in [ ] brackets.		

Name:	ID:	Type:
SYSTEM.FLAGCLEAR	**16**	Numeric - w/o
Description: Clears one or more interrupt bit(s). By writing an integer value to this property, each bit set clears the corresponding bit in the system interrupt register. The system interrupt register bits are defined as follows: Bit 0: Off-Hook Detect: Set when the loop current rises above the off-hook threshold. Note that it is delayed by the de-bounce period used to qualify the hook status. Bit 1: On-Hook Detect: Set when the loop current falls below the off-hook threshold. Note that it is delayed by the de-bounce period used to qualify the hook status. Bit 2: Digital Input A: Set when the rear panel digital input pin changes state. Note that pulses shorted than 200 us may not be detected. Bit 3: Digital Input B: Set when the rear panel digital input pin changes state. Note that pulses shorted than 200 us may not be detected. Bit 4: DTMF Detect: Set when the DTMF.DIGIT property changes from zero, to a non-zero value. Bit 5: DTMF Capture Start: Set when the DTMFCAP properties have detected a valid DTMF digit. Bit 6: DTMF Capture End: Set when the DTMFCAP properties detect the end of a valid DTMF digit. Bit 7: Timer: Set when TIMER.INTPERIOD counts down to zero.		

Name:	ID:	Type:
SYSTEM.FLAGGET	**14**	Numeric - r/o
Description: Returns the current system interrupt register value. See FLAGCLEAR for a listing of all assigned interrupt bits.		

Name:	ID:	Type:
SYSTEM.FLAGSET	**15**	Numeric - w/o
Description: Sets one or more interrupt bit(s). By writing an integer value to this property, each bit set in the value sets the corresponding bit in the system interrupt register. See FLAGCLEAR for a listing of all assigned interrupt bits.		

Name:	ID:	Type:
SYSTEM.HALID	**3**	String - r/o
Description: Returns a string representing the hardware abstraction layer currently used by the firmware. The version numbers are enclosed in [ ] brackets.		

Name:	ID:	Type:
SYSTEM.HALTCMDS	**17**	Numeric
Description: Enables or disables command processing from either the serial RS-232 port or the USB port according to the following values: 0 = Command processing from both serial and USB is enabled 1 = Command processing from serial port is disabled 2 = Command processing from USB port is disabled 3 = Command processing from both ports is disabled. Note, if a line break is detected on the serial port then command processing will be resumed (if currently disabled). If the USB port goes into reset or is un-plugged, then command processing is resumed (provided the USB port status indicates it is enumerated by the PC).		

Name:	ID:	Type:
SYSTEM.OPTIONS	**18**	Numeric - r/o
Description: Returns an option code. The code may represent hardware or software options that are enabled or present. At this time no options are defined.		

Name:	ID:	Type:
SYSTEM.RESET	**11**	Numeric - w/o
Description: Writing any value to this property causes a reset of the AI-7280. All properties are returned to their power up default value.		

Name:	ID:	Type:
SYSTEM.SOFTID	**2**	String - r/o
Description: Returns a string containing the device model number and firmware version enclosed in [ ] brackets. Following the version number is the firmware program identifier enclosed in ( ) brackets. For example, reading this property returns: "AI-7280 Software Version [4.14](1)"		

Name:	ID:	Type:
SYSTEM.SUBFUNCTION	**10**	Numeric - w/o
Description: Writing a value performs a low level system call. Do not use.		

Name:	ID:	Type:
SYSTEM.SUBINDEX	**6**	Numeric
Description: Accesses a low level system call parameter. Do not use.		

Name:	ID:	Type:
SYSTEM.SUBINTEGER	**8**	Numeric
Description: Accesses a low level system call parameter. Do not use.		

Name:	ID:	Type:
SYSTEM.SUBSTRING	**9**	String
Description: Accesses a low level system call parameter. Do not use.		

Name:	ID:	Type:
SYSTEM.SUBVALUE	**7**	Numeric
Description: Accesses a low level system call parameter. Do not use.		

Name:	ID:	Type:
SYSTEM.UNITID	**1**	String - r/o
Description: Returns a 16 hexadecimal value representing the device's unique ID code.		

Name:	ID:	Type:
SYSTEM.VTPID	**5**	String - r/o
Description: Returns a string representing the program execution units (PEU) currently used by the firmware. The version numbers are enclosed in [ ] brackets.		

3.26 TELINT Properties

The TELINT properties control the settings of the AI-7280's telephone interface. This includes the line voltage, loop current, and line impedance. Additional properties determine the how signals are routed to the telephone interface and how signals are measured from the telephone interface.

Name:	ID:	Type:
TELINT.BALANCE	**57**	Numeric
Description: Sets the line balance impedance. This impedance is used as part of the internal hybrid which separates the transmit signals from the receive signals on the telephone line. For maximum rejection of transmit signals it should be set to match the impedance of the connected terminal equipment device. The following settings are available: 0 = 600 ohms; 1 = 900 ohms; 2 = TBR-21; 3 = optional impedance (only available if installed in the AI-7280)		

Name:	ID:	Type:
TELINT.BNCINGAIN	**59**	Numeric
Description: Sets the gain from the BNC input connector to the telephone interface output. Normally this value is zero. To route signals from the input connector to the telephone interface, set this property value to the desired signal gain. When ringing is started this value is forced to 0.0. When ringing stops the gain setting returns to its prior value. Note, it is possible to change this value once ringing has started. Though once ringing ends, the new setting is overwritten with the previous value.		

Name:	ID:	Type:
TELINT.CURRENT	**52**	Numeric
Description: Sets the off-hook loop current in units of mA. If a value of -1 is written to this property, the AI-7280 changes into a constant voltage mode of operation. In this mode, the loop current is determined solely by the programmed line voltage and the total loop resistance. The total loop resistance is the sum of the AI-7280 internal feed current resistance (400 ohms), resistance of any artificial lines, and the resistance of the connected terminal equipment device. When operating in constant current mode, the valid range for this property is from 5 to 72 mA. If the Extended Line Feed Option (AI-E702) is installed, the maximum loop current setting is increased to 105 mA. When setting this property to a value less than TELINT.HOOKTHRESH, then TELINT.HOOKTHRESH is modified to match the TELINT.CURRENT value.		

Name:	ID:	Type:
TELINT.GENGAIN	**58**	Numeric
Description: Adjusts the level of all tone, FSK, DTMF, and noise generators prior to the signal being transmitted by the telephone interface. By default, this property is set to 1. When ringing is started this setting is forced to a value of 1. Once ringing stops, it is returned to the previous value. It is possible to change this property once ringing has started. Though when ringing stops the setting is returned to the prior value.		

Name:	ID:	Type:
TELINT.HOOKDETECT	**54**	Numeric - r/o
Description: This read only property returns the off-hook detection status. If a connected TE is off-hook a value of 1 is returned. If on-hook zero is returned. The amount of loop current used to determine the on or off hook state is set with the HOOKTHRESH property. When a TE transitions to either the on or off hook state, a de-bounce delay (2 ms) is incurred. However the TIMER.ONHOOK and TIMER.OFFHOOK time stamps are corrected for this delay. A larger de-bounce delay is incurred when ringing is active (20 ms). In addition, when ringing stops this property is not updated to the current line state until a delay of 50 ms has elapsed. This prevents false on or off-hook detections when ringing is stopped due to large loop current transients when high REN loads are used.		

Name:	ID:	Type:
TELINT.HOOKTHRES	**60**	Numeric
Description: Sets the off-hook current trip threshold in units of mA. The valid range from 5 to 25 mA. Note, this setting must be less than the TELINT. CURRENT setting when operating in constant current feed mode.		

Name:	ID:	Type:
TELINT.LINEIMP	**53**	Numeric
Description: Sets the AC line impedance of the telephone interface as follows: 0 = 600 ohms 1 = 900 ohms 2 = TBR-21 3 = optional impedance (only available if installed in the AI-7280)		

Name:	ID:	Type:
TELINT.MEASPOINT	**55**	Numeric
Description: This property determines how signals are sensed from the AI-7280's front panel RJ-11 jack. A value of zero (default) forces the AI-7280 to perform all AC and DC voltage measurements from the inner two conductors of the RJ-11 jack. These pins are connected to the telephone interface of the AI-7280. Writing a non-zero value causes all measurements to be made from the outside pair of conductors on the RJ-11 jack. These two pins are not connected to the telephone interface.		

Name:	ID:	Type:
TELINT.MEASRANGE	**56**	Numeric
Description: Sets the AC signal measurement range. If set to zero (default), the maximum AC signal voltage that can be measured is approximately 5 Vrms. Writing a non-zero value to this property allows AC signals of up to 100 Vrms to be measured.		

Name:	ID:	Type:
TELINT.OSI	**50**	Numeric
Description: Writing a non-zero value to this property removes the DC feeding voltage and AC signal source impedance from the inner two conductors of the front panel RJ-11 jack. The default value of zero connects the DC feeding voltage and AC signal source impedance to the RJ-11 jack.		

Name:	ID:	Type:
TELINT.REVERSE	**49**	Numeric
Description: If set to a non-zero value the polarity of the telephone line interface is reversed.		

Name:	ID:	Type:
TELINT.VOLTAGE	**51**	Numeric
Description: Sets the telephone interface line voltage. The valid range for this property is from 0 to 72 volts. If the Extended Line Feed Option (AI-E702) is installed, the maximum line voltage setting is increased to 105 volts.		

Name:	ID:	Type:
TELINT.VRAMPDEST	**61**	Numeric
Description: This property is used with VRAMPRATE to change the line voltage at a constant slew rate. VRAMPDEST sets the destination line interface voltage. If a negative voltage is specified, the line voltage ramps down to zero, toggles the REVERSE property, and then ramps back up to the destination line voltage.		

Name:	ID:	Type:
TELINT.VRAMPRATE	**62**	Numeric
Description: Sets the voltage slew rate. The units are in Volts/ms and must be in the range of 0 to 20. Writing to this property starts the process of ramping the line voltage towards the value set by VRAMPRATE. When the line voltage reaches the destination voltage this property value is reset to zero. Writing zero at any time stops the change in the line voltage. Note, if high voltage slew rates are specified and a line polarity change is required (negative VRAMPDEST value), then distortion in the line voltage waveform may be incurred due to the delay in toggling the polarity reversing relay.		

3.27 TIMER Properties

The following timer properties are used to record when events occur or schedule actions with user programs. All of the timer values are in units of seconds.

Name:	ID:	Type:
TIMER.FAST	**45**	Numeric
Description: Fast update timer. This timer counts up every 25.6 microseconds to a maximum of 100 seconds. It can be written to with any value within the range of 0 to 100.		

Name:	ID:	Type:
TIMER.INTPERIOD	**48**	Numeric
Description: Interrupt period timer. Writing a value to this timer causes it to count down towards zero. Once zero is reached, it sets an interrupt bit and restarts the timer with the original value written to it. This process continues indefinitely. To stop the timer, write a negative value to it. The interrupt bit can be programmed to start a user program that has been halted by using the HALTUNTIL statement. In addition, status on the interrupt bits can be read by using the SYSTEM.GETFLAGS property. The maximum value that can be written to this timer is 100,000 seconds.		

Name:	ID:	Type:
TIMER.OFFHOOK	**47**	Numeric
Description: This property gets set to the value of the SLOW timer when the telephone interface line status goes off-hook. Note that this property does not get updated until after off-hook the de-bounce period has expired. However its value is time corrected for the de-bounce period.		

Name:	ID:	Type:
TIMER.ONHOOK	**46**	Numeric
Description: This property gets set to the value of the SLOW timer when the telephone interface line status goes on-hook. Note that this property does not get updated until after on-hook the de-bounce period has expired. However its value is time corrected for the de-bounce period.		

Name:	ID:	Type:
TIMER.ROLLAT	**228**	Numeric
Description: Sets a value for TIMER.SLOW that causes it to reset to zero. When TIMER.SLOW resets it causes TIMER.ROLLCOUNT to increment by 1. TIMER.SLOW only resets if TIMER.ROLLAT is a positive value. Writing zero or a negative value prevents any automatic reset of TIMER.SLOW. A positive value to this property bounds TIMER.SLOW to a range between zero and this property less 200 microseconds.		

Name:	ID:	Type:
TIMER.ROLLCOUNT	**229**	Numeric
Description: Used to count the number of times TIMER.SLOW has reached the TIMER.ROLLAT value and been reset to zero. When TIMER.SLOW resets this property value is converted to an integer and incremented by one.		

Name:	ID:	Type:
TIMER.SLOW	**44**	Numeric
Description: Slow update timer. This timer counts up every 200 microseconds to a maximum of 100,000 seconds. It can be written to with any value in the range of 0 to 100,000. The timer value is used to record on-hook and off-hook transition times, as well as recording the start and stop times for any DTMF digits captured (DTMFCAP.STARTTIME, DTMFCAP.STOPTIME). Note that as the timer value increases, the precision by which it can be read decreases. This is due to limitations in the 32 bit floating point format used by the AI-7280. If TIMER.ROLLAT is set to a positive value this timer resets to zero when it reaches the TIMER.ROLLAT value, which also causes TIMER.ROLLCOUNT to incremented by one.		

Name:	ID:	Type:
TIMER.SYSTEM	**43**	Numeric
Description: Secondary slow updating timer. This timer counts up every 200 microseconds to a maximum of 100,000 seconds. It can be written to with any value in the range of 0 to 100,000. Note that as the timer value increases, the precision by which it can be read decreases. This is due to limitations in the 32 bit floating point format used by the AI-7280.		

3.28 TONE Properties

The TONE properties are used to control one or more of the four tone generators (A, B, C, or D) in a simultaneous manner. It supports changing the enable state and phase of any of the four tone generators at exactly the same time.

Name:	ID:	Type:
TONE.ENABLE	**226**	Numeric - w/o
Description: Writing a non-zero value enables all of the tone generators specified by the MASK property. Writing zero disables all of the tone generators specified by the MASK property.		

Name:	ID:	Type:
TONE.MASK	**225**	Numeric
Description: Sets an integer bit mask which determines which tone generators are affected by the ENABLE and PHASE properties. Bit 0 = If set, modify ToneA settings Bit 1 = If set, modify ToneB settings Bit 2 = If set, modify ToneC settings Bit 3 = If set, modify ToneD settings No other bit values are valid and are forced to zero.		

Name:	ID:	Type:
TONE.PHASE	**227**	Numeric - w/o
Description: Sets the phase angle setting of all the tone generators specified by the MASK property. The valid range is from 0 to 360 degrees. Note, the tone generator's phase is modified only if it is currently disabled. If enabled (running), the phase is unaffected.		

3.29 TONEA Properties

This signal generator can output a single tone of arbitrary level and frequency. It is more flexible than tone generators B, C, or D in that it can be either FSK or AM modulated. The MODULATION parameter determines the operating mode. For AM modulation, tone generator B is the modulating signal while the TONEA.FREQ property sets the carrier frequency. The FSK modulator operates with either an internal data source (the DATA properties), or an external data source (digital input A). While the default wave shape is sine, it may be changed to either triangle, square, or a user defined wave shape.

Name:	ID:	Type:
TONEA.AMDEPTH	**107**	Numeric
Description: This property controls the amplitude modulation depth between the limits of 0 to 100 percent. If the MODULATION property is set to AM (2), then the ToneB generator is used as the modulating source, while ToneA is the carrier frequency. The level of ToneB is ignored as the modulation depth is controlled solely by this property.		

Name:	ID:	Type:
TONEA.BITTIMEMARK	**101**	Numeric
Description: When using the FSK modulator mode of operation, this property sets the time duration of the mark bits. The value must be in the range of 0.00025 to 1.0 seconds.		

Name:	ID:	Type:
TONEA.BITTIMESPACE	**100**	Numeric
Description: When using the FSK modulator mode of operation, this property sets the time duration of the space bits. The value must be in the range of 0.00025 to 1.0 seconds.		

Name:	ID:	Type:
TONEA.ENABLE	**95**	Numeric
Description: The tone generator is enabled by writing a non-zero value. Likewise it is turned off by write a value of zero. If using ToneA as an FSK modulator, the FSK properties must be set along with the modulation mode before writing a non-zero value, otherwise the FSK modulator will not start. Also if the FSK modulator is run in single-shot mode (not continuous) the ENABLE property is not set to zero once the FSK ends. It must be manually reset to zero. In addition when the FSK signal ends, the tone level properties are forced to zero.		

Name:	ID:	Type:
TONEA.FREQ	**96**	Numeric
Description: In single tone mode or AM mode of operation, this property controls the tone frequency. In FSK mode it controls the frequency of the space tone. The valid range of values is from 10 Hz to 18,000 Hz.		

Name:	ID:	Type:
TONEA.FREQMARK	**97**	Numeric
Description: When using the FSK modulator mode of operation this property sets the frequency of the mark tone. The value must be in the range of 10.0 Hz to 18,000 Hz.		

Name:	ID:	Type:
TONEA.FSKACTIVE	**108**	Numeric - r/o
Description: The current state of the FSK modulator is returned by this property as either 1 (on) or zero (off). If the FSKCONTINUOUS property is non-zero, FSKACTIVE will return 1 until the FSK modulator is turned off by setting ENABLE to zero. If the FSKHOLDCARRIER is non-zero, FSKACTIVE will return 1 until the last bit has been sent. Following the last bit this property returns 0 even though the mark or space tone is still be generated. The tone must be turned off by setting ENABLE to zero.		

Name:	ID:	Type:
TONEA.FSKBITINDEX	**102**	Numeric
Description: This property is used to set or read the current data bit pointer for the FSK data array. The FSK data to be sent is stored in a buffer managed by the DATA properties. This property acts as the index pointer into that buffer. While the FSK modulator is active, this property increments from its starting value to the maximum number of bits in the buffer. Writing to this property changes the index pointer and effects which data bits are being generated. To start the FSK data at the first bit set this value to zero before writing a non-zero value to the ENABLE property.		

Name:	ID:	Type:
TONEA.FSKCONTINUOUS	**104**	Numeric
Description: Writing a non-zero value to this property before the FSK modulator is enabled causes the FSK modulator to indefinitely repeat the bit pattern stored in the data buffer. Writing zero disables this option. A repeating FSK bit pattern can be turned off by writing zero to the ENABLE property.		

Name:	ID:	Type:
TONEA.FSKHOLDCARRIER	**105**	Numeric
Description: Writing a non-zero value to this property causes the FSK modulator to maintain the frequency and level of the last FSK bit generated indefinitely. The FSK modulator can be turned off by writing zero to the ENABLE property.		

Name:	ID:	Type:
TONEA.FSKNUMBITS	**103**	Numeric - r/o
Description: The total number of FSK data bits stored in the data buffer is returned by this property. Clearing the data buffer or adding bits to it is controlled by the DATA properties.		

Name:	ID:	Type:
TONEA.LEVEL	**98**	Numeric
Description: In single tone or AM mode of operation this property controls the tone generator output level. In FSK mode, it controls the level of the space tone. Its valid range is from 0 to 4.0 Vrms. When the FSK signal ends this value is set to zero.		

Name:	ID:	Type:
TONEA.LEVELMARK	**99**	Numeric
Description: When using the FSK modulator mode of operation this property sets the level of the mark tone. The value must be in the range of 0 to 4.0 Vrms. When the FSK signal ends this value is set to zero.		

Name:	ID:	Type:
TONEA.MODULATION	**106**	Numeric
Description: The modulation mode of the tone generator is set by this property. The supported modes are: 0 = Single Tone 1 = FSK Modulation 2 = Amplitude Modulation (Tone B is the modulating source) 3 = FSK Modulation - External Digital Input A. If in the amplitude modulation mode tone generator B does not produce any output, except as the modulating source for tone A.		

Name:	ID:	Type:
TONEA.PHASE	**109**	Numeric
Description: Reads or sets the tone generator phase angle from 0 to 360 degrees.		

Name:	ID:	Type:
TONEA.PHASEADJ	**250**	Numeric - w/o
Description: Performs an instantaneous phase adjustment to the tone generator's output between the range of 0 to 360 degrees. For example, writing a value of 90 advances the phase of the tone generator by 90 degrees.		

Name:	ID:	Type:
TONEA.WAVESHAPE	**110**	Numeric
Description: Determines which wave shape used. This property can be set to any of the following values: 0 = sine 1 = triangle 2 = square 3 = user defined Note, the LEVEL property is only calibrated for the sine wave shape. For information on how to program the user defined wave shape, please contact technical support.		

3.30 TONEB Properties

This signal generator can output a single tone of arbitrary level and frequency. While the default wave shape is sine, it may be changed to either triangle, square, or a user defined wave shape.

Note that if Tone A is set for AM modulation then this tone generator becomes the modulation source. The depth of the modulation is set by TONEA.AMDEPTH and TONEB.LEVEL has no effect.

Name:	ID:	Type:
TONEB.ENABLE	**80**	Numeric
Description: Writing a non-zero value enables the tone generator. Writing zero turns off the tone generator. Note, if ringing is enabled the tone generator cannot be enabled. In addition the tone generator is forced off when ringing is started.		

Name:	ID:	Type:
TONEB.FREQ	**81**	Numeric
Description: Sets the tone frequency from 10 Hz to 18 kHz.		

Name:	ID:	Type:
TONEB.LEVEL	**82**	Numeric
Description: Sets the tone level from 0 to 4 Vrms.		

Name:	ID:	Type:
TONEB.PHASE	**83**	Numeric
Description: Reads or sets the tone generator phase angle from 0 to 360 degrees.		

Name:	ID:	Type:
TONEB.WAVESHAPE	**84**	Numeric
Description: Determines which wave shape used. This property can be set to any of the following values: 0 = sine; 1 = triangle; 2 = square; 3 = user defined. Note, the LEVEL property is only calibrated for the sine wave shape. For information on how to program the user defined wave shape, please contact technical support.		

3.31 TONEC Properties

This signal generator can output a single tone of arbitrary level and frequency. While the default wave shape is sine, it may be changed to either triangle, square, or a 256 point user defined wave shape.

Note, if this tone is enabled, the MF generator cannot be enabled. Changing any TONEC property (except PHASE) disables the MF generator (if it is active).

Name:	ID:	Type:
TONEC.ENABLE	**85**	Numeric
Description: Writing a non-zero value enables the tone generator. Writing zero turns off the tone generator. Note, if ringing is enabled the tone generator cannot be enabled. In addition the tone generator is forced off when ringing is started.		

Name:	ID:	Type:
TONEC.FREQ	**86**	Numeric
Description: Sets the tone frequency from 10 Hz to 18 kHz.		

Name:	ID:	Type:
TONEC.LEVEL	**87**	Numeric
Description: Sets the tone level from 0 to 4 Vrms.		

Name:	ID:	Type:
TONEC.PHASE	**88**	Numeric
Description: Reads or sets the tone generator phase angle from 0 to 360 degrees.		

Name:	ID:	Type:
TONEC.WAVESHAPE	**89**	Numeric
Description: Determines which wave shape used. This property can be set to any of the following values: 0 = sine 1 = triangle 2 = square 3 = user defined Note, the LEVEL property is only calibrated for the sine wave shape. For information on how to program the user defined wave shape, please contact technical support.		

3.32 TONED Properties

This signal generator can output a single tone of arbitrary level and frequency. While the default wave shape is sine, it may be changed to either triangle, square, or a 256 point user defined wave shape.

Note, if this tone is enabled, the MF generator cannot be enabled. Changing any TONED property (except PHASE) disables the MF generator (if it is active).

Name:	ID:	Type:
TONED.ENABLE	**90**	Numeric
Description: Writing a non-zero value enables the tone generator. Writing zero turns off the tone generator. Note, if ringing is enabled the tone generator cannot be enabled. In addition the tone generator is forced off when ringing is started.		

Name:	ID:	Type:
TONED.FREQ	**91**	Numeric
Description: Sets the tone frequency from 10 Hz to 18 kHz.		

Name:	ID:	Type:
TONED.LEVEL	**92**	Numeric
Description: Sets the tone level from 0 to 4 Vrms.		

Name:	ID:	Type:
TONED.PHASE	**93**	Numeric
Description: Reads or sets the tone generator phase angle from 0 to 360 degrees.		

Name:	ID:	Type:
TONED.WAVESHAPE	**94**	Numeric
Description: Determines which wave shape used. This property can be set to any of the following values: 0 = sine 1 = triangle 2 = square 3 = user defined Note, the LEVEL property is only calibrated for the sine wave shape. For information on how to program the user defined wave shape, please contact technical support.		

3.33 USB Properties

The following properties are used to control the USB port, check its status, send data, and receive data.

Name:	ID:	Type:
USB.ENABLE	**36**	Numeric
Description: Enables or disables the USB port. Writing a 0 disables the port. Writing a 1 enables the USB port. However enabling the port may not allow communications unless the USB cable is plugged in and the PC has enumerated this device and the driver is installed. Use the STATUS property to check if communication with the PC is enabled.		

Name:	ID:	Type:
USB.GETBYTE	**38**	Numeric - r/o
Description: Returns the next byte value (0 to 255) from the receive buffer. This property should only be read if RXCOUNT returns a value greater than zero.		

Name:	ID:	Type:
USB.RXCOUNT	**37**	Numeric - r/o
Description: Returns the number of data bytes waiting in the receive buffer. Unless SYSTEM.HALTCMDS bit 1 is set the command processor routines will automatically remove data bytes from the receive buffer.		

Name:	ID:	Type:
USB.SENDBYTE	**39**	Numeric - w/o
Description: Writing to this property sends a single byte value to the transmit buffer. The TXFREE property should be checked prior to sending a byte as there may not be any free space left in the buffer.		

Name:	ID:	Type:
USB.SENDSTRING	**40**	Numeric - w/o
Description: Similar to the SENDBYTE property, but instead sends a character string to the transmit buffer. The TXFREE property should be checked prior to sending the string as there may not be any free space left in the buffer.		

Name:	ID:	Type:
USB.STATUS	**42**	Numeric - r/o
Description: Returns the status of the USB connection as follows: 0 = USB port not enabled 1 = No host connected or its power is off 2 = A host is connected but the USB port has not been enumerated 3 = The port is enumerated and can be used to send or receive data		

Name:	ID:	Type:
USB.TXFREE	**41**	Numeric - r/o
Description: This read only property returns the amount of free space left in the transmit buffer. It is important not to write any data bytes into the transmit buffer unless this property returns a value greater than zero.		

Note: The properties to receive data bytes should not be accessed unless the SYSTEM.HALTCMDS property has bit 2 set. Otherwise the command processor will automatically remove bytes from the receive data buffer.

4. Programming

4.1 Overview

As mentioned in section 1.2 Firmware Overview, the AI-7280 includes the ability to store and execute user defined programs. These programs may read or write to all of the AI-7280's hardware abstraction layer (HAL) properties giving them a wide range of capabilities.

There are two primary motivations for developing programs. They are:

- Time critical tasks: In some testing situations the timing requirements between telephony events makes it difficult to operate the AI-7280 using the PC. The un-deterministic latency of programs running on a PC along with latency within the serial or USB communication channel causes a large degree of timing uncertainty. Using the program execution units (PEU) within the AI-7280 eliminates the PC as a possible source of timing uncertainty.
- Offloading simple tasks: For more complex test systems it may be simpler to break the system into two levels. Simple tasks can be performed by the AI-7280 PEUs, while the higher level test control tasks performed by the PC.

Capabilities

The AI-7280 contains six program execution units (PEU). Each one can be thought as a very simple processor executing instructions that manipulate HAL properties in order to perform user specified tasks. The operation of each PEU is controlled by commands sent to the AI-7280. A basic summary of a PEUs capability is:

- Read and/or write data to any of the HAL properties.
- Perform basic math operations.
- Flow control structures for implementing loops and conditional statements.
- A simple stack supporting functions and subroutines.
- Ability to fully control any of the other six PEUs, thus supporting multiple programs running in parallel or a single program consisting of multiple processes.
- A private data pool used for storing variable data.
- A global data pool shared by all PEUs. Normally used to pass data between PEUs or between a PEU and the PC.
- Executes from either RAM or the AI-7280's non-volatile flash memory. Programs executing from RAM are downloaded from the PC.

Program Development

The AI-7280 PEUs are very simple virtualized processors. They execute code that consists of only printable ASCII characters. The ASCII strings representing the programs are either downloaded into the AI-7280's RAM or stored in its flash memory. While it is possible to manually create the executing code, an easier approach is to use a compiler that translates higher level statements into the low level code.

The TRsSim software supports the capabilities to compile high level scripting statements into the object code used by the AI-7280. While this software is primarily meant to provide a high level graphical user interface for control, it also includes a development environment for creating programs.

Once a program is compiled into object code by the TRsSim software, the object code can be used in a number of ways including:

- TRsSim can download the object code to the AI-7280's RAM and execute the program.
- TRsSim can store the object code into the AI-7280's non-volatile flash memory. This allows future execution of the program by simply sending an appropriate command.
- The object code can be downloaded into the AI-7280's RAM by using the provided Win32 DLL.
- The object code can be downloaded into the AI-7280's RAM manually by sending the appropriate 'PC', 'PL', and 'PA' commands.

The following section provides an example of creating a program for the AI-7280 by using the TRsSim software.

Note: An older PC software application called A.I.WorkBench can also be used to compile high level scripting statements into object code for the AI-7280. This software was designed to operate with the older and less capable AI-80; however, it also supports the AI-7280 via an RS-232 connection (USB is not supported). The A.I.WorkBench provides the same basic capability as the TRsSim software to compile, download, and execute programs. However it lacks support for some of the advanced features unique to the AI-7280 that do not exist on the older AI-80 product.

4.2 Programming Example

The basic concepts in creating a program for the AI-7280 are explored in this section and the steps needed to create a program using the TRsSim software are described in a tutorial fashion. It is assumed that the version of TRsSim running on the PC is at least 3.0.

What is TRsSim

TRsSim stands for Tip/Ring Signal Simulator. It is a Windows PC software application that provides a high level graphical user interface for a number of products. This includes the AI-7280.

Using TRsSim with the AI-7280 makes it easy to perform complex telephony related tests such as dialing verification, ring generation, call display testing, and fixed line SMS testing. Included with TRsSim is a development environment for creating script programs to automate tasks. While the scripting system runs and executes entirely within TRsSim it can be configured to generate programs for the AI-7280 program execution units (PEU). This is the mode used in the following example.

To start, the TRsSim software requires either a RS-232 serial or USB connection with an AI-7280. To accomplish this please consult the TRsSim user guides included with the software.

Creating a Device Script Project

The scripting environment within TRsSim supports working with multiple projects at the same time. A project is simply one or more source code files that represent a complete program. However TRsSim only supports one project as the active project at a time. It is only the active project that is compiled and executed.

Step 1: From the VIEW menu in TRsSim, select the SCRIPT EDITOR command.

This command opens the scripting editor window from where all projects are created and managed. The window shows one of three different views as selected by the tabs located in the top right corner. They are **Editor**, **Settings**, and **Status**.

The editor view is used to modify script programs and managing the projects. Each project can contain multiple source code modules which may be ordered in a hierarchical fashion. By default, all projects are created with just a single source code module.

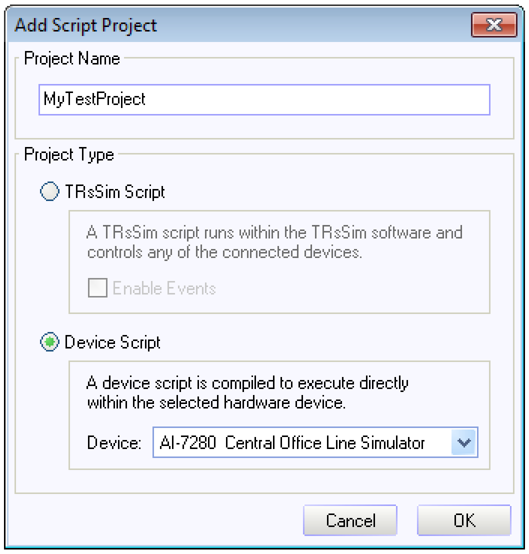
The settings view shows various settings for each project. The specific settings available depend on the type of project created.

Finally, the status view shows debugging information for the currently active script project.

Note that the debugging information is only available for scripting programs that control TRsSim itself. Not for device script programs that execute on the AI-7280.

Step 2: From the PROJECT menu select the ADD PROJECT command.

This command creates a new project that is used in this example. It shows a window similar to the following where project details are specified.


Add Script Project

Project Name

MyTestProject

Project Type

TRsSim Script

A TRsSim script runs within the TRsSim software and controls any of the connected devices.

Enable Events

[✓ Device Script]

A device script is compiled to execute directly within the selected hardware device.

Device: AI-7280 Central Office Line Simulator

For this example, ensure that **Device Script** is selected and that the target device is the AI-7280. Optionally a name for the project may be entered in the text box at the top of the window.

By selecting Device Script the TRsSim scripting environment knows that the program compiled will be executed directly within AI-7280.

Clicking the OK button shows the new project listed in the upper right corner of the editor view in a bold font with green highlighting. The green highlighting indicates that the new project is now the active project. The text editor displays the default source code module called **Start**. From this point the program source code can be created.

All scripting programs are composed into one of three different types of blocks. They are PROCESS blocks, FUNCTION blocks, or SUB blocks. When writing programs for the AI-7280 at least one process block must exist. This is from where program execution starts.

TRsSim automatically creates a single process block called START within the editor. All program statements that execute must be contained within this or other blocks.

Step 3: Add the following scripting statements to the editor window.

Note, text lines that start with ***** **'** or **;** are treated as comments and ignored by the compiler.

```
Process START

*   Generate a tone sweep using a logarithmic step size.

    Const cFreqStart = 100          'start frequency
    Const cFreqStop = 3000          'end frequency
    Const cNumPoints = 20           'number of frequency points
    Const cTimeAtPoint = 200        'in ms
    Const cLevel = 0.5              'output level in Vrms

    With ToneA                      'Using tone generator A for the sweep

        'Make sure the tone is off and set the level and start frequency.
        .Enable = 0
        .Level = cLevel
        .Freq = cFreqStart

        'Calculate by how much the tone frequency needs to increase by
        'for each step.
        Numeric Lstep = cFreqStop / cFreqStart
        Lstep = Log(Lstep) / cNumPoints
        Lstep = Exp(Lstep)

        'Generate the tone sweep.
        .Enable = 1
        Loop cNumPoints
            Me.Wait = cTimeAtPoint
            .Freq = .Freq * Lstep
        End Loop

    End With

End Process
```

The above program uses one of the AI-7280's four tone generators to create a logarithmic frequency sweep from 100 to 3000 Hz.

All of the statements are contained inside the process block called ***START***. This is the only process block in the program and will be the one executed when the program is started.

The first five statements in the block define constants used in the program. These set the frequency sweep range, the number of points, how long to wait at a single frequency point, and the tone level during the sweep.

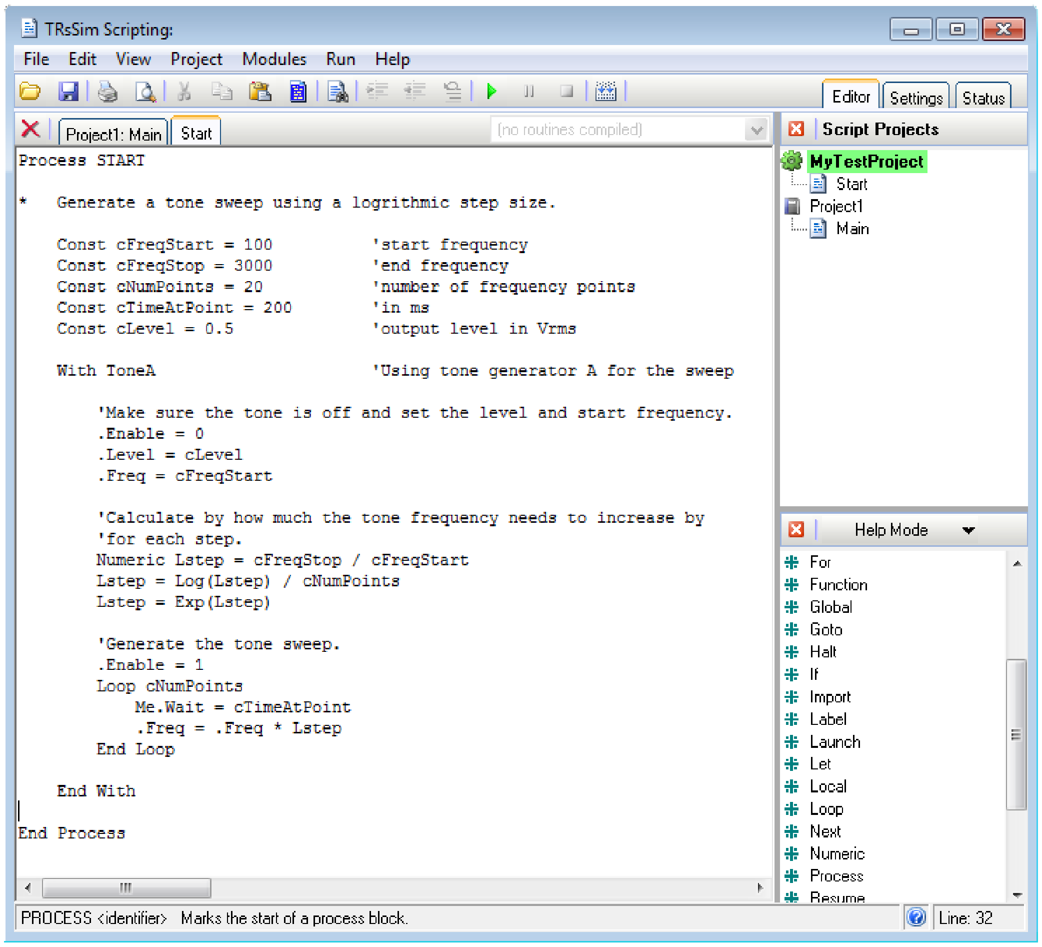
Next is the ***With ToneA*** statement, which tells the compiler that any properties that start with a dot belong to tone generator A. This is really a short cut for reducing the amount of typing needed. Instead of having to type the full name of a property, like ***ToneA.Enable***, using a 'with' statement simplifies this to just ***.Enable***.

The program continues by making sure the tone generator is turned off, sets the tone level, and then sets the initial frequency equal to the ***cFreqStart*** constant.

Next are three statements calculating how much the frequency should increase by in order to perform a logarithmic frequency sweep. A variable holding numeric data called ***Lstep*** is calculated from the start frequency, stop frequency, and number of data points.

Finally, the tone generator is turned on by setting its Enable property to a non-zero value. The program enters a loop repeating two statements which first suspend program execution by the number of milliseconds to wait at each frequency, and then to increase the frequency by the amount calculated in the ***Lstep*** variable. These two statements are executed for as many frequency points specified in the constant cNumPoints.

The editor window should look similar to the following picture.



```
Process START

*   Generate a tone sweep using a logrithmic step size.

    Const cFreqStart = 100          'start frequency
    Const cFreqStop = 3000          'end frequency
    Const cNumPoints = 20           'number of frequency points
    Const cTimeAtPoint = 200        'in ms
    Const cLevel = 0.5              'output level in Vrms

    With ToneA                      'Using tone generator A for the sweep

        'Make sure the tone is off and set the level and start frequency.
        .Enable = 0
        .Level = cLevel
        .Freq = cFreqStart

        'Calculate by how much the tone frequency needs to increase by
        'for each step.
        Numeric Lstep = cFreqStop / cFreqStart
        Lstep = Log(Lstep) / cNumPoints
        Lstep = Exp(Lstep)

        'Generate the tone sweep.
        .Enable = 1
        Loop cNumPoints
            Me.Wait = cTimeAtPoint
            .Freq = .Freq * Lstep
        End Loop

    End With

End Process
```

PROCESS <identifier> Marks the start of a process block.

Step 4: From the RUN menu select the BUILD ALL command.

This command compiles the program. If any errors are encountered, a message is displayed and the text line in error is highlighted.

Once the program compiles without errors it can be executed.

Step 5: From the RUN menu select the START command, or press F5, or click the tool bar button with the green arrow.

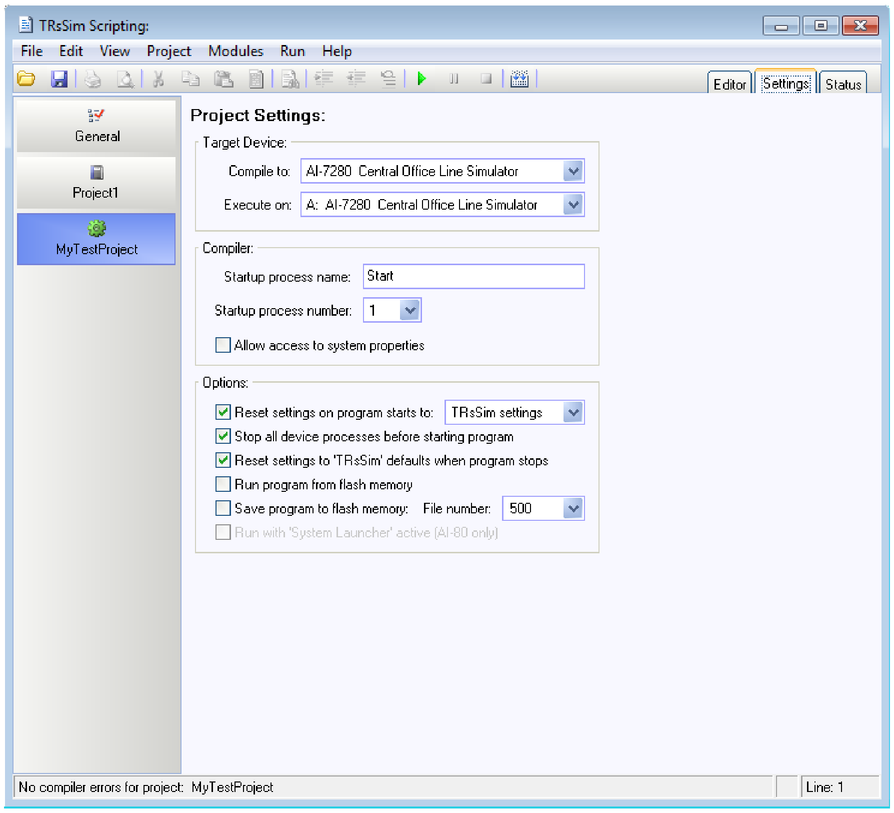
If the program has been compiled without error, TRsSim then downloads the program into the AI-7280's RAM and begins its execution.

While a program runs on the AI-7280 the start button on the tool bar (green arrow) is disabled. However, the pause button (blue bars) and stop button (red box) are enabled. Clicking on either pauses or stops the program execution respectively. The program can be started or stopped as many times as desired.

If any changes are made to the program it is recompiled automatically the next time the program is started.

Step 6: Click the 'Settings' tab to show the project settings.

The settings tab allows for customization of how the TRsSim software manages the script projects. After clicking the tab, a window similar to the following is displayed.



Project Settings:

Target Device:

Compile to: AI-7280 Central Office Line Simulator

Execute on: A: AI-7280 Central Office Line Simulator

Compiler:

Startup process name: Start

Startup process number: 1

☐ Allow access to system properties

Options:

☑ Reset settings on program starts to: TRsSim settings

☑ Stop all device processes before starting program

☑ Reset settings to 'TRsSim' defaults when program stops

☐ Run program from flash memory

☐ Save program to flash memory: File number: 500

☐ Run with 'System Launcher' active (AI-80 only)

No compiler errors for project: MyTestProject

Line: 1

All of the projects are listed on the left side of the window. Clicking on any of them shows the settings specific to that project. For this example project a brief description of each setting follows:

Target Device Settings:

Compile to:	Determines what hardware device the compiler creates object code for. In this example, it is the AI-7280.
Execute on:	Sets which hardware device the object code is executed on. TRsSim can connect with up to four devices, each labeled A through D. In this example the compiled program will execute on device 'A', which is an AI-7280.

Compiler Settings:

Startup process name:	The name of the process block where execution begins. If there is only one process block in the program the name in this field does not matter. However, if there are more than one process blocks then this field determines which one is executed on program start.
Startup process number:	Sets which one of the 6 program execution units (PEUs) within the AI-7280 is used to execute the program.
Access system properties:	If checked, the compiler allows reading or writing to various system related HAL properties. Otherwise the compiler reports an error. Caution must be exercised when using any AI-7280 system related properties as it can interfere with the TRsSim software's control over the AI-7280.

Optional Settings:

Reset settings on start:	If checked TRsSim ensures that before program execution is started all of the AI-7280's HAL properties are set to match either the current TRsSim settings, or the power up default

	settings of the AI-7280.
Stop processes before start.	If checked TRsSim stops the execution of all PEUs on the AI-7280 before starting the execution of the compiled program. This ensures that only the compiled program is executing.
Reset settings on stop:	If checked TRsSim restores all of the AI-7280 HAL properties to match current TRsSim settings when the program stops.
Run from flash memory:	If checked TRsSim executes the program from the flash memory. The program number executed is part of the following setting.
Save to flash memory:	If checked TRsSim writes the program into the flash memory following every successful compile. The program number is specified in the text box. Note: It is important NOT to use program numbers 1 through 10. These are reserved for system programs used when the AI-7280 boots. Using these numbers will overwrite a system program. Note: Program number 900 and 901 are used exclusively for TRsSim. Overwriting this program will cause problems when TRsSim attempts to control aspects of the AI-7280.
Run with system launcher:	This setting is only applicable when writing programs and being connected to the AI-80.

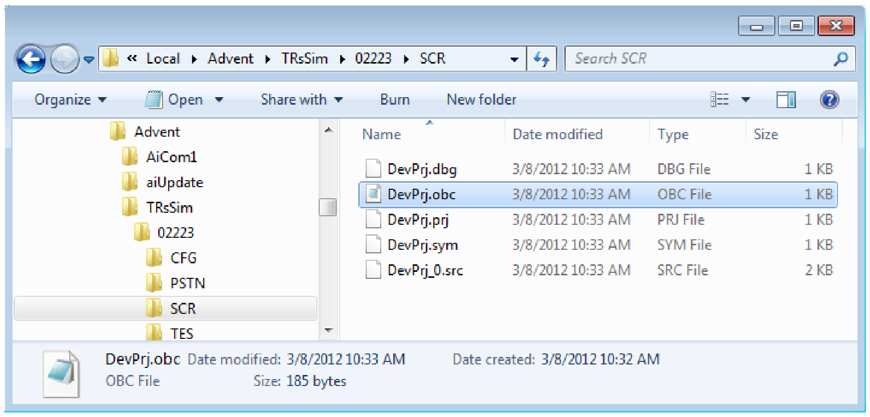
Direct Program Loading

While the TRsSim software can be used to compile an AI-7280 device script, in some situations it may be desirable to manage the loading of the programs and their execution from another application. Applications can load and control AI-7280 programs by either sending commands directly as described in previous sections of this document, or using the available Windows DLL. The DLL manages all of the low level communication and control of the AI-7280 while the APIs provide a higher level interface for applications.

For either method the object code file created by the TRsSim compiler is required. This file contains the low level instructions executed by the AI-7280 PEUs. TRsSim writes the object code to a file at the following path:

```
C:\Users\<user>\AppData\Local\Advent\TRsSim\<loc>\SCR\DevPrj.obj
```

Where <user> is the name of the currently active user account in windows, and <loc> is a five digit number derived from the TRsSim's installation path. Each installation of TRsSim at a different path creates a separate five digit number.

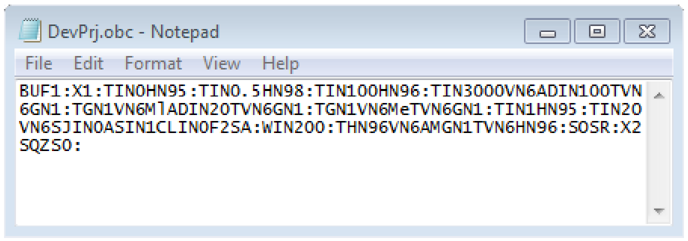


Name	Date modified	Type	Size
DevPrj.dbg	3/8/2012 10:33 AM	DBG File	1 KB
DevPrj.obc	3/8/2012 10:33 AM	OBC File	1 KB
DevPrj.prj	3/8/2012 10:33 AM	PRJ File	1 KB
DevPrj.sym	3/8/2012 10:33 AM	SYM File	1 KB
DevPrj_0.src	3/8/2012 10:33 AM	SRC File	2 KB

DevPrj.obc Date modified: 3/8/2012 10:33 AM Date created: 3/8/2012 10:32 AM
OBC File Size: 185 bytes

Note, the above figure was taken with Windows 7.

The 'DevPrj.obj' file contains only printable ASCII characters. For this example, opening the file in Windows Notepad shows the object code created by the TRsSim compiler.



```
BUF1:X1:TIN0HN95:TIN0.5HN98:TIN100HN96:TIN3000VN6ADIN100TVN
6GN1:TGN1VN6MlADIN20TVN6GN1:TGN1VN6MeTVN6GN1:TIN1HN95:TIN20
VN6SJIN0ASIN1CLIN0F2SA:WIN200:THN96VN6AMGN1TVN6HN96:S0SR:X2
SQZS0:
```

The contents of this file can either be passed to the AI-7280 Windows DLL for download into theAI-7280, or combined with the 'PL' command described in section 2.3 Program Control Commands for manual download.

4.3 Script Language Basics

This section provides an overview of the scripting language used by the TRsSim compiler. For more a complete reference, please consult the Language Reference Guide document included with the TRsSim software.

It is important to note that the compiler generates programs for execution either within the TRsSim software itself or within a device like the AI-7280. The two compiler targets while virtually the same have some differences between them. As a result, not all of the language functions and statements are available for both targets.

Process Blocks

The basic unit of any AI-7280 program is a process block. Programs are constructed from at least one process block, and may execute up to six simultaneously.

Each process runs completely independently from each other. However, if needed, a common data pool is available to be shared between the processes for inter-process communications.

If a program contains only one process block it is executed when the program is started. When a program contains multiple process blocks then only the process marked as the startup process by the compiler is started. The other process blocks must be started from within the program. The startup process of a program is specified as part of the project settings.

The syntax for a process block is as follows:

```
PROCESS <identifier>
        <statements>
        [EXIT PROCESS]
END PROCESS
```

The <identifier> is a unique name that consists of only letters and numbers.

When a program reaches the END PROCESS or EXIT PROCESS command, that process is terminated. Process blocks cannot be nested.

Function & Subroutine Blocks

In addition to processes, functions and subroutines can be defined and called from within a program. Both the subroutines and functions may pass arguments for processing. Subroutines do not return any data, while functions return either a numeric or string data type. Subroutines are initiated by the CALL command while functions may be part of an expression. The syntax for the function and subroutine blocks are as follows.

```
FUNCTION <type> <identifier> [(<type> <identifier> ...[,<type>
                              <identifier>])]
        <statements>
        [EXIT FUNCTION [ WITH <expression> ]
END FUNCTION [ WITH <expression> ]


SUB <identifier> [(<type> <identifier> ...[,<type> <identifier>])]
        <statements>
        [EXIT SUB]
END SUB
```

All user defined functions or subroutines must be declared outside of a PROCESS block, and cannot be nested (defined inside another). The identifier chosen for a routine is global in scope and may not be used as constants or variables anywhere else in the project.

If parameters are passed to the functions and subroutines, a parameter list enclosed in ( ) brackets must be specified. The parameter list indicates the data type for each parameter, along with a unique identifier for the parameter. When calling functions or subroutines, the calling statement's parameter list is compared to the parameter list in the FUNCTION or SUB statement. If the number of parameters is different or the data types do not match an error is reported.

The syntax for calling a subroutine is:

```
CALL  <identifier>  [ ( <param1> [ , <param2> ] ) ]
```

While functions are called inside expressions as:

```
<identifier>  [ ( <param1> [ , <param2> ] ) ]
```

It is important to note some restrictions with functions and parameters.

- When calling a function or subroutine, expressions cannot be used inside a parameter list. Only variables and constants can be passed.
- All parameters are "read only" from inside a function or subroutine. The called routine cannot modify any of the parameter values, as they are treated as constants within the routine.
- Variable storage within routines does not support recursion. If a function or subroutine is called with recursion, its variable contents are identical to that of the calling instance, and changing its value changes it for all of the other instances
- The GOSUB and RETURN statements are not allowed inside a function or subroutine. These statements are only allowed inside a PROCESS block.

To return a value from a function supply an expression following the WITH keyword, which occurs after EXIT FUNCTION or END FUNCTION. The expression is computed and its results returned when the function ends. The WITH keyword is optional and if missing the function will return either zero or an empty string, depending on the function's data type.

An END SUB or END FUNCTION statement must be included to mark the end of the routine. Once the program reaches this point control is passed back to the calling section.

To exit prematurely from a routine, use the EXIT SUB or EXIT FUNCTION statement.

Variables & Constants

The scripting language supports the use of variables for holding and processing data. The type of data stored by the variable and its scope is determined by how and where it is declared. Every variable in a script program is declared by using one of the data type keywords:

```
NUMERIC, or STRING
```

As implied, numeric variables hold a single floating point numeric value while string variables contain ASCII characters. The syntax for declaring a variable is as follows:

```
[<modifiers>] NUMERIC|STRING [(<register>)] <identifier> [ =
                                 <expression> ]
```

- <modifiers> Optionally, one or more of the following keywords may be used to define the scope and usage of the variable:

  ```
  LOCAL, GLOBAL, EXPORT, IMPORT
  ```

- <register> An integer value used to force the compiler to use the specified register number as the storage location for the variable. This field is optional and is not normally used. In most instances the compiler should be allowed to allocated register location for optimal memory usage.

- <identifier> Unique name for the variable. The name can only contain the characters 'A' to 'Z', numbers '0' to '9' and the underscore character '_'. All variable names must start with a letter and are case insensitive.
- <expression> Optionally, declared variables may be assigned a value by the evaluation of the supplied expression provided the declaration occurs inside a subroutine or function block.

The scope of the variable determines which routine or module may access it. There are two classes of variable scope. They are global (specified by the GLOBAL keyword), and local (specified by the LOCAL) keyword. If the scope is not specified it is assumed to be local.

If a variable has global scope it can be accessed from any routine in any module. However a variable with local scope may be accessed only from within the routine or module it is declared in.

The scoping rules are as follows:

- A variable declared with the GLOBAL keyword: This variable can be accessed from any function or subroutine in any module.
- LOCAL variable declared outside of all functions or subroutines: The variable can be accessed from any function or subroutine within the same source code module. As such, this variable can be shared between all routines defined in the same module. The variable is not accessible outside the module it is declared in.
- LOCAL variable defined inside a function or subroutine: Variable can only be accessed from within the function or subroutine. No other routine can access the variable.

Normally all variables are localized to the program execution unit they execute in. As such, each process has its own instance of all declared variables. To share data between processes use the IMPORT and EXPORT modifiers. They force the compiler to allocate the variable into a data pool that is shared and accessible by all of the processes.

Note that currently read/write restrictions are not in effect for shared variables. As such it does not matter if the IMPORT or EXPORT keyword is used. However it is recommend that EXPORT be used when writing to shared variables and IMPORT when reading from shared variables for future compatibility

Program constants are defined by using the CONST keyword. Constants can be of either numeric or string data type and be used anywhere variables are used. The only exception is being assigned a value by the LET statement. The syntax for defining constants is:

```
[LOCAL|GLOBAL]  CONST  <identifier>  =  <literal>
```

The optional scoping keywords LOCAL and GLOBAL determine which modules and routines have access to the constant. A GLOBAL constant is known by every routine within the script project (every module), while a LOCAL constant is more restricted. The scope of local constants depends on where it is declared. If a LOCAL constant is defined within a function or subroutine it is known only within that routine. However, if defined outside a function or subroutine it is known by every routine in the module.

If no scoping keyword is supplied then all constants defined inside a routine are assumed to be LOCAL and all constants defined outside a routine are assumed to be GLOBAL.

Statements

The following table outlines the various program statements that can be used in a script program for the AI-7280.

LET

Usage:

```
    [LET] <variable> = <expression>
```

The LET statement computes the result of an expression and transfers the value to a variable. The expression itself may contain variables, constants, literals, or function calls. The data type of the variable must match the data type of the expression. As such, numeric variables can only be assigned numbers and string variables can only by assigned strings.

Note, using the keyword LET is optional.

LOOP

Usage:

```
    LOOP  [<expression>]
        <statements>
        [EXIT LOOP]
    END LOOP
```

Repeats a series of statements a specified number of times or an infinite number of times. The optional expression following the LOOP keyword determines the number of times the statements are repeated. If an expression is present it must evaluate to a numeric value. If no expression is present the statements loop indefinitely. Optionally, an EXIT LOOP statement immediately exits the loop.

FOR-NEXT

Usage:

```
    FOR <var> = <exp1> TO|DOWNTO <exp2> [STEP <exp3>]
        <statements>
        [EXIT FOR]
    NEXT <var>
```

Where:

<var> is a variable modified by the FOR-NEXT loop. Must be of a numeric data type.

<exp1> is an expression that represents the initial value for <var>. The expression must evaluate to a numeric value.

<exp2> is an expression that represents the value compared to the <var>. If counting up (TO keyword), the loop ends when the value of <exp2> is less than the <var>. Otherwise when counting down (DOWNTO keyword), the loop ends when <exp2> is greater than the <var> value.

If the optional STEP keyword is used, then <var> is incremented by <exp3> if counting up (TO keyword), or decremented by <exp3> if counting down (DOWNTO keyword). If the STEP keyword is not included in the statement, the <var> is either incremented or decremented by the value of 1.

The FOR-NEXT statement is an alternative method for repeating a block of statements instead of using the LOOP command. It steps a variable either higher or lower between a start value and end value.

Optionally, EXIT FOR is used to break out of the loop.

IF-THEN-ELSE-ELSEIF

Usage:

```
    IF  <expression>  THEN
        <statements>
    [ELSEIF <expression> THEN
        <statements>  ]
    [ELSE
        <statements>  ]
    END IF
```

Where:

<expression> must evaluate to a numerical value.

If the expression following the IF keyword evaluates to true then the following program statements are executed. A true expression is one in which the result is a non-zero value. An expression that evaluates to zero is considered false. Optionally, the ELSEIF or ELSE statement are used to define a block of statements that are executed if the prior expression is false. The END IF statement is required to mark the end of the IF statement.

SELECT-CASE

Usage:

```
    SELECT  <exp1>
        CASE  <exp2> | ELSE
              <statements>
        ...
        [CASE  <expn> | ELSE
              <statements> ]
    END SELECT
```

Where:

<exp1> is an expression that evaluates to a number or string.

<exp2> an expression that is compared to <exp1>

The SELECT statement is an alternative to the IF-THEN-ELSE statement for conditional program execution. It allows for an unlimited number of comparisons and execution paths. The result of the expression following the SELECT keyword is compared to the result of the expression following each CASE keyword. If the two expressions represent the same value, the statements following the CASE keyword are executed. When another CASE keyword is encountered or the END SELECT statement, program execution jumps to the first statement following END SELECT. Optionally the keyword ELSE may be used instead of an expression. It always evaluates to true, and any following statements are executed if none of the previous CASE expressions matched.

CALL

Usage:

```
    [CALL] <identifier>[(<param>[,<param>...])]
```

Where:

<identifier> is the name of the subroutine to call.

<param> are values passed to the subroutine. The number of parameters and their data type must match the subroutine definition.

The CALL statement is used to execute a subroutine. The subroutine must be either defined by using the SUB statement, or built into the scripting language.

The keyword CALL is optional. It is valid to call a subroutine by only specifying the subroutine's identifier and parameter list.

Note that parameter values must be variables, constants, or literals. Expressions are not allowed.

LAUNCH, HALT, STOP, RESUME

Usage:

```
    LAUNCH FILE <file ID number> WITH <PEU number>|ME
    LAUNCH PROCESS <identifier> WITH <PEU number>

    STOP  <PEU number> | ME
    HALT  <PEU number> | ME
    RESUME <PEU number>
```

The LAUNCH statement is used to start execution of a process block, either within the same program or within a new file (program). With the STOP, HALT, and RESUME statements, any of the program execution units can be terminated, halted or resumed respectively. The keyword ME refers to the program execution unit that is performing the statement

The resume statement only functions if the specified PEU is currently in the halted state. If running or stopped the resume statement has no effect.

At the end of a program the compiler automatically adds the equivalent statement “STOP ME”, which stops the current process. However, if the initial process started other processes at some time during its execution then it is responsible for terminating them. Otherwise the other processes will continue to execute even after the startup process has ended.

SWITCH

Usage:

```
    SWITCH [IF <expression>] TO <identifier>
```

Where:

<expression> must evaluate to a numeric result

<identifier> is the name of the process block to switch execution to.

The SWITCH statement provides a means to transfer execution of the current process block to the beginning of the same or different process block.

When the optional IF keyword is used then the provided expression must evaluate to a non-zero value in order for execution to transfer to the specified process. If the expression evaluates to zero then execution continues to any following statements.

LABEL

Usage:

```
    LABEL  <identifier>
```

The LABEL statement marks a position within the current PROCESS, SUB, or FUNCTION block. It can be used with the GOTO statement to immediately move the program execution point to the position of the label.

GOTO

Usage:

```
    GOTO  <identifier>
```

The GOTO statement works in conjunction with the LABEL statement. It causes program execution to jump to the first statement following the specified label.

Note, the GOTO statement is not permitted inside a LOOP block.

GOSUB, RETURN

Usage:

```
    GOSUB  <identifier>

    RETURN
```

Simple subroutines (without parameter passing) can be implemented with the GOSUB command. It causes program execution to jump to the specified label. When the RETURN statement is encountered program execution resumes at the next statement following the GOSUB statement.

Note, GOSUB and RETURN statements are only allowed inside PROCESS blocks.

WITH

Usage:

```
    WITH  <property>
        <statements>
    END WITH
```

The WITH statement is a compiler shortcut simplifying access to the hardware abstraction layer (HAL) properties. By specifying a partial property name the complete property is accessed by typing the remainder of the full property name.

ASM

Usage:
```
    ASM  { object code statements }
```

The ASM statement provides a means to insert object code statements directly into the compiler output. Normally this statement should not be used as it can result in illegal program instructions which immediately halt the program. It can be used as a means for inserting debugging statements or accessing low level registers within the PEU.

Built in Routines and Properties

The script compiler includes a number of pre-defined subroutines, functions, and properties that may be accessed by AI-7280 programs. The following table describes their operation.

Functions:

ABS: Absolute value

Usage:
```
    NUMERIC ABS(<NUMERIC>)
```

Returns the absolute value of the passed numeric value.

ASC: Convert to ASCII code

Usage:
```
    NUMERIC ASC(<STRING>,<NUMERIC>)
```

Returns the ASCII character code of a single character contained in the string at the position of the passed numeric parameter. For example, ASC(“Hello”,2) returns the ASCII code for the second character, which is “e” or 101.

CHR: Convert to character

Usage:
```
    STRING CHR(<NUMERIC>)
```

Converts the numeric ASCII value into a string of one character length.

EXP: Raise to the power of 10

Usage:
```
    NUMERIC EXP(<NUMERIC>)
```

Returns the result of the passed numeric value raised to the tenth power.

GETBYTE: Read byte

Usage:
```
    NUMERIC GETBYTE(<NUMERIC>)
```

Return a byte value from either the local PEU's data pool or the data pool shared by all PEUs. The passed numeric value specifies the read location as follows:

Values 0 to 1199: Byte read from local data pool.
Values 10000 to 11199: Byte read from shared data pool.

HALTUNTIL: Halt until interrupt

Usage:
```
    NUMERIC HALTUNTIL(<NUMERIC>)
```

The HALTUNTIL function suspends program execution until one or more interrupt flags are set. All devices have a system wide interrupt flags register in which various bits may be set when certain conditions are detected. In addition, each executing process block in a program has its own private interrupt flags register.

By using the HALTUNTIL function the program's operation may be synchronized to any of the interrupt flag bits.

The function accepts a single numeric parameter. Two bit-wise AND operations are performed between the passed value and the system wide interrupt flags register and again with the process's interrupt flags register. The two results are then bit-wise OR together. If the result is a non-zero value then program execution resumes and the result is returned by the function.

The purpose of each bit in the system wide interrupt flag register is specific to the device model. They are generally set when various signals or events are detected. By using the HALTUNTIL function a script programs can wait for various events rather than using a polling method.

HEX: Convert to hexadecimal character string

Usage:
```
    STRING HEX(<NUMERIC1>, <NUMERIC2>)
```

Returns a string representing the hexadecimal value of the first numeric value. The second numeric value specifies how many characters are returned by the function.

INT: Convert to integer

Usage:
```
    NUMERIC INT(<NUMERIC>)
```

Returns the integer portion of the passed number. This is the largest integer value less than the passed number. For example, Int(3.99) returns 3, Int(-3.01) returns -4.

ISTR: Convert to integer then to character string

Usage:
```
    STRING ISTR(<NUMERIC>)
```

Converts a numeric value to an integer and then to a string. The largest integer value less than the passed number is returned. For example, Istr(3.99) returns "3", Istr(-3.01) returns "-4".

LEN: String length

Usage:
```
    NUMERIC LEN(<STRING>)
```

Returns the number of characters contained in the passed character string.

LOG: Convert to base 10 logarithm

Usage:
```
    NUMERIC LOG(<NUMERIC>)
```

Returns the base 10 logarithm of the numeric value.

MID: Return single character from a string

Usage:
```
    STRING MID(<STRING>, <NUMERIC>)
```

Returns a string with a single character taken from the passed string. The second parameter determines the character position to return. The valid range for the character position is between 1 and the length of the string.

NEG: Negate value

Usage:
```
    NUMERIC NEG(<NUMERIC>)
```

Returns the negative of the passed value.

NOT: Logical inversion

Usage:
```
    NUMERIC NOT(<NUMERIC>)
```

Returns the logical inverse of the passed numeric value. Passing a non-zero value returns zero, while passing zero returns 1.

SIN: Sine

Usage:
```
    NUMERIC SIN(<NUMERIC>)
```

Returns the sine of the passed numeric value. The units of the passed value are in radians.

STR: Convert number to string

Usage:
```
    STRING STR(<NUMERIC>)
```

Converts a numeric value into a string. If a decimal point is required, the period character '.' is used. The function may format the number in scientific notation depending on its value.

VAL: Convert a string to a number

Usage:
```
    NUMERIC VAL(<STRING>)
```

Attempts to convert the passed character string to a numeric value. If a decimal point is used in the string, it must be a period '.', and not a comma ','. If the string cannot be interpreted as a numeric value the value 0 is returned.

Subroutines:

SETBYTE: Write byte

Usage:
```
    CALL SETBYTE(<NUMERIC1>, <NUMERIC2>)
```

Stores a byte value directly to either the local PEU's data pool or the data pool shared by all PEUs. The first parameter is the byte value to store and the second value specifies where in the data pool to store the byte. The valid locations are as follows:

Values 0 to 1199: Byte written to local data pool.

Values 10000 to 11199: Byte written to shared data pool.

WAIT: Suspend program execution

Usage:
```
    CALL WAIT(<EXPRESSION>)
```

Temporarily suspends the process execution. The expression must evaluate to a numeric value which determines how long to suspend the process. The units are in milliseconds (ms). All other active processes continue to execute while the current process is suspended.

Properties:

ME.IDENT: Process identifier

Usage:
```
    read-only NUMERIC
```

Returns numerical value representing of the PEU reading the property. This value ranges from 1 to 6.

ME.FLAGS: Process interrupt flag set

Usage:
```
    write-only NUMERIC
```

Sets the PEU's interrupt flag register to the specified value. All negative values are forced to -1 which sets all bits.

ME.TIMER: Process timer

Usage:
```
    read/write NUMERIC
```

Reads from or writes to the PEU's internal timer. Writing a value greater than zero loads the timer with the value and clears the PEU's interrupt flags. The timer value is decremented by 1 every millisecond. When it reaches zero it is reloaded with the initial value and the PEU's interrupt flag register is set to the value stored in ME.TIMERFLAG. Writing a value less or equal to zero stops the timer and clears the interrupt flag.

ME.TIMERFLAG: Interrupt flags to set on timer reset

Usage:
```
    write-only NUMERIC
```

Specifies the timer flag value. This value is bit-wise OR'ed with the PEU's interrupt flag register every time the timer value counts down to zero and reloads. All negative values are forced to -1 which sets all bit.

ME.WAIT: Suspend process execution

Usage:
```
    write-only NUMERIC
```

Suspends program execution for the specified number of milliseconds. Writing to this property performs the same operation as using the Wait() subroutine.

The timer and timer flag properties internal to the PEU can be used for a number of different timing purposes. There are three common ways to they can be used.

- The first method writes a .ME.TIMERFLAG value of zero. In this case the PEU's interrupt flag register is never set when the ME.TIMER reaches zero. ME.TIMER is then used by polling its value.
- The second method for using the ME.TIMERFLAG property is to set it to a positive value. Then when ME.TIMER reaches zero the PEU's interrupt flag register gets bitwise OR'ed with ME.TIMERFLAG. A program then uses the HALTUNTIL() function to resume execution when ME.TIMER reaches zero or when any system wide interrupt flag is set. The value that HALTUNTIL() returns with depends on the passed mask parameter. Program execution then resumes on either a timer interrupt or any of the system wide interrupts.
- The final method is to write a negative value to ME.TIMERFLAG. A negative value has a special meaning to the HALTUNTIL() function. It causes program execution to resume when ME.TIMER reaches zero regardless of what the mask parameter is. This allows calling HALTUNTIL(0) which would normally never resume execution. However, in this situation it resumes execution only when ME.TIMER reaches zero and not when any other system wide interrupt flags are set.

Example:
```
;Use the timer to generate ringing once every
;four seconds with a duration of 750 ms.
Me.Timer = 4000
Me.TimerFlag = -1
;Manually set the interrupt now so the first ringing
;cycle starts immediately.
Me.Flags = -1
Loop
    ;Wait for the interrupt flags to be set
    Numeric Flag = HaltUntil(0)
    ;Ring for 750 ms
    Ring.Enable = 1
    Me.Wait = 750
    Ring.Enable = 0
End Loop
```

Appendix A: Firmware Revisions

Updates to the AI-7280's firmware are periodically released which may change the operation in one of the following manners:

- Addition of new properties.
- Expanded capability of existing properties.
- Perform fixes to known bugs.

Any time new properties are added or their capabilities expanded all attempts are made to preserve backwards compatibility. This permits the firmware to be upgraded anytime without concern for breaking applications that were developed with older versions of the firmware.

For the most current listing of the AI-7280's firmware's revision history, please see:

http://www.adventinstruments.com/Products/AI-7280/Support/Firmware%20Revisions

Appendix B: Property Listing by Index

All of the Hardware Abstraction Layer (HAL) properties are listed by index number in the following table.

Note that the property table assumes an AI-7280 firmware version of at least 4.14. If an earlier version is used, some of the following properties may not be available.

ID#	Name	Data Type	Access Restriction
1	System.UnitID	String	Read Only
2	System.SoftID	String	Read Only
3	System.HalID	String	Read Only
4	System.FfsID	String	Read Only
5	System.VtpID	String	Read Only
6	System.SubIndex	Numeric	
7	System.SubValue	Numeric	
8	System.SubInteger	Numeric	
9	System.SubString	String	
10	System.SubFunction	Numeric	Write Only
11	System.Reset	Numeric	Write Only
12	System.ErrorSet	Numeric	Write Only
13	System.ErrorGet	Numeric	Read Only
14	System.FlagGet	Numeric	Read Only
15	System.FlagSet	Numeric	Write Only
16	System.FlagClear	Numeric	Write Only
17	System.HaltCmds	Numeric	
18	System.Options	Numeric	Read Only
19	File.IDlow	Numeric	
20	File.IDhigh	Numeric	
21	File.Exist	Numeric	Read Only
22	File.ItemID	Numeric	
23	File.ItemType	Numeric	Read Only
24	File.ItemNumber	Numeric	Read Only
25	File.ItemString	String	Read Only
26	Comm.Init	Numeric	Write Only
27	Comm.Baud	Numeric	Write Only
28	Comm.RxCount	Numeric	Read Only
29	Comm.GetByte	Numeric	Read Only
30	Comm.SendByte	Numeric	Write Only
31	Comm.SendString	String	Write Only
32	Comm.RxStatus	Numeric	Read Only
33	Comm.TxFree	Numeric	Read Only
34	Comm.CTS	Numeric	Write Only
35	Comm.RTS	Numeric	Read Only
36	USB.Enable	Numeric	
37	USB.RxCount	Numeric	Read Only
38	USB.GetByte	Numeric	Read Only
39	USB.SendByte	Numeric	Write Only
40	USB.SendString	String	Write Only
41	USB.TxFree	Numeric	Read Only
42	USB.Status	Numeric	Read Only

43	Timer.System	Numeric	
44	Timer.Slow	Numeric	
45	Timer.Fast	Numeric	
46	Timer.OnHook	Numeric	
47	Timer.OffHook	Numeric	
48	Timer.IntPeriod	Numeric	

(see #228 to #229 for additional Timer properties)

49	TelInt.Reverse	Numeric	
50	TelInt.OSI	Numeric	
51	TelInt.Voltage	Numeric	
52	TelInt.Current	Numeric	
53	TelInt.LineImp	Numeric	
54	TelInt.HookDetect	Numeric	Read Only,
55	TelInt.MeasPoint	Numeric	
56	TelInt.MeasRange	Numeric	
57	TelInt.Balance	Numeric	
58	TelInt.GenGain	Numeric	
59	TelInt.BncInGain	Numeric	
60	TelInt.HookThreshold	Numeric	
61	TelInt.VRampDest	Numeric	
62	TelInt.VRampRate	Numeric	

63	Source.Meter	Numeric	
64	Source.Analyzer	Numeric	
65	Source.BNCoutput	Numeric	

(see #208 for additional Source property)

66	Measure.Smoothing	Numeric	
67	Measure.Level	Numeric	Read Only
68	Measure.Freq	Numeric	Read Only
69	Measure.NotchLevel	Numeric	Read Only
70	Measure.DcSmoothing	Numeric	
71	Measure.LineVoltage	Numeric	Read Only
72	Measure.LoopCurrent	Numeric	Read Only
73	Measure.Unbalance	Numeric	Read Only

(see #209 to #211 for additional Measure properties)

74	Filter.Type	Numeric	
75	Filter.HiFreq	Numeric	
76	Filter.LoFreq	Numeric	
77	Filter.NumNotch	Numeric	
78	Filter.N1Freq	Numeric	
79	Filter.N2Freq	Numeric	

80	ToneB.Enable	Numeric	
81	ToneB.Freq	Numeric	
82	ToneB.Level	Numeric	
83	ToneB.Phase	Numeric	
84	ToneB.Waveshape	Numeric	

85	ToneC.Enable	Numeric	
86	ToneC.Freq	Numeric	
87	ToneC.Level	Numeric	
88	ToneC.Phase	Numeric	
89	ToneC.Waveshape	Numeric	

90	ToneD.Enable	Numeric	
91	ToneD.Freq	Numeric	
92	ToneD.Level	Numeric	
93	ToneD.Phase	Numeric	
94	ToneD.Waveshape	Numeric	

95	ToneA.Enable	Numeric	
96	ToneA.Freq	Numeric	
97	ToneA.FreqMark	Numeric	
98	ToneA.Level	Numeric	
99	ToneA.LevelMark	Numeric	

100	ToneA.BitTimeSpace	Numeric	
101	ToneA.BitTimeMark	Numeric	
102	ToneA.FskBitIndex	Numeric	
103	ToneA.FskNumBits	Numeric	Read Only
104	ToneA.FskContinuous	Numeric	
105	ToneA.FskHoldCarrier	Numeric	
106	ToneA.Modulation	Numeric	
107	ToneA.AmDepth	Numeric	
108	ToneA.FskActive	Numeric	Read Only
109	ToneA.Phase	Numeric	
110	ToneA.Waveshape	Numeric	

(see #250 for additional ToneA property)

111	Ring.Enable	Numeric	
112	Ring.Freq	Numeric	
113	Ring.Level	Numeric	
114	Ring.Phase	Numeric	
115	Ring.Waveshape	Numeric	
116	Ring.DcLevel	Numeric	

(see #207 for additional Ring property)

117	Noise.Enable	Numeric	
118	Noise.Level	Numeric	

119	Data.Clear	Numeric	Write Only
120	Data.Parity	Numeric	
121	Data.StopBits	Numeric	
122	Data.AddMark	Numeric	Write Only
123	Data.AddSpace	Numeric	Write Only
124	Data.AddAlternate	Numeric	Write Only
125	Data.AddByte	Numeric	Write Only
126	Data.AddChar	Numeric	Write Only
127	Data.AddString	String	Write Only
128	Data.AddXsum	Numeric	Write Only
129	Data.XsumEnable	Numeric	
130	Data.XsumType	Numeric	
131	Data.XsumValue	Numeric	
132	Data.AddHexString	String	Write Only

(see #222, #237 to #242 for additional Data properties)

133	MFGen.Index	Numeric	
134	MFGen.Value	Numeric	
135	MFGen.Level	Numeric	
136	MFGen.FreqAdjust	Numeric	
137	MFGen.OnTime	Numeric	
138	MFGen.OffTime	Numeric	
139	MFGen.Symbol	Numeric	Write Only
140	MFGen.String	String	
141	MFGen.Active	Numeric	

(see #230 to #236 for additional MFGen properties)

142	Dio.OutA	Numeric	
143	Dio.OutB	Numeric	
144	Dio.OutC	Numeric	
145	Dio.InA	Numeric	Read Only
146	Dio.InB	Numeric	Read Only

147	DTMF.Enable	Numeric	
148	DTMF.Digit	Numeric	Read Only
149	DTMF.FreqTol	Numeric	
150	DTMF.FreqTime	Numeric	
151	DTMF.MinLevel	Numeric	
152	DTMF.LowFreq	Numeric	Read Only
153	DTMF.LowLevel	Numeric	Read Only
154	DTMF.HighFreq	Numeric	Read Only
155	DTMF.HighLevel	Numeric	Read Only

156	Echo.TapIndex	Numeric	
157	Echo.TapDelay	Numeric	

158	Echo.TapGain	Numeric	

(see #205, #206 for additional Echo properties)

159	FSK.Active	Numeric	
160	FSK.LevelThreshold	Numeric	
161	FSK.MarkTime	Numeric	
162	FSK.Count	Numeric	
163	FSK.Index	Numeric	
164	FSK.ByteValue	Numeric	Read Only
165	FSK.ByteStatus	Numeric	Read Only
166	FSK.LastByte	Numeric	Read Only

167	DcCap.Index	Numeric	
168	DcCap.Count	Numeric	
169	DcCap.ReadIndex	Numeric	
170	DcCap.Voltage	Numeric	Read Only
171	DcCap.Current	Numeric	Read Only
172	DcCap.HexString	String	Read Only

173	DTMFCap.NumDigits	Numeric	Read Only
174	DTMFCap.Delete	Numeric	Write Only
175	DTMFCap.Index	Numeric	
176	DTMFCap.Digit	Numeric	Read Only
177	DTMFCap.LowLevel	Numeric	Read Only
178	DTMFCap.HighLevel	Numeric	Read Only
179	DTMFCap.LowFreq	Numeric	Read Only
180	DTMFCap.HighFreq	Numeric	Read Only
181	DTMFCap.StartTime	Numeric	Read Only
182	DTMFCap.StopTime	Numeric	Read Only

183	AcCap.Mode	Numeric	
184	AcCap.Index	Numeric	
185	AcCap.Count	Numeric	
186	AcCap.SampleIndex	Numeric	
187	AcCap.Sample	Numeric	
188	AcCap.PlayIndex	Numeric	
189	AcCap.PlayCount	Numeric	

(see #212 to #215 for additional AcCap properties)

190	Bulk.Source	Numeric	
191	Bulk.Dest	Numeric	
192	Bulk.Length	Numeric	
193	Bulk.Space	Numeric	
194	Bulk.AutoHalt	Numeric	

195	SignalIO.BncOutGain	Numeric	

196	FskDropout.Clear	Numeric	Write Only
197	FskDropout.Index	Numeric	
198	FskDropout.BitIndex	Numeric	
199	FskDropout.Gain	Numeric	

200	MeterPulse.Count	Numeric	
201	MeterPulse.Freq	Numeric	
202	MeterPulse.Level	Numeric	
203	MeterPulse.Duration	Numeric	
204	MeterPulse.Repeat	Numeric	

205	Echo.Enable	Numeric	
206	Echo.RingDisable	Numeric	

(see #156, #157, #158 for additional Echo properties)

207	Ring.Trip	Numeric	

(see #111 to #116 for additional Ring properties)

208	Source.PhaseRef	Numeric	

(see #63 to #65 for additional Source properties)

209	Measure.PhaseLevel	Numeric	Read Only
210	Measure.Phase	Numeric	Read Only
211	Measure.PhaseDelay	Numeric	
(see #66 to #73 for additional Measure properties)			

212	AcCap.PlayLoopStart	Numeric	
213	AcCap.PlayLoopEnd	Numeric	
214	AcCap.PlayGain	Numeric	
215	AcCap.RecLoopEnd	Numeric	
(see #183 to #189 for additional AcCap properties)			

216	DcProfile.Index	Numeric	
217	DcProfile.Voltage	Numeric	
218	DcProfile.Rate	Numeric	
219	DcProfile.Count	Numeric	
220	DcProfile.LoopStart	Numeric	
221	DcProfile.LoopEnd	Numeric	
(see #243 for additional DcProfile property)			

222	Data.Duplicate	Numeric	
(see #119 to #132, #237 to #242 for additional Data properties)			

223	Status.A	Numeric	Read Only
224	Status.B	Numeric	Read Only

225	Tone.Mask	Numeric	
226	Tone.Enable	Numeric	Write Only
227	Tone.Phase	Numeric	Write Only

228	Timer.RollAt	Numeric	
229	Timer.RollCount	Numeric	
(see #43 to #48 for additional Timer properties)			

230	MFGen.Level1	Numeric	Write Only
231	MFGen.Level2	Numeric	Write Only
232	MFGen.FreqAdjust1	Numeric	Write Only
233	MFGen.FreqAdjust2	Numeric	Write Only
234	MFGen.FreqOffset1	Numeric	Write Only
235	MFGen.FreqOffset2	Numeric	Write Only
236	MFGen.Reset	Numeric	Write Only
(see #133 to #141 for additional MFGen properties)			

237	Data.PatternLength	Numeric	
238	Data.AddPattern	Numeric	Write Only
239	Data.StopBitValue	Numeric	
240	Data.BitCount	Numeric	Read Only
241	Data.BitIndex	Numeric	
242	Data.BitValue	Numeric	
(see #111 to #116, #222 for additional Data properties)			

243	DcProfile.Mode	Numeric	
(see #216 to #221 for additional DcProfile properties)			

244	Scheduler.Reset	Numeric	Write Only
245	Scheduler.Action	Numeric	
246	Scheduler.AtCount	Numeric	
247	Scheduler.AtTime	Numeric	
248	Scheduler.Parameter	Numeric	
249	Scheduler.Count	Numeric	Read Only

250	ToneA.PhaseAdj	Numeric	Write Only
(see #95 to #110 for additional ToneA properties)			

Appendix C: Error Codes

If commands sent to the AI-7280 cannot be interpreted correctly, an error response is returned. The format of the error response is:

ERR=<x>

Where <x> represents the error code as an integer value. The following table lists all of the possible command error codes.

Code	Description
100	Unknown command. The command character(s) specified do not match any supported commands.
101	The set '?' command is missing the '=' character which is used to separate the register from its new contents.
102	The set '?' command is missing a value. There is no valid data following the '=' character.
110	Invalid sector number was specified with a non-volatile flash memory command.
111	Invalid address was specified with a non-volatile flash memory command.
112	Invalid byte count was specified with a non-volatile flash memory command.
113	Missing the '|' character with a non-volatile flash memory command.
114	The non-volatile flash memory write command does not contain valid data.
115	A write operation failed with the non-volatile flash memory command. The physical memory device may be bad or the sector had not been erased.
116	An erase operation failed with the non-volatile flash memory command. The physical memory device may be bad.
120	Invalid program execution unit (PEU) number specified. Valid numbers range from 1 to 6.
121	Invalid program execution unit (PEU) start command format.
122	The specified program number stored in the non-volatile flash memory does not exist. The program cannot be executed.
501	The register class character is invalid. The first character of a register must be one of the following class types: 'H', 'G', or 'V'
502	The register data type character is invalid. The second character of a register must be one of the following: 'N', 'S'
503	Invalid internal reference encoding. This error should not occur and represents an internal fault with the firmware.
504	Invalid variable register ('G' class) register specified.
505	Invalid program execution unit (PEU) ('V' class) register specified.

The following error codes are returned if a hardware abstraction layer (HAL) register is improperly accessed. The integer value returned is always 6 digits long, where the first two digits represents the type of access error and the following 4 digits represents the register number being accessed.

Code	Description
10xxxx	Attempting to set (>) the value of a HAL register which does not exist.
12xxxx	Attempting to set (>) the value of a HAL register which is read-only.
13xxxx	Attempting to set (>) the value of a HAL register with an incompatible data type. Mismatch between numeric and character string data types.
15xxxx	Attempting to get (?) the value of a HAL register which does not exist.
17xxxx	Attempting to get (?) the value of a HAL register which is write-only.
18xxxx	Attempting to get (?) the value of a HAL register with an incompatible data type. Mismatch between numeric and character string data types.

Appendix D: Support

For assistance regarding any of the topics discussed in this document, or any general questions, please contact us in one of the following methods.

- Email: Technical Questions:
  techsupport@adventinstruments.com

  Sales Inquiries:
  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

Glossary of Terms

ACK Tone

Acknowledgement signal sent by the TE in response to detecting a DT-AS or CAS tone. The signal is usually a DTMF digit 'D' or 'A'.

Bell 202

An FSK modulation standard that uses 1200 Hz for the mark tone frequency and 2200 Hz for the space tone frequency. The baud rate used is 1200 bits per second.

CAS Tone

CPE Alerting Signal. See DT-AS.

Channel Seizure

A FSK modulated alternating zero/one bit pattern that may be sent prior to any data bytes.

Checksum

For the purposes of Caller ID or SMS DLL, the checksum is a byte value sent after the message data. It represents the two's complement sum of all the message byte values. Used for error detection.

dBm

Unit of signal power level. Calculated as 10 times the base 10 logarithm of the ratio of the signals power relative to 0.001 Watt. For example, 1 mW = 0 dBm, 100 mW = 20 dBm. For most telephone applications, signal levels expressed in dBm are expressed with a 600 ohm terminating impedance. This fixes the relationship between dBm and dBV as 0 dBm = -2.218 dBV, or 0 dBV = 2.218 dBm.

dBV

Unit of signal voltage level. Calculated as 20 times the base 10 logarithm of the signal's voltage. For example, 1 Vrms = 0 dBV, 0.1 Vrms = -20 dBV.

DLL

Data Link Layer: A layer in the SMS protocol stack above the physical layer, but below the TL layer. Used to transport TL messages between SME's.

DT-AS

Dual Tone Alerting Signal: A signal composed of two tones (2130 Hz & 2750 Hz) used to alert the TE of an incoming Caller ID data transmission. Also referred to as a CAS tone.

DTMF

Dual Tone Multi-Frequency: A signal comprised of two tones used for dialing or other signaling purposes. Eight different frequencies represented by a 4x4 matrix are used for the two tones.

Flash

Telephone Line Flash: An action by a TE in which it momentarily goes on-hook, then returns to the off-hook state. The on-hook duration is usually in the range of 100 ms to 1000 ms.

FSK

Frequency Shift Keying: A method of signal modulation for data transmission. For Caller ID and SMS applications, a tone's frequency is shifted between two values representing either a mark or space bit.

TE

Terminal Equipment: A device connected to the telephone network. Also commonly referred to as a CPE (Customer Premise Equipment).

MDMF

Multiple Data Message Format: A structure used to convey FSK modulated data of up to 255 bytes in length. Most commonly used for Caller ID applications or for sending SMS DLL messages.

MF

Multi-Frequency: A method used to convey signaling information over telephone networks. Uses signals comprised of two, or possible more, tones from a standardized list of frequencies.

Off-Hook

A state in which a TE draws loop current from the tip/ring interface. Denotes that the device is in use or active.

On-Hook

A state in which a TE does not draw any loop current from the tip/ring interface. Denotes that the device is idle or in-active.

OSI

Open Switching Interval: An interval of time in which the central office disconnects the DC feed voltage from the TE. Can be used as a method of informing a TE of an incoming Caller ID message, or simply generated as an artifact by the central office when various resources are being connected/disconnected from the telephone line.

PDU

Protocol Data Unit: For the purposes the ETSI Protocol 1 SMS, the PDU is a data structure used to convey information. These PDU's are the Transfer Layer (TL) messages and take the form of one of six different types. They are: SMS-DELIVER, SMS-DELIVER-REPORT, SMS-SUBMIT, SMS-SUBMIT-REPORT, SMS-STATUS-REPORT, and SMS-COMMAND.

pps

Pulses per Second: A measurement of pulse dialing which expresses the rate of dialing pulses. A dialing pulse is a short interval of time in which the TE goes on-hook and then returns to the off-hook state. The most common pulse rate in use is 10 pps; however various PSTN's may require faster or slower rates for dialing.

PSTN

Public Switched Telephone Network: A network of devices allowing a TE to establish a connection to another device for the purposes of passing voice band signals.

RP-AS

Ring Pulse Alerting Signal: A short ringing burst that is used to alert the TE of an incoming Caller ID data transmission.

SAS

Subscriber Alerting Signal: A signal used as part of the call waiting service alerting the subscriber of an incoming call. Also may be used during a Type II Caller ID transmission by preceding the CAS signal.

SDMF

Single Data Message Format: A data structure that may be used to send Type I Caller ID data consisting of date/time and calling number information. See also MDMF.

SM

Short Message: Information sent between SME's using the Short Message Service (SMS)

SME

Short Message Entity: A device having the capability to send or receive short messages (SM). Can be either a SM-TE or SM-SC.

SMS

Short Message Service: A service used to send short messages (SM) to and from SME's.

SM-SC

Short Message Service Center: A functional unit that receives or sends short messages to a SM-TE.

SM-TE

Short Message Terminal Equipment: A terminal equipment device that is capabile of sending or receiving short messages to or from a SM-SC.

Start Bit

For Caller ID and SMS applications, every 8 bit data byte transmitted via a FSK modulated signal is preceeded with a single start bit. The start bit is always represented with the space tone.

Stop Bit

For Caller ID and SMS applications, every 8 bit data byte transmitted via a FSK modulated signal is followed with at least one stop bit. The stop bit is always represented with the mark tone.

THD+N

Total Harmonic Distortion plus Noise: A measurement normally expressed as a ratio of the signal's harmonic distortion level plus noise to the total signal level.

TL

Transfer Layer: A layer in the SMS protocol stack providing a service to the SMS application layer. The transfer layer messages convey short message data or status information between SME's.

TPDU

Transfer Protocol Data Unit. See ETSI TS 100 901 section 9 for PDU structure as it relates to SMS applications.

Twist

Commonly refers to the ratio in signal level between the row and column tones of DTMF, or the mark and space tones of FSK. The ratio is normally expressed in decibels (dB). In the case of DTMF, positive twist indicates that the column signal level is greater than the row signal level. For FSK the most common convention is that positive twist indicates that the mark signal level is higher than the space level.

Type I

Caller ID data transmission occurring while the TE is in the on-hook state.

Type II

Caller ID data transmission occurring while the TE is in the off-hook state.

V.23

An FSK modulation standard that uses 1300 Hz for the mark tone frequency and 2100 Hz for the space tone frequency. The baud rate used is 1200 bits per second.