ADVENT INSTRUMENTS INC.

AI-80 CALLER ID SIGNAL GENERATOR

User Guide & Reference Manual

Release 3
July 2002

Overview

Table of Contents

Section 1 Installation & Setup

Introduction

The AI-80 is a high performance Caller ID Signal Generator designed primarily for production environments. Built around a flexible signal processing engine, the AI-80 supports all Caller ID signaling protocols and data transmission formats. This includes both the Bell 202 and V.23 FSK data transmission standards, as well as the various DTMF based standards. The AI-80 can generate the Caller ID signals specified by the various standards in use today, such as:

- Bellcore (Telecordia)
- TIA (Telecommunications Industry Association)
- ETSI (European Telecommunications Standards Institute)
- BT (British Telecom)
- CCA (Cable Communication Association)

In addition to providing Caller ID testing capabilities, the AI-80 can be programmed to perform other standard telephone tests, such as:

- Pulse dialing
- DTMF dialing
- Flash timing
- Network tone detection

Designed to be rugged and compact for use in manufacturing environments, the AI-80 can operate by itself for simple testing applications without any other equipment. An optional I/O module can expand the AI-80's capabilities by providing DC measurement capabilities, audio I/O ports, and digital I/O ports for creating small self contained ATE (Automated Testing Environment) systems. For more sophisticated testing applications, the AI-80 can be interfaced to a host computer via a common RS-232 serial port.

The AI-80 comes standard with a large selection of programs for generating Caller ID signals in accordance to various standards. In situations requiring specialized and custom test sequences, the standard programs can be modified to suit the user's need, or new applications can be developed. The A.I. WorkBench software, included with the AI-80, provides a development environment in which existing AI-80 programs, or new programs may be modified or created. Programming the AI-80 is accomplished via a high level language for ease of use. Once compiled, programs can be downloaded and stored into the non-volatile flash memory of the AI-80. The flexible nature of the AI-80 software system allows for easy field upgrades along with a wide range of capabilities. As program enhancements become available, the AI-80 can be updated by simply connecting a PC and executing the accompanying software.

Package Items

Use care when unpacking the AI-80 to avoid damage to the packing materials and the instrument. The packing materials should be retained in case the instrument is to be transported in the future.

The package should contain the following items:

- AI-80 Caller ID Signal Generator
- A.I.WorkBench Software (CD-ROM)
- Cid Lite Software (CD-ROM)
- User Guide & Reference Manual
- 9 Pin Serial Cable (for connection to host PC)
- Telephone Cord with RJ-11 connectors
- AC Power Cord (dependent on destination country)

Safety Precautions

Please observe the following precautions to avoid injury and prevent damage to this product, or any products connected to it.

- Insure the AC power source is compatible with the voltage and frequency requirements stated on the rear label of the instrument.
- Do not operate the instrument ungrounded or with the ground pin of the power cord removed.
- Do not operate the instrument with the case opened.
- Do not allow any liquid to enter the interior of the instruments.
- Observe all terminal ratings to avoid fire or shock hazards. Refer to the product manual for making connections to the instrument.

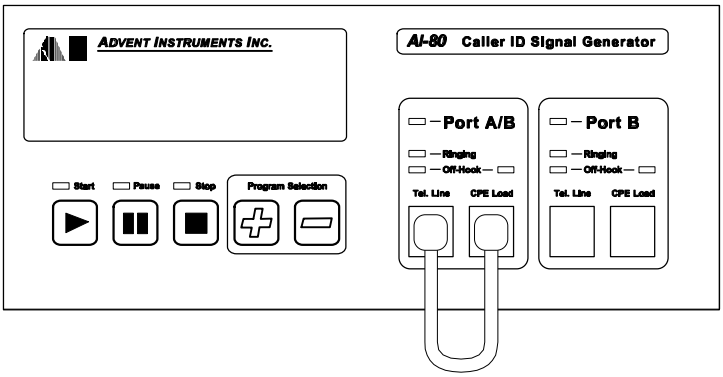
Functional Check of the AI-80

To perform a quick check that the instrument is performing properly, follow these steps:

1) Ensure the unit is turned off via the power switch on the rear panel.

2) Connect the unit to the AC power source that meets the voltage and frequency limits specified on the rear label of the unit.

3) Connect the supplied RJ-11 telephone cord to the Tel. Line jack and CPE Load jack for Port A/B as shown below.



4) Turn on the unit while continuously holding the Pause key down. Keep holding the Pause key until the second beep is heard. This will occur approximately 3 seconds after the power is turned on.

5) Release the Pause key. This will start a short functional test of the instrument.

6) The test will take up to 10 seconds after which the display will either show “PASS” or “F xx”, where xx is a fault code. If a fault code is displayed the unit is operating outside its specified limits and should be returned for repair or calibration. Please contact technical support for repair or calibration procedures.

7) Press the Stop key to reset the unit.

Installing the Software

Verifying the System Requirements

The A.I. WorkBench software requires the following minimum system setup for the target computer.

- Intel 486DX PC computer (or equivalent and compatible processor) (Pentium class or equivalent strongly recommended)
- VGA type monitor
- Microsoft Windows 95, 98, ME, NT4, or 2000 operating system
- Sixteen Megabytes of RAM
- One standard serial port (9 pin or 25 pin)

A.I. WorkBench Software Installation

To install the software package, turn on the computer and launch either the Windows 95, 98, ME, NT4, 2000 operating system. Insert the supplied CD-ROM into the computer. The setup program should automatically run after a few seconds. If not, click the mouse on the START button on the Task bar, followed by the RUN selection. Then type "X:\setup", where X is the driver letter of the CD-ROM, and press ENTER.

The setup program can be used to install the software for any of our products. By following the instructions provided, the setup program can automatically scan the computer for a connected AI-80. If the AI-80 is not connected or turned on, the check box beside the "AI-80" in the list of products must be manually checked in order to install the software.

Following the confirmation of the software installation, the setup program will install each of the selected applications. The setup program will extract all the necessary files and install them into the directory you select. The default directory is set to "C:\Program Files\Advent\AiWb"; however, alternate directories may be chosen by following the instructions in the setup program.

The setup program will automatically create a program short-cut for the A.I. WorkBench software inside the "Advent Instruments" folder.

If any errors messages are displayed during the installation procedure, please contact Technical Support (see Appendix F) for assistance.

Cid Lite Software Installation

Though not required to use the AI-80, the Cid Lite software works in conjunction with the AI-80 Caller ID Signal Generator to provide a simple means for generating the caller ID signals specified by various standards and specifications in use today. This includes both the Bell 202 and V.23 FSK data transmission standards, as well as the DTMF based standards. Operating under Microsoft's Windows 95 or 98 (NT & Windows 2000 version also available), the CidLite

program uses a serial RS-232 communications port to control an AI-80 Caller ID Signal Generator.

Normally the Cid Lite software is installed automatically following the installation of the A.I.WorkBench software. The default directory is set to "C:\Program Files\CidLite"; however, alternate directories may be chosen by following the instructions in the setup program.

The setup program will automatically create a program short-cut for the Cid Lite software inside the "Advent Instruments" folder.

Connecting the AI-80 to the Host PC

Even though the AI-80 will operate without any connection to a PC, in order to create or customize programs, a serial connection is required.

The AI-80 Host Serial port on the rear panel is configured as a standard 9 pin female DCE (Data Communications Equipment) interface. The connection from the PC can be either a 9 pin or 25 pin male DTE (Data Terminal Equipment) interface. If a 9 pin port is available on the PC, connect the supplied 9 pin cable between the AI-80 and the host PC. If only a 25 pin port is available, attach a 25 pin to 9 pin adapter (not supplied) to the PC, before connecting the supplied 9 pin cable between the AI-80 and PC.

The serial port used on the PC must be recognized and configured by the Windows operating system as either COM 1, COM 2, COM 3, or COM 4.

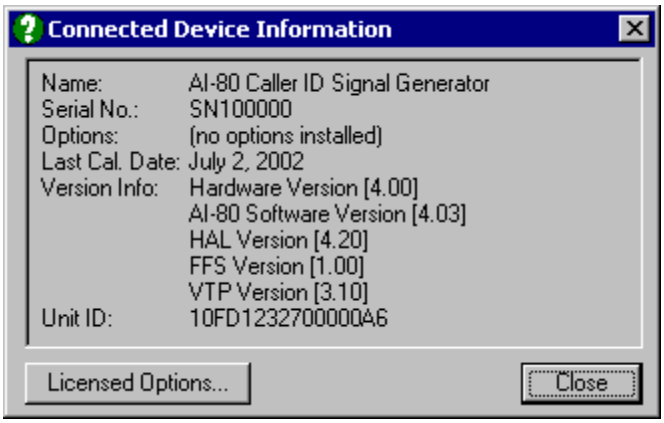
Note: Since the AI-80 is configured as a DCE (Data Communications Equipment), a "straight-through" serial cable is required. Do not use a "null-modem" or "cross-over" cable when connecting the PC to the AI-80.

Once the cable has been connected, ensure the AI-80 is turned on. Start the A.I.WorkBench software by clicking the START button on the Windows Task bar. Then select Programs, Advent Instruments, A.I.WorkBench. The program will automatically search the available communication ports and configure them for communication with the AI-80. If successful, the program will display the following status line:

If a connection could not be established, the status line will show the following:

For more detailed information on the serial communication between the PC and AI-80, along with trouble shooting tips, see Appendix D.

Once a connection is established to the AI-80, new programs can be developed and downloaded into its flash memory, or existing programs can be uploaded and saved to disk. Various details regarding the instrument can be viewed by selecting the [HELP] [DEVICE INFORMATION] menu command, as shown in the following figure.



Section 2 Using the AI-80

This section presents an overview of the AI-80 and some information on how to operate the unit. For information on developing custom programs for the AI-80 using the included A.I. WorkBench software, see section 3.

Section Contents:

Section 2-1 Overview

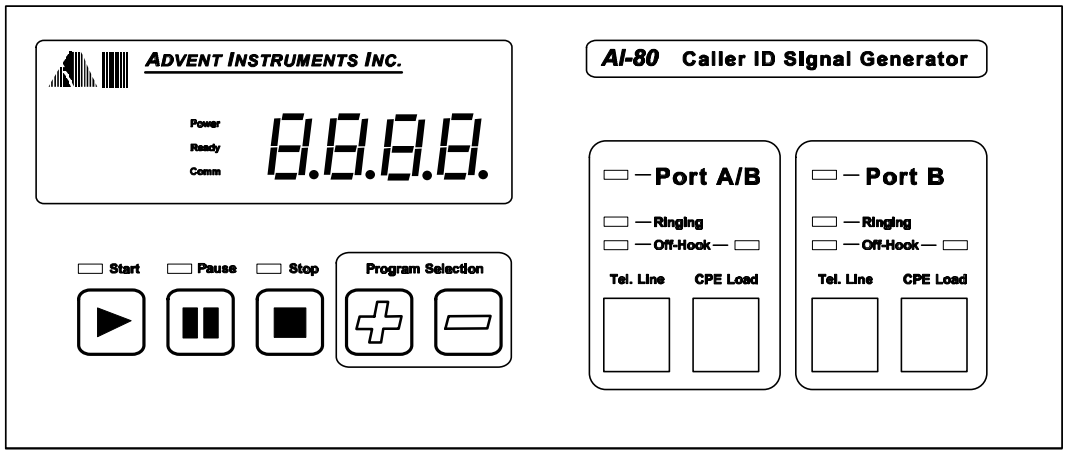
The AI-80 Caller ID Signal Generator is primarily designed for use in testing telephone devices in a production environment. It presents a simple user interface for ease of operation, along with a sophisticated degree of programmability to accommodate a wide range of test requirements.

Standard factory installed testing programs are built in the AI-80 for immediate use, while the A.I. WorkBench software can be used to customize the existing programs, or create new programs.

Front Panel Connections

The AI-80 has four standard RJ-11 telephone jacks located at the lower right corner of the front panel. The jacks are divided into two groups referred to as Port A/B and Port B. Only one of the two ports is active at any one time. For production situations, testing can occur on the active port, while the next unit to test is being connected to the inactive port. This assists in increasing through-put by eliminating the waiting time associated with attaching a new unit to the AI-80.

The two RJ-11 jacks associated with each ports are labeled “Tel. Line” and “CPE Load”. The Tel. Line jack is the output of the central office (CO) simulation circuitry. The CO simulator circuitry generates a DC voltage of 48 Volts across the tip and ring leads of the active Tel. Line jack. Along with the DC feeding voltage, the CO simulator can generate audio signals in the band of 20 Hz to 10,000 Hz, ringing signals up to 80 Vrms, reverse the line polarity, and an OSI (open switching interval).



The second RJ-11, labeled “CPE Load”, is connected to circuitry that simulates a CPE (Customer Premise Equipment) device. The CPE load can be programmed to be in either the on-hook or off-hook state, along with being able to measure audio and ringing signals. When testing Caller ID adjunct devices, or stutter dial

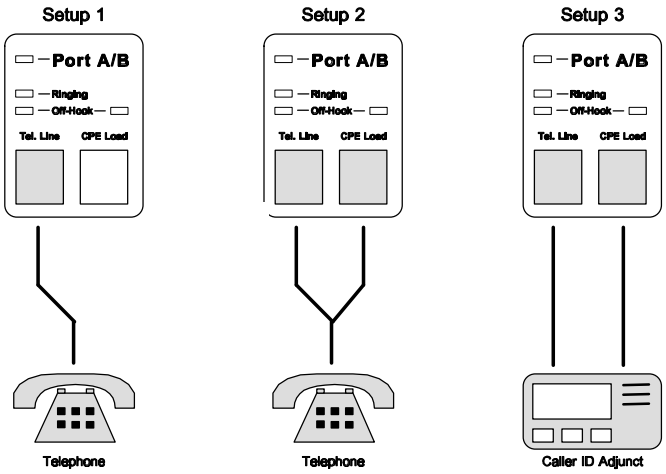
tone detection circuitry, the CPE load finds its use as a programmable telephone that can go on or off-hook during the testing procedure.

The most common connection configurations between the AI-80 and any device under test fall in one of three different variations. As shown below, the first is the simplest connection.

Setup 1, connects the Tel. Line jack to a telephone via a single cable. In this configuration, the AI-80 can perform various Caller ID tests and other telephone signaling tests, such as pulse & DTMF dialing, flash timing.

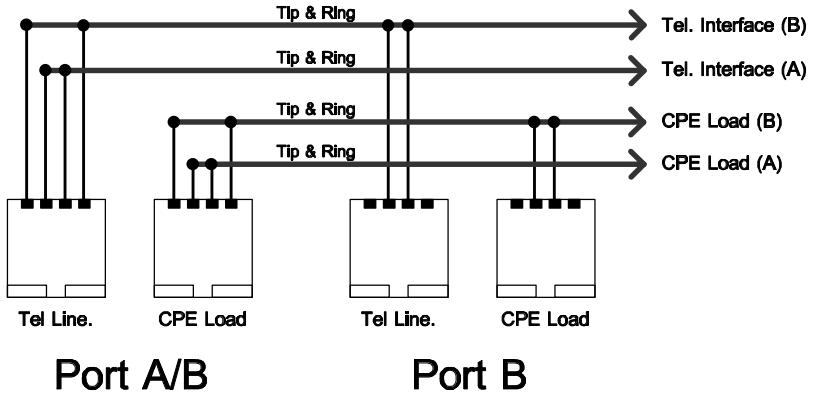
More sophisticated telephone devices may require stutter dial tone testing, or MEI (multiple extension in-use) testing. In this case, setup 2 connects the CPE Load jack in parallel to the telephone under test. For this type of testing, the CPE load is programmed to act as an extension phone that can go either on-hook or off-hook. Depending on the state of the CPE load, the telephone under test is then checked for proper response when receiving various signals.

The third common connection is setup 3. This is generally used in testing Caller ID adjunct devices, where a CPE is required to be connected to the telephone jack of the adjunct. Similar to setup 2, the adjunct is tested for proper response to various signals with the CPE load in either the on-hook or off-hook state.



The connection setups described above operate in the same manner for either port A or port B. Only one port can be active at any one time, and is indicated by the status of the LED beside the port label.

For added flexibility, multi-line telephones that utilize the outer pair of contacts on the RJ-11 jack as a second line, can also be tested with the AI-80. The two left most RJ-11 jacks that are Port A/B utilize the inner pair of contacts when port A is active and the outer pair of contacts when port B is active. Thus the Port A/B jacks are always active on either the inner conductors or outer conductors depending on which port is active. The following figure shows the logical connections between the RJ-11 jacks, the telephone interface, and CPE load ports.

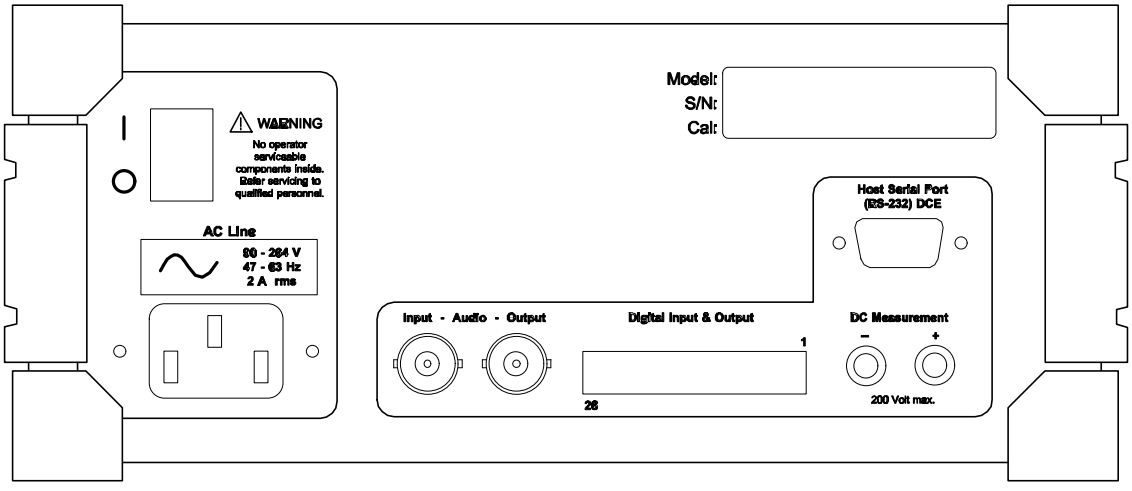


As shown above, the Port B jacks only have connections to the inner conductors and is active only when port B is active. However, the Port A/B jacks have connections to both ports for testing multi-line telephones that utilize the outer pair of conductors.

The four status LED's associated with each port are used to indicate which port is active, status of the ringing generator, if the CO simulator circuitry detects an off-hook condition, and the status of the CPE load hook switch. Though the Port A/B RJ-11 jacks are active with either port setting (on different wire pairs), the status LED's reflect the activity of port A only. Likewise, for the Port B RJ-11 jacks, the status LED's reflect the activity of port B only.

Rear Panel Connections

The AI-80 rear panel, as shown in the following figure, contains a standard AC power receptacle, power switch, host serial communications port, and various connectors associated with the optional modules.



For proper operation, the AC power source must have a voltage in the range of 90 Vrms to 264 Vrms and a frequency in the range of 47 Hz to 63 Hz. Operation outside this range may be unreliable and can damage the unit. As there are no

user serviceable components inside the AI-80, please refer servicing to qualified personnel. It is important that the AI-80 be properly grounded for safety and measurement repeatability reasons.

The 9 pin female connector, on the right side of the rear panel, is the host serial port. The port is configured as DCE (Data Communications Equipment) with standard RS-232 voltage levels. When connecting to a PC, no "null modem" or "cross over" cables are needed. It is possible to connect the host serial port to other serial devices, such as a printer. In this case, a "cross over" or "null modem" cable is required for proper operation. The pin connections for the host serial port are:

Pin 1 No Connection
Pin 2 Receive Data (output from AI-80)
Pin 3 Transmit Data (input to AI-80)
Pin 4 Data Terminal Ready
Pin 5 Ground
Pin 6 Data Set Ready
Pin 7 Request to Send (input to AI-80)
Pin 8 Clear to Send (output from AI-80)
Pin 9 No Connection

The AI-80 does not require any specific RTS (request to send) / CTS (clear to send) signaling in order to communicate with another serial device. The only required connections are Receive Data, Transmit Data, and Ground. The DTR (data terminal ready) and DSR (data set ready) pins are internally connected together and are not used by the AI-80. For more information on the host serial communications port, and trouble shooting tips, see Appendix D.

The optional I/O Module adds to the rear panel, two BNC connectors for audio input and output, a 26 pin IDC header for digital input and output, and two banana jacks for measuring DC voltages.

For more information on the optional modules, see section 2-3.

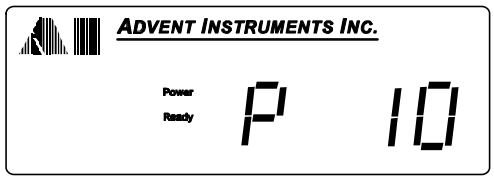
Section 2-2 Operation

Front Panel Controls

Once the AI-80 is turned on, it begins to load and execute various software components. During this process, the display area will flash the software revision for about half a second. In the situation where the unit is damaged, or some of the memory files are corrupted, the display will show “Err” followed by a number. Appendix C contains information on the meaning of the error codes along with possible remedies and solutions.

Once the AI-80 passes all of the internal checks, it is ready for use. The AI-80 can be operated by either the control of a PC via the host serial port, or by the five keys located on the front panel.

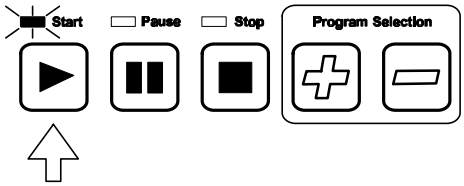
The unit contains within its non-volatile flash memory a series of program files, which perform specific testing functions. Various programs are built into the AI-80, and by using the A.I. WorkBench software, custom programs can be created and downloaded into the AI-80. Each program contained within the unit is assigned a unique number. Under normal operations, after power up the display shows the active program number. See Appendix B for details on each of the factory installed programs contained in the AI-80.

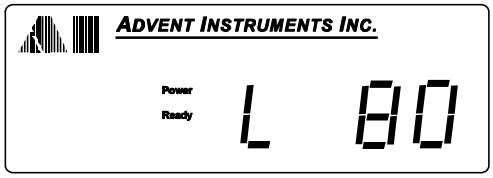


The program numbers are broken into two groups. Those between 10 and 99 represent the built in programs supplied with the AI-80. Program numbers above 99 are reserved for custom programs created with the A.I. WorkBench software. Pressing the program selection plus or minus keys will increment or decrement the program number. Program numbers do not need to be sequential, and unused numbers will be skipped by the program selection keys.

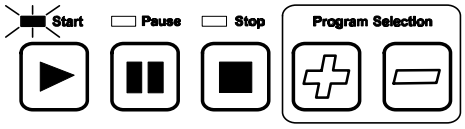
Once a program has been selected with the plus/minus keys, the Start, Pause, and Stop keys can be used to control its execution. Pressing the Start key will begin execution of the program, while Pause suspends any running programs, and Stop will terminate any running programs. The LED’s above the three keys indicate the status of selected program, as either running, paused, or stopped. When pressing the Stop key, the stop LED is only active momentarily.

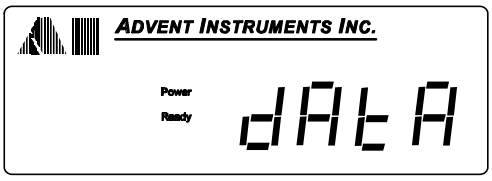
The default program at power up is number 10. This program sends an FSK based Caller ID message, consisting of date & time, calling number, and calling name, after two seconds of ringing. Pressing the Start key illuminates the start LED and begins executing the program, as shown below.





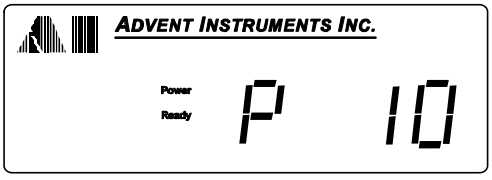
During the first two seconds of the program, the ringing generator will be active. At this time, the display indicates the ringing level, which is set to 80 Vrms. A short time following the ringing, an FSK modulated carrier is started, that transmits Caller ID information in a Multiple Message format. During the data transmission, the display shows “data”, as seen below.





At any time the program is executing, the Pause key can be pressed to suspend the program. To restart the program, from the pause mode, the Start key must be pressed. Pressing the Stop key at any time will terminate the program execution and reset all the AI-80 settings to their default values. The program selection plus/minus keys are disabled while a program is running or in pause mode.

Once the FSK data has been sent, the program is finished. The start LED is turned off, all the AI-80 settings are returned to their default state, and the program number is again displayed.



Many of the standard Caller ID programs are designed to automatically stop once the information has been sent. However, other included programs, such as measuring flash times, run indefinitely. These programs will not end themselves, and can be stopped at any time by pressing the Stop key. This will reset the AI-80 settings to their default values and return the display to the active program number.

Standard Programs

The AI-80 includes a number of built-in programs that send various Caller ID signals or perform common telephone related tests. The following table provides a listing of these programs with a brief description of each one. For more information on the usage and details of each program, see Appendix B.

To run any of the following programs, use the program selection plus/minus keys until the program number displayed matches the desired program. Then press the Start key to begin the program.

No. **10** Title: **Bellcore Type I - Multiple Message**

Ring Port A for 2 seconds, then send a FSK Caller ID message containing the date/time, calling number, and calling name in the Bellcore Multiple Message Format.

No. **11** Title: **Bellcore Type I - Single Message**

Ring Port A for 2 seconds, then send a FSK Caller ID message containing the date/time and the calling number in the Bellcore Single Message Format.

No. **12** Title: **Bellcore VMWI - Activate**

Send a FSK Caller ID message containing the Visual Message Waiting Indicator Activate command, in the Bellcore Multiple VMWI format.

No. **13** Title: **Bellcore VMWI - Deactivate**

Send a FSK Caller ID message containing the Visual Message Waiting Indicator Deactivate command, in the Bellcore Multiple VMWI format.

No. **15** Title: **Bellcore Type II - Multiple Message**

Generate a SAS tone, then CAS tone. Wait for up to 165 msec for an ACK tone. If the ACK tone is received, send a FSK Caller ID message containing the date/time, calling number, and calling name in the Bellcore Multiple Message Format.

No. **20** Title: **UK BT Type I CLIP Message**

Reverse telephone line polarity, then send DTAS tone for 100 msec. 150 msec after DTAS tone, send a FSK (V.23) Caller ID Call Setup message containing the date/time, calling number, and calling name. After FSK data generate two ringing bursts.

No. **21** Title: **UK CCA Type I CLIP Message**

Generate a ringing burst for 350 msec, then send a FSK (V.23) Caller ID Call Setup message containing the date/time, calling number, and calling name. After the FSK data generate two ringing bursts.

No. **22** Title: **France Type I CLIP Message**

Generate a ringing burst for 250 msec, then send a FSK (V.23) Caller ID Call Setup message containing the date/time, calling number, and calling name. After the FSK data generate two ringing bursts.

No. **23** Title: **Australia Type I (Ring Burst Alert)**

Generate a ringing burst for 400 msec, then send a FSK (Bell 202) Caller ID message containing the date/time, calling number, and calling name in the Bellcore Multiple Message format. After the FSK data generate two ringing bursts.

No. **24** Title: **Australia Type I (Line Reverse Alert)**

Reverse the telephone line polarity, then send a FSK (Bell 202) Caller ID message containing the date/time, calling number, and calling name in the Bellcore Multiple

Message format. After the FSK data generate two ringing bursts.

No. **25** Title: **China Type I (Odd Parity)**

Ring Port A for 2 seconds, then send a FSK Caller ID message containing the date/time, calling number, and calling name in the Bellcore Multiple Message Format, except that all ASCII characters are encode in 7 bits with odd parity.

No. **30** Title: **Japan NTT Type I**

Reverse the telephone line polarity, then generate the CAR ringing for up to 6 seconds. If the CPE goes off-hook, send FSK Caller ID message containing the calling number in NTT message format. Wait till CPE goes on-hook, then generate one ringing cycle.

No. **40** Title: **DTMF Caller ID (Line Reverse Alert)**

Reverse the telephone line polarity, then send a DTMF Caller ID message containing the calling number with a start code of D, and stop code of C. Then generate two ringing bursts.

No. **41** Title: **DTMF Caller ID (Ring Burst Alert)**

Generate a ringing burst for 500 msec, then send a DTMF Caller ID message containing the calling number with a start code of D, and stop code of C. Then generate two ringing bursts.

No. **50** Title: **Dial Tone Generation**

Wait till CPE goes off-hook, then generate 350 Hz / 440 Hz dial tone. Stop dial tone once CPE goes on-hook. Repeat until program stopped.

No. **51** Title: **Stutter Dial Tone Generation**

Wait till the CPE goes off-hook, then generate 350 Hz / 440 Hz stutter dial tone (100 msec on, 100 msec off, 10 cycles). Stop dial tone once CPE goes on-hook. Repeat until the program is stopped.

No. **60** Title: **Measure Flash Timing**

Wait till the CPE goes on-hook, then display “F”. When CPE goes off-hook, display the on-hook time in msec. Repeat until the program is stopped.

No. **61** Title: **Measure Pulse Dialing PPS**

During pulse dialing, display the digit count. Once finished, display the pulses-per-second. Then wait for the next digit. Repeat until the program is stopped.

No. **62** Title: **Measure Make and Break Times**

During pulse dialing, display the digit count. Once finished, display the break time in msec. followed by the make time in msec. Then wait for the next digit. Repeat until the program is stopped.

The following programs are meant to demonstrate the capabilities of the optional I/O module. If the I/O module is not installed, executing the programs will show “nA” on the front panel display.

No. **80** Title: **IO Module: Measure Analog Input Channels**

Measures and displays the voltage present on the four analog input channels. The input voltages can range from -10 to +10 volts. Use the + and - keys to cycle through the different input channels.

No. **81** Title: **IO Module: Measure DC Voltages**

Displays the voltage reading present at the rear panel banana jacks. Use the + and - keys to change the measurement integration time between 100 msec and 1.0 seconds. Longer integration times results in more stable readings if the signal has large common mode power line hum components.

No. **82** Title: **IO Module: Measure Signal Frequency**

Measures the signal frequency present at digital I/O pin 6 and displays the results in units of kHz. Using a gating time of 200 msec, the frequency resolution is 5 Hz. The applied signal must conform to standard TTL 5 volt signal level limits.

No. **83** Title: **IO Module: Measure Pulse Timing**

Measures the signal timing present at digital I/O pin 7 and displays the results in units of msec. By using the + and - keys the program can measure the time between rising edges, falling edges, positive pulse duration, and negative pulse duration. The applied signal must conform to standard TTL 5 volt signal level limits.

No. **84** Title: **IO Module: Generate Pulses**

Generates positive going pulses with a programmable duration between 1 msec and 1000 msec. The signal is present at the Timer Output pin on the I/O module 26 pin header. The pulse duration can be changed with the + and - keys. The speaker will beep while the pulse is active.

No. **85** Title: **IO Module: Generate Square Waves**

Generates square waves with a programmable frequency between 1 Hz and 10 kHz. The signal is present at the Timer Output pin. The frequency is adjusted be pressing the + and - keys.

No. **86** Title: **IO Module: Generate PWM Output**

Generates a pulse width modulated signal at the Timer Output pin. Using the + and - keys, the PWM output can be set from 0 to 1023, representing the minimum and maximum duty cycles. This signal can be used as a 10 bit analog voltage output by passing it through a low pass filter.

No. **87** Title: **IO Module: UART Echo**

This program accepts an asynchronous serial data steam at digital I/O pin 1 and after receiving a carriage return ASCII character (or 64 bytes), it will echo the received characters out on digital I/O pin 2. The baud rate and parity is programmable via the + and - keys.

External Start Trigger

Programs can be started by using an external trigger in addition to pressing the Start key. In situations where the AI-80 serial host port is unused, creating a connection between pins 7 (RTS) and 8 (CTS) of the 9 pin serial host port will simulate pressing the Start key.

Section 2-3 Optional Modules

The AI-80 can include a number of factory installed options. These options provide additional capabilities that may be required for some applications.

Complex Line Impedance

In addition to the standard 600 ohm and 900 ohm line impedances built into the AI-80, this option adds a complex line impedance. Specified at the time of order, the complex impedance can be one of the following values:

- Complex #1: 220 ohm + (820 ohm || 115 nF)
- Complex #2: 270 ohm + (750 ohm || 150 nF)
- Complex #3: 370 ohm + (620 ohm || 310 nF)

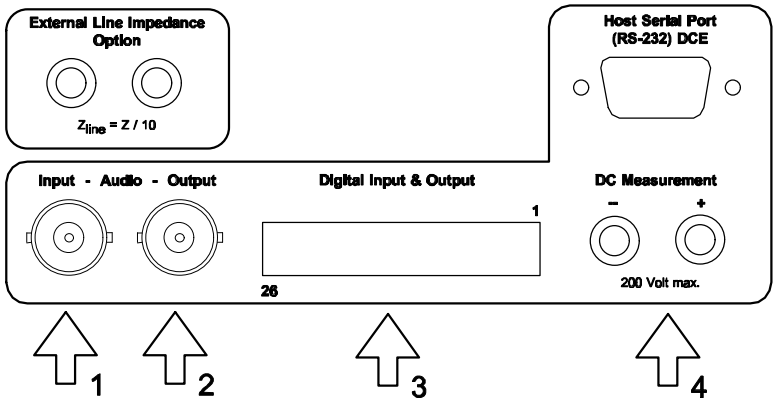
Analog & Digital I/O

The Analog & Digital I/O module provides a number of expanded capabilities to the AI-80. Applications include utilizing the AI-80 in small scale ATE systems, where by the AI-80 can interface and control other circuitry in order to create a basic automated testing environment. Closed loop CPE testing may also be possible via the digital I/O, provided an interface to the CPE is available.

The optional module provides the following functionality:

- Audio Input Port [1]
- Audio Output Port [2]
- DC Voltage Measurement [4]
- Digital Output Port (8 bits) [3]
- Digital Input Port (8 bits) [3]
- Digital I/O Port (7 bits) [3]
- Analog Voltage Input Channels (4) [3]
- Analog Comparator Input Channels [3]
- Pulse Generation and Timing [3]
- Dual PWM Output [3]
- Asynchronous Serial Communications Port [3]
- Non-Volatile Memory for Program Data

All of the optional ports are located on the rear of the AI-80, as shown in the figure below. Two BNC connectors, labeled 1 and 2, provide the audio input and output ports respectively. The 26 pin boxed header, provides all of the digital I/O and analog input channel signals. Finally, the two banana jacks labeled 4 are used to measure DC voltages.



Analog Input & Output

Signals applied to the BNC audio input can be measured by the AI-80, or mixed with the internal tone generators and routed to the telephone interface port. This gives the ability to apply custom signals to a CPE under test. The BNC audio output port can be used to monitor the signals present at either the telephone interface, CPE load interface, or internal tone generators. The function of the audio I/O is programmable via the A.I.WorkBench software. See section 4-2 for more information on programming the audio I/O.

Digital Input & Output

The 26 pin boxed header provides all the digital I/O signals, as well as the analog input channels. Unless otherwise stated, all the I/O signals must conform to standard 5 volt TTL logic levels. Pin 1 of the connected is located in the upper right corner, while pin 26 is in the lower left corner. The figure below shows function of pin on the connector. The function of the digital I/O is programmable via the A.I.WorkBench software. See section 4-2 for more information on programming the digital I/O.

25	23	21	19	17	15	13	11	9	7	5	3	1
Do2	Do4	Do6	Do8	Dio2 TxD	Dio4 PW2	Dio6 Pci	Tout PW1	Di2 Ac-	Di4	Di6 Ain2	Di8 Ain4	Gnd
26	24	22	20	18	16	14	12	10	8	6	4	2
Do1	Do3	Do5	Do7	Dio1 RxD	Dio3	Dio5	Dio7 Pti	Di1 Ac+	Di3	Di5 Ain1	Di7 Ain3	Gnd

The following table describes the functions of each of the I/O pins:

Pin(s)	Function
1-2	**Signal and earth ground:** These two pins are connected to earth ground and represent the ground reference for all the other pins.
3-6	**Digital Inputs 5 to 8 and Analog Input Channels 1-4:** Input signal only. Can be used as 5 volt TTL inputs, or analog voltage inputs. The input impedance is approximately 100 kohms with a maximum input voltage range of +/- 10 volts.

7-8 **Digital Inputs 3 to 4:**
Input signal only. Digital 5 volt TTL inputs.

9-10 **Digital Inputs 1 to 2 and Analog Comparator Inputs:**
Input signal only. Can be used as 5 volt TTL inputs, or inputs to an analog voltage comparator. Pin 10 is the comparator positive input, while pin 9 is the comparator negative input. The comparator voltage input range is 0 to 5 volts.

11 **Timer Output and PWM Channel 1:**
Output signal only. This pin can be used to output digital pulses of programmable duration, or continuous square waves of a programmable frequency. As an alternate function, it can be set as a 10 bit pulse width modulator (PWM) output.

12 **Digital Input/Output 7 and Pulse Timer Input**
Input or output signal. This pin can be programmed as a fixed digital input or output. As an alternate function, it can be used to measure digital pulse durations between rising edges, falling edges, positive pulses, and negative pulses. In the input modes, the signal must conform to +5 volt TTL levels.

13 **Digital Input/Output 6 and Pulse Counter Input**
Input or output signal. This pin can be programmed as a fixed digital input or output. As an alternate function, it can used to count rising edges in either a free-running mode or time gated mode of operation. In the input modes, the signal must conform to +5 volt TTL levels..

14 **Digital Input/Output 5**
Input or output signal. This pin can be programmed as either an input or output.

15 **Digital Input/Output 4 and PWM Channel 2:**
Input or output signal. This pin can be programmed as a fixed digital input or output. As an alternate function, it can be used as a 10 bit pulse width modulator (PWM) output.

16 **Digital Input/Output 3:**
Input or output signal. This pin can be programmed as either an input or output.

17 **Digital Input/Output 2 and Async Serial Transmits Data Output:**
Input or output signal. This pin can be programmed as either an input or output. As an alternate function, it can be used for asynchronous serial communications as a transmit data output pin.

18 **Digital Input/Output 1 and Async Serial Receive Data Input:**
Input or output signal. This pin can be programmed as either an input or output. As an alternate function, it can be used for asynchronous serial communications as a receive data input pin.

19-26 **Digital Output 1 to 8:**
Output signal only. These 8 pins are fixed 5 volt TTL level digital outputs.

DC Voltage Measurement

Two rear panel banana jacks can be used to measure DC voltages up to +/- 200 volts. The input impedance of the banana jacks are approximately 1 Mohms to earth ground, and 2 Mohms with respect to each other. The maximum voltage between either input and earth ground is +/- 200 volts. Care must be taken not to exceed this limit. The DC voltage measurement function is programmable via the A.I.WorkBench software. See section 4-2: IO Object, for more information on programming the this function.

Data Storage

In addition to the input and output functions, the IO module adds the capability for any AI-80 program to store and retrieve up to 32 numeric values or up to 128 characters in the IO module's non-volatile storage. This can be used by programs to save setup or configuration information, as the data is retained even if the AI-80's power is removed.

Bell 202, V.23 FSK Decoder

This optional component adds the capability to receive and decode FSK modulated serial data. It can capture up to 700 bytes of asynchronous serial data encoded by either Bell 202 or V.23 Frequency Shift Keying (FSK). This expands the AI-80's testing capability by including two-way FSK applications such as ADSI and SMS.

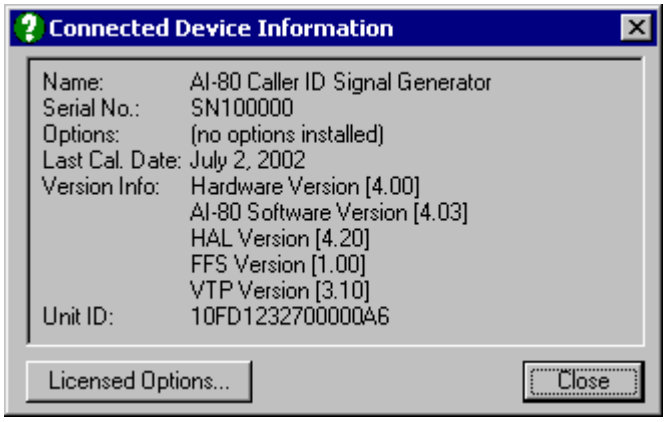
As this option is a software component, it can be added to any AI-80 (revision 3 software or later) at any time.

Enabling The FSK Decoder

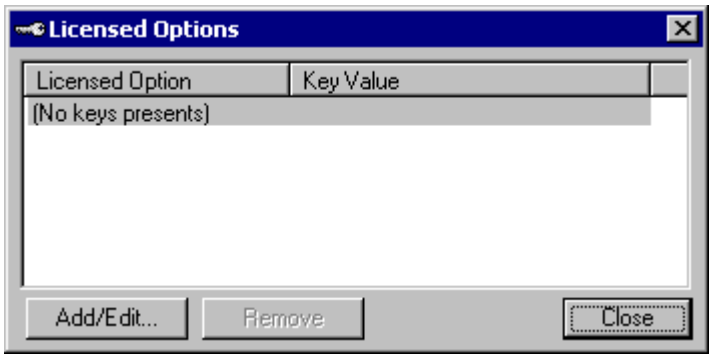
To enable the FSK decoder, a software key must be entered into the AI-80. This is performed with the A.I.WorkBench software with the following steps:

1) Connect the AI-80 to the PC and execute the A.I.WorkBench software.

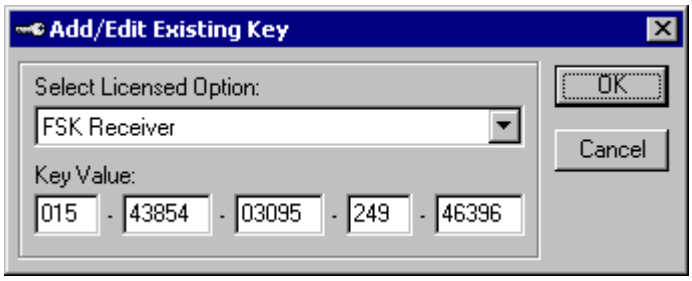
2) Select the [HELP] [DEVICE INFORMATION] menu command. This shows the following window:



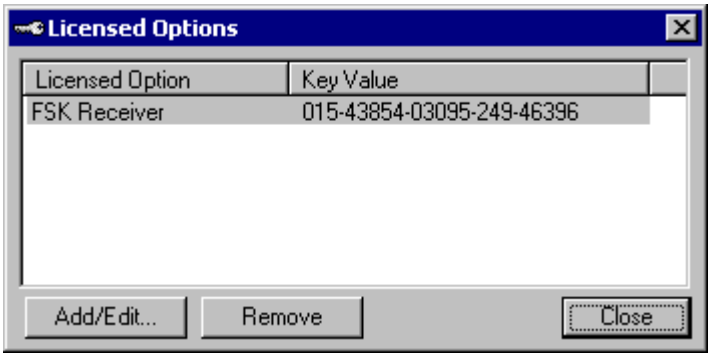
3) Click the mouse on the "Licensed Options" button in the lower left corner of the window. This reads all the software keys from the AI-80 and displays their contents.



4) To enter the key required by the FSK decoder, click the mouse on the "Add/Edit" button. In the displayed window, enter the provided software key and click the "OK" button.



5) The key value entered is now shown in the Licensed Options window. To make any changes click the "Add/Edit" button. Check to ensure the key value is correct. Once verified, click the "Close" button. This saves the key contents to the AI-80.



6) As the AI-80 only checks for the software keys upon power up, exit the A.I.WorkBench software. Then turn off the AI-80, wait a second, and turn on the AI-80. If the correct key value has been entered, the display momentarily flashes "So 1" following the software version.

Using the FSK Decoder

For information on how to use the FSK decoder in a program, see section 4.2 Hardware Abstraction Layer. The description of the "FSK" properties explains the decoder can be used to receive data bytes from external devices.

Section 3 Using the A.I. WorkBench Software

This section describes how to use the A.I.WorkBench software to create and modify programs for the AI-80. Detailed reference information on the programming language, AI-80 hardware abstraction, and system software functionality is described in section 4.

Section Contents:

- Connecting to the AI-80
- Creating a new program
- Working with the flash memory
- Working with projects
- Using the source file editor
- Executing programs

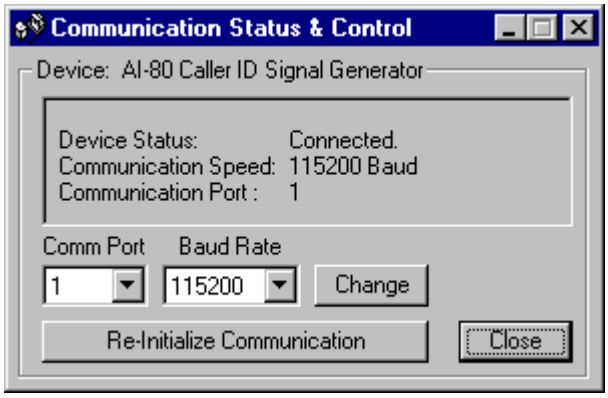
Section 3-1 Connecting to the AI-80

Once the A.I.WorkBench program starts, it attempts to establish communications with the AI-80. Though not necessary to develop and compile programs, a connection is needed to transfer any program or data to and from the host PC and AI-80. To simplify the complexities of establishing a serial RS-232 link with the AI-80, the program automatically scans all communications ports between COM 1 and COM 4, along with configuring the port settings, such as baud rate, parity, and number of stopbits, to the correct settings.

If an AI-80 has been connected property to a host PC, and the unit is turned on, the status bar of the A.I.WorkBench program should show the following

This indicates that a successful link is setup, and that the AI-80 is ready to initiate transfers of data to and from the host PC. The communications port number and port settings are saved to a configuration file when the A.I.WorkBench program is closed. As such, the next time the program is started, it will automatically use the last settings in searching for an AI-80. If unsuccessful, it will fall back to scanning through all the available ports on the PC.

At any time, the status of the communications link with the AI-80 can be viewed by selecting the [VIEW] [DEVICE COMMUNICATIONS LINK] menu command. This displays a window similar to the figure below. The current status of the connection is displayed, along with options to change either the communications port used, or the baud rate setting.

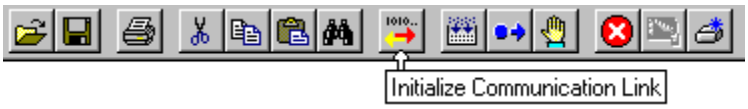


While this window is active, it will automatically test the connection at regular intervals. Disconnecting the AI-80 or turning it off will result in a lost connection, which will be reflected in the above status window.

To change the communications port or baud rate settings, simply select the new settings from the two pull down boxes and click the mouse on the Change button. The host PC will then attempt to re-establish communications using the new settings.

If the AI-80 becomes disconnected for extended periods of time, turned off, or another AI-80 is connected to the host PC, the communications link will be broken. To reconnect to the AI-80, click the mouse on the Re-Initialize Communication button.

A toolbar short cut can also be used to re-establish lost communications with the AI-80. By clicking the mouse on the indicated tool bar button below, the Communication Status & Control window will appear and attempt to restore communications with the connected AI-80.



Section 3-2 Introduction to Programming

The basic concepts in creating a program for the AI-80 are explored in this section. Later sections explain in greater detail the process of writing programs. All the steps needed to create a program are described in a tutorial fashion. It is assumed an AI-80 has been connected to the host PC, and that a successful communications link is established.

The steps needed to create a program for the AI-80 can be broken down as follows:

Step:
1. Create a new project file for the program
2. Create the program source file(s)
3. Compile the program, and correct any errors detected
4. Load the program into the AI-80's memory for debugging
5. Execute and debug the program
6. Once error free, load the program into the AI-80's flash memory

Creating a New Project

Before a program can be written, a project file must be created. The project file contains various details on the program, such as its title, program ID number, number of source files, compiler options, and debugging options. Each program is treated as a separate project, and requires a distinct project file. It is possible to use an existing project file, and modify its contents to suit a new program. However, for the purpose of this tutorial, a new project file will be created.

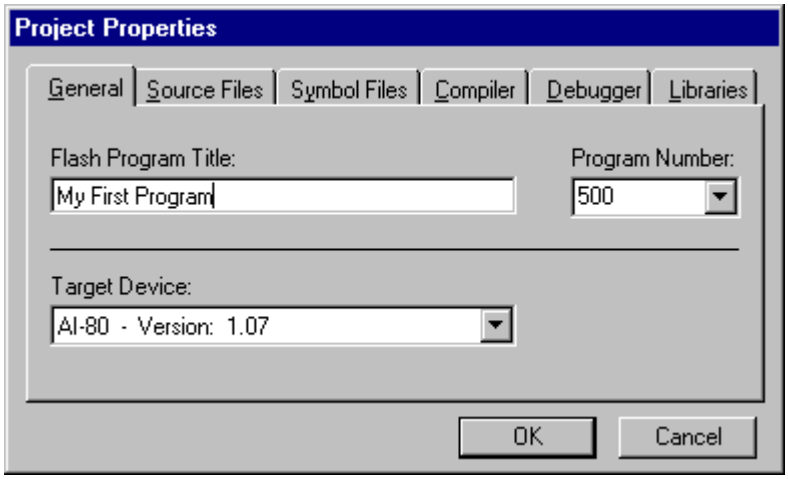
Step 1: From the FILE menu, select NEW PROJECT

Note: If a project is already loaded into the A.I.WorkBench program, and a source file has been changed, the program will ask if the current project should be saved before creating a new project.

Selecting NEW PROJECT automatically displays the Project Properties window. All of the various project settings are shown here and can be changed as desired. For most common programs, only the information in the General tab needs to be specified. The Flash Program Title refers to the displayed name of the program, when viewing the file directory of the AI-80's non-volatile memory. The Program Number is the key manner in which the AI-80 identifies each program. As such, it must be a unique number for each program contained in the AI-80. Using an existing program number, will result in the older program being over written when the new program is saved into the AI-80's flash memory.

Note: Before saving a program into the flash memory, the A.I.WorkBench program will request confirmation if a program of the same ID number already exists.

As a general convention, the program numbers should fall under one of two groupings. The first group of program numbers from 10 to 99 is reserved for standard programs that are factory loaded. Program numbers from 500 to 599 is reserved for user programs.



The Target Device pull down box is used to specify the output format for the compiler. It should be set to show "AI-80". It is important the version number displayed is not greater than the software version number of the AI-80 connected. If so, the compiler may use instructions that are not available in AI-80 being used. The A.I.WorkBench is always distributed with the latest AI-80 software version files as shown in the pull down box. As such, the AI-80 can be upgraded to match the same version as the compiler. To upgrade the AI-80 software, see Section 4-6: Upgrading the AI-80 Software.

To continue with this project:

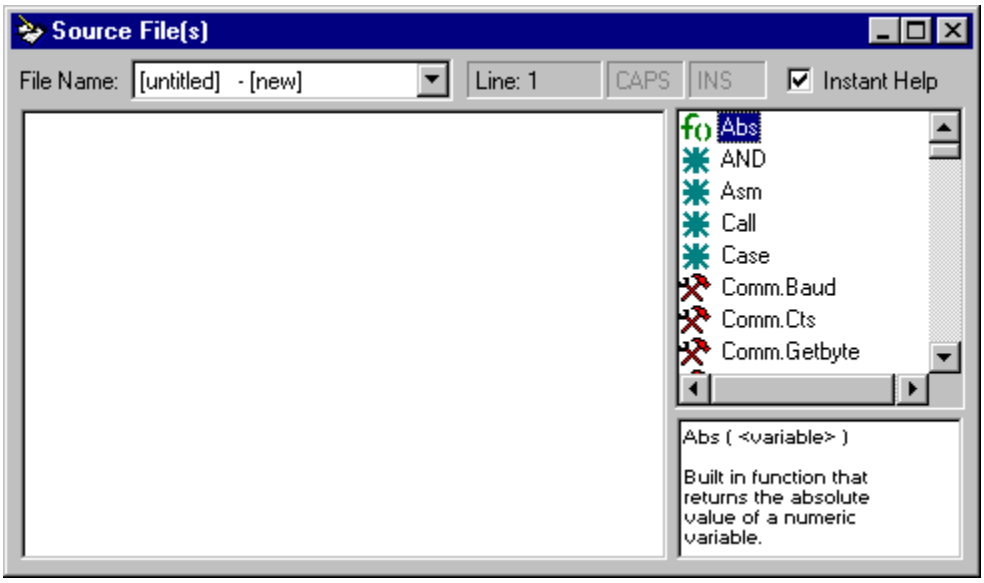
Step 2: Set the Flash Program Title to: My First Program.
Set the Program Number to 500.
Click the OK button.

Clicking OK will close the Project Properties windows, and display the Source File editor window. The editor is used to write the program source files. It supports working with multiple files, and includes a feature to ease programming by providing immediate help on all language commands and statements.

Writing the Program

As shown below, the editor will display an empty file titled "[untitled] - [new]". This indicates that no file name has been given to the source file and it is a new file. For very large projects, the program can be broken into multiple source files. This provides a easy way to organize and collect common program modules, by keeping them in the same file. When creating a new project, only one source file

is created. If more files are desired, they can be added in the Project Properties window. Only one source file will be used in this example program.



The Instant Help feature of the editor uses the right side of the window. It can be turned on or off by selecting the check box in upper right corner of the editor. When enabled, the Instant Help will list all the programming language keywords, functions, subroutines, and built-in variables available. Clicking on any of the items, displays additional information on its use in the text area below the list. In the above figure, a brief description of the built-in absolute (Abs) function is described.

When typing a keyword in the editor window, the Instant Help list box will attempt to find a matching entry in its list. It highlights its best guess, and displays any additional information available. Pressing the TAB key will complete the word automatically, by transferring it to the program listing. Also, double clicking the mouse on any item in the Instant Help list will copy it to the program listing.

The example program will perform a simple function by ringing a telephone connected to the AI-80 Port A. If the telephone goes off-hook, the program will detect this and generate a dial tone. A listing for the sample program follows below.

Step 3: Type the following example program into the source file editor. Note, this same project has been included in the A.I.WorkBench distribution. To load the project and source file (instead of typing it in), do the following:

a) Select the [FILE] [OPEN PROJECT] menu setting. If the programs asks to save the contents of the project created, select NO.

b) From the \PROJECTS directory, select the project file: FirstProgram.prj and click the OPEN button.

The structure of the sample program is relatively straight forward. All program lines that start with an asterisk '*' or semi-colon ';' are treated as comment lines and ignored.

```
* Title:  My First Program
* Program Number:  500

* This program will ring the telephone line until a connected
* telephone goes off-hook.  At that point, the program
* will generate a dial tone.  Once the telephone goes back
* on-hook, the program will end

Const True = 1
Const False = 0

Process Start

        ;set the ringing parameters and turn on the ringing gen
        Let Ring.Freq = 25.0
        Let Ring.Level = 60.0
        Let Ring.Enable = True

        ;wait till the phone goes off-hook, then turn off ringing
        Loop
                If Telint.Hookdetect = True Then
                        Exit Loop
                End If
        End Loop
        Let Ring.Enable = False

        ;generate a dial tone (440 and 350 Hz)
        Let Tonea.Freq = 440
        Let Toneb.Freq = 350
        Let Tonea.Level = 0.1           ;level at 0.1 Vrms
        Let Toneb.Level = 0.1           ;level at 0.1 Vrms
        Let Tonea.Enable = True         ;turn on tone
        Let Toneb.Enable = True         ;turn on tone

        ;wait for telephone to go on-hook, then end program
        Loop
                If Telint.Hookdetect = False Then
                        Exit Loop
                End If
        End Loop

End Process
```

The body of the program is contained within a PROCESS block. All of the program statements, for any program, must reside inside either a process, subroutine, or function block. Since the AI-80 supports parallel execution of multiple programs, the process block identifies the statements associated within a single process. Most programs require only one PROCESS block; however, more complex programs can divide their task into multiple PROCESS blocks, which execute in parallel. For more information on the programming language, see Section 4-1.

The program can be broken down into 4 logical sections. The first, simply sets the desired ring generator parameters, and enables the ring generator. The next

section consists of a loop where the program will wait indefinitely till the connected telephone enters the off-hook state. Once off-hook, the loop is exited, and the ring generator disabled. The third section generates a dial tone. This is accomplished by setting tone generators A and B to 440 Hz and 350 Hz respectively, and then enabling them. Finally, the fourth section of the program consists of a loop waiting for the telephone to go on-hook. The program will wait indefinitely for this to occur. Once on-hook, the program terminates when it reaches the END PROCESS statement.

At this stage, the above program listing should be contained in the Source File Editor window. The next step is to compile the program.

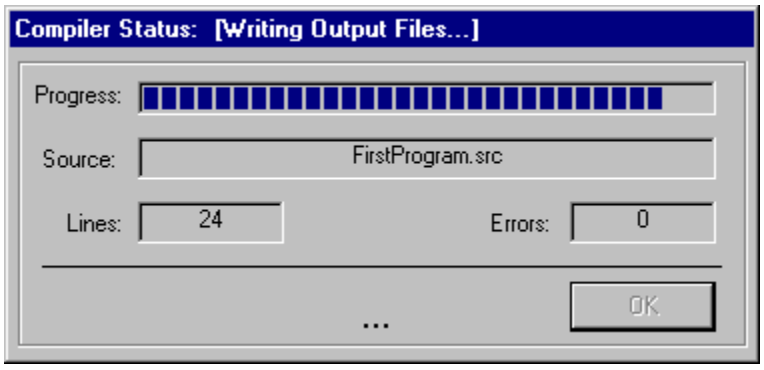
Compiling the Program

The compiler converts the high level language statements contained in the source file(s) into the native low-level language used in the AI-80. It is started by selecting the [RUN] [COMPILE] menu command, or by pressing Shift-F5.

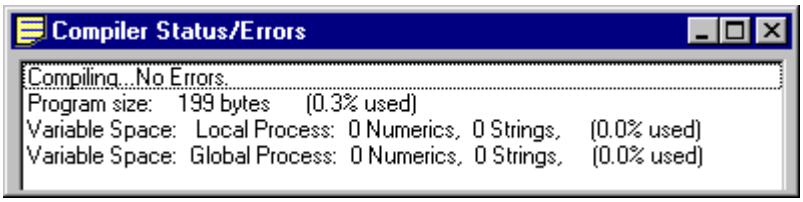
Step 4: Start the compiler by selecting the [RUN] [COMPILE] menu command.

Note: Invoking the compiler will always save the current project file settings and source file contents to disk. When creating a new project with an untitled project and/or source file, you will be asked to enter a name for the project and/or source file. For this example, use the name of “FirstProgram” for both the project file and source file.

A status window shows the progress of the compiler as it processes the source file(s). It indicates the number of lines compiled along with a count of the number of errors detected. One of the default settings in the project file will stop the compiler after the first error. As a result, if an error is detected, the compilation process is immediately stopped.



Once the compiler has finished, or has detected too many errors, the status window will disappear and the results of the compilation are displayed in the Compiler Status/Error window. If the above program was entered correctly, the window will report that no errors were detected, as shown below.



Upon a successful compilation, the status window displays the program size and number of variables used along with their respective usage percentages.

In case the compiler does find an error in the source file(s), it will display the error message in the status window, along with the line number and source file where the error was located. Clicking the mouse on the error message will highlight the error in the source file editor.

Step 5: After the compiler has finished, ensure that no errors were detected. If an error was reported, click the mouse on the error message to highlight the offending line in the source file. Ensure the highlighted source file line matches the program listing shown above. Re-compile the source file and proceed when error free..

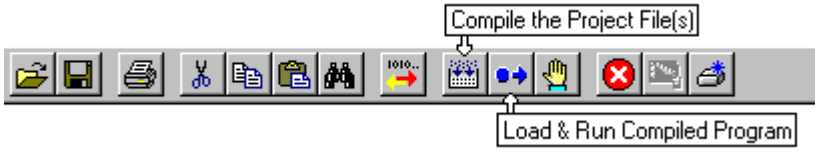
Loading and Running Programs

Once a program has been compiled without error, it can be loaded into the AI-80's memory and executed. Though a project can be compiled without an attached AI-80, in order to load the program into the AI-80, the host PC must have an established communications link.

Step 6: Select the [RUN] [LOAD & RUN] menu command, or press the F5 key. This transfers the program into the AI-80 and starts it. Any telephones connected to the Tel. Line RJ-11 jack on Port A should start to ring.

The telephone will ring until it goes off-hook. At that time it will start to generate a dial tone. Once the telephone goes back on-hook, it will end the execution of the program.

Two toolbar shortcuts can be used to either compile the program, or load & run it. These are shown below. It is also possible to use the [COMPILE & LOAD & RUN] menu command, which performs both steps automatically. A keyboard short cut for this command is Ctrl-F5.



At this stage, the program can be modified and/or debugged by making source file changes, compiling the program, and executing it in the AI-80. Once the program is operating as desired, it can be loaded into the AI-80's non-volatile

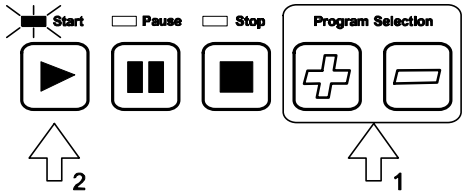
memory storage. Programs stored in the non-volatile memory can be executed without the use of the host PC.

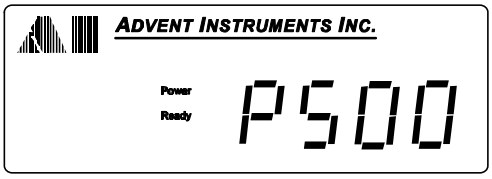
Saving the Program In Flash Memory

In order to use any programs when the AI-80 is not connected to the host PC, the program must be stored in its non-volatile flash memory. Programs are identified by their program number, which is specified in the Project Settings window. When storing a program into the AI-80 flash memory with the same number as an existing program, the existing program will be overwritten with the new program. Before this occurs, the A.I.WorkBench program will request confirmation.

> *Step 7: Save the program to flash memory. Select the [FILE] [LOAD PROGRAM INTO FLASH] menu command, or press Ctrl-W.*

In this example, the program number was set to 500. Once the program has been stored in the flash memory, it can be accessed and executed by the front panel keys. Using the Program Selection Plus/Minus keys, the displayed program can be incremented to 500. Pressing the Start key will begin execution of the program by ringing the telephone line. The program can stopped anytime by pressing the Stop key, or suspended by pressing the Pause key.





At this point, the connection to the host PC is no longer required. The AI-80 can be turned off, disconnected from the host PC, turned back on again with the new program still stored in the unit.

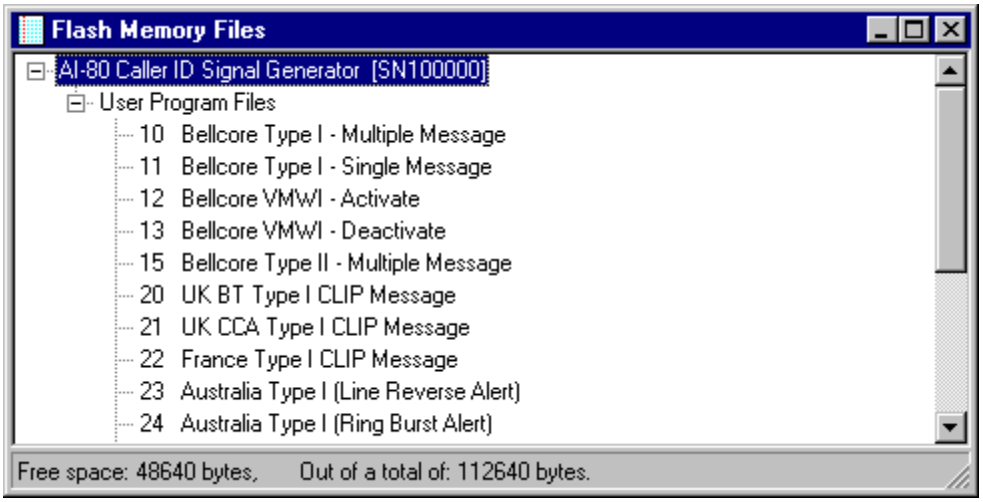
These are the basic steps used to create programs for the AI-80. The programming language of the AI-80 is very flexible and can perform a wide range of functions. Even the programs that provide some of the system functions are written in the same programming language. Thus it is possible to completely customize the AI-80 for various applications. Section 4 provides more details in the programming language as well as an overview to the system files and how they can be customized.

Section 3-3 Working with the Flash Memory

The flash memory within the AI-80 provides a means to execute custom developed programs without the need for a permanent connection to a host PC. In some production related applications, it is far more convenient to have the AI-80 perform a pre-programmed task without any additional equipment.

Listing Files

Managing the contents of the flash memory is done via the Flash Memory Files window within the A.I.WorkBench software. Clicking the mouse on the [VIEW] [FLASH MEMORY FILES] menu command displays a window similar to the figure below. The AI-80 is queried for a list of its files which are then displayed in a simple hierarchical manner.



Normally, only the User Program Files are displayed. These are the programs created with the A.I.WorkBench software to perform various functions and tasks in testing telephone related equipment. Other files, such as system related files are also stored within the flash memory, but not normally displayed. The User Program Files are listed by program number and title. The program number is used to reference the various programs, and each program number must be unique. It is defined within the project settings, and can be changed if required.

Though normally, only the User Program Files are displayed, it is possible to display and modify some of the system level files. These files control and set the user interface of the AI-80 and may be modified by the A.I.WorkBench software. Applications for this would be the optimization of the software for very specific and repetitive tasks. For more information on how to display the system files,

their operation and function, and common customization techniques, see section 4-3 System Software Overview

Note: Depending on the file sizes, some file operations can consume large amounts of time to execute when the AI-80 communication speed is set to slow baud rates, such as 9600. If possible, the baud rate should be increased to a higher rate in order to minimize file transfer times. See section 3-1: Connecting to the AI-80 for more information.

File Operations

With the Flash Memory Files window opened, the following basic file operations can be performed in order to manage the file contents.

- Load a file into the AI-80
- Read a file from the AI-80
- Delete a file from the AI-80
- Display a file's properties

The AI-80 program files are stored on the host PC using a special file format, with the .apf file extension. These files include not only the program object code, that the AI-80 executes, but additional information such as the program number, title, compiler object code file name and its date and time stamp. The .apf files can only be created in two manners. The first is by the compiler, which if no errors were detected, creates the file. The second is by reading a file from the AI-80's flash memory and saving it to disk.

To load a file into the AI-80, select the [FILE] [LOAD FILE INTO FLASH] menu command. A window will appear from where the .apf file can be chosen. If the selected file contains a program number that already exists within the connected AI-80, confirmation is requested before proceeding. Since every file number must be unique within the flash memory, loading a file with an existing program number will over write the existing file.

To read a file from the AI-80, first click mouse and highlight the file to save in the Flash Memory Files window. Then select the [FILE] [READ FILE FROM FLASH] menu command. A file name, with a .apf extension, must be then chosen. The file contents from the AI-80 will then be transferred to the host PC and saved with the file name chosen. Storing a copy of the AI-80 files on a PC can serve as a backup function or allow for transferring the files to another AI-80.

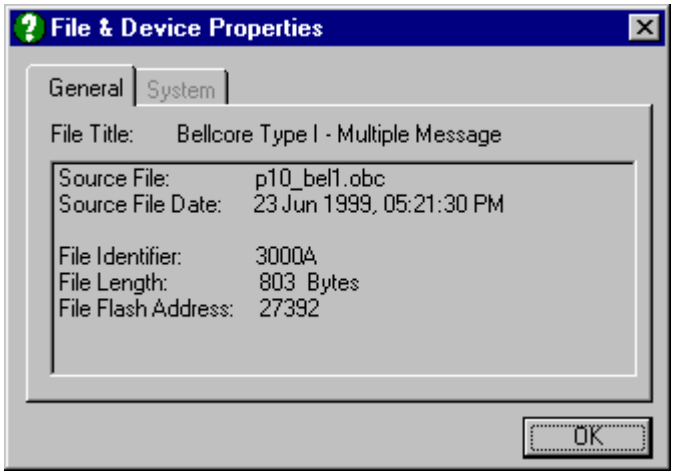
If a file is no longer required within the AI-80, it can be deleted by first highlighting the file in the Flash Memory Files window. Then select the [FILE] [DELETE FILE FROM FLASH] menu command. A confirmation is always requested before the file is deleted, as once deleted, the file can not be recovered.

Note: If the need to transfer or delete a large number of files from the AI-80 arises, it can be much faster to create a simple ASCII text script that lists in sequence the file operations needed. See the section 4-4: Auto-Config Command Files, on how to create and use these command files.

File Properties

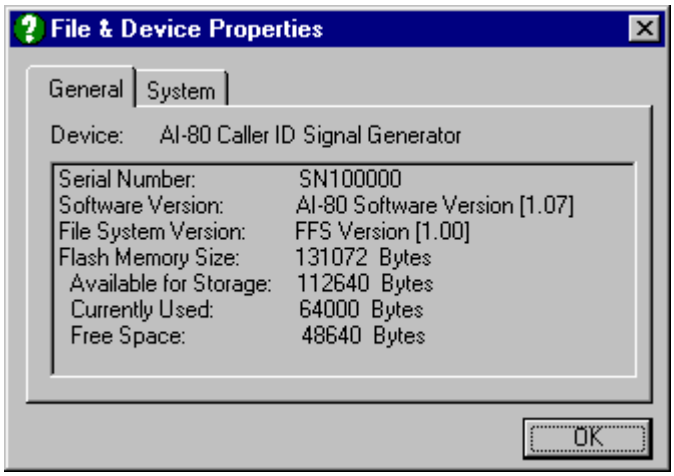
The file properties window displays additional file information that is not shown in the normal listing. Usually the most important information displayed is the file's source. For programs, this is the name of the object code file, which is the same name as the project file. Thus it is possible to determine what project file is associated with a program contained in the flash memory. Also, the date and time the source file was created is included as a file property. This finds use in tracking revisions of the source file and ensuring the latest file is contained in the AI-80's flash memory.

File properties are displayed by clicking the mouse on the desired file in the Flash Memory Files window, and then either pressing the right mouse button or selecting the [FILE] [PROPERTIES] menu command. A window similar to the figure below is displayed.



The file title and its source file name with date/time is shown, along with additional data such as the file identifier, length, and address. Normally this additional information is only needed for debugging purposes.

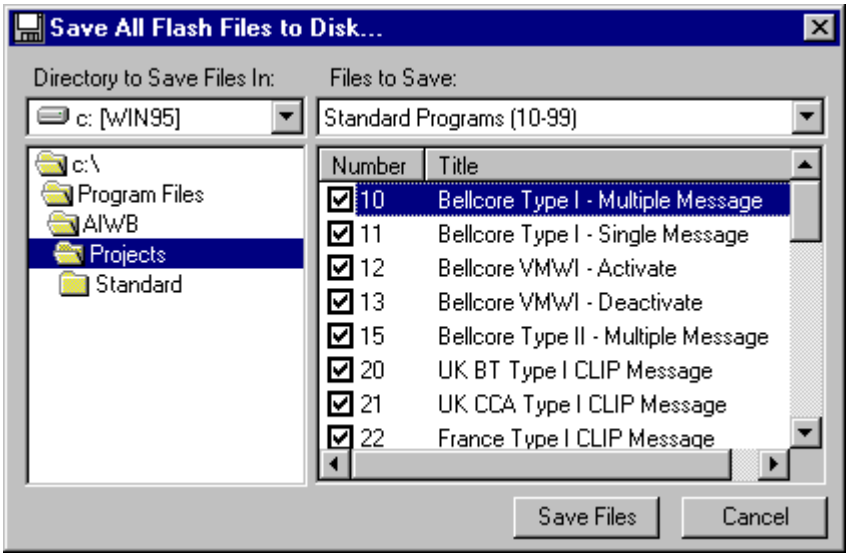
Though not a file, some information of the connected AI-80 is shown when displaying its properties. This is accomplished by clicking the mouse on the first item in the Flash Memory Files window, and selecting the [FILE] [PROPERTIES] menu command.



This displays the serial number of the AI-80 in addition to its software version and current utilization of the flash memory.

Saving All Program Files

At times it may be convenient to read all the program files from an AI-80 and save them to a PC file. Instead of reading a single file at a time, multiple files can be read by selecting the [FILE] [READ ALL FILE(S) FROM FLASH] menu command. This displays a dialog window similar to the following.



To save the AI-80 files, choose a directory to save the files to from the list boxes on the left side of the above window. Then select the individual files to save from the right side list box. The program number and title for each file stored in the flash memory is listed. If the program has a check mark beside its number, it will be saved. To prevent saving a particular program, click the mouse on its check box. This toggles the check mark on and off.

Section 3-4 Working with Projects

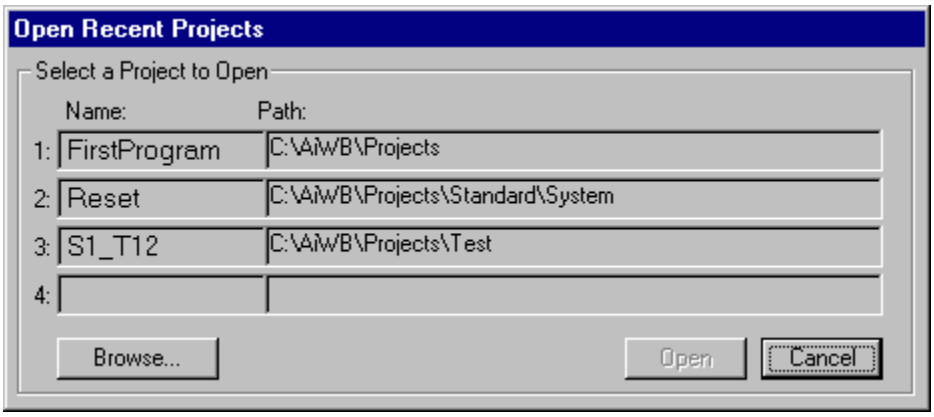
This section describes how to work with various projects. A project consists of a series of compilation options and one or more source files, where the final result is a program that can be loaded into an AI-80 and executed. All the various settings needed to produce a single executable program are contained within the project file.

New & Existing Projects

The first step in creating any program for the AI-80 is to create a new project file. This can be done in two ways. The first is to use all the default settings, by selecting the [FILE] [NEW PROJECT] menu option. This creates a new project, as of yet untitled, with all the default settings and one untitled source file. The default project settings are designed to be used with any AI-80 user program, which is the most common case. Programs that are designed to perform system level functions may require changes to the default project settings.

The second method to creating a new project is to open an existing project, along with all the source files, and save the project to a new file name. If the functionality of the new program is similar to an existing program, this method is much simpler. Once the project has been copied to a new file name, only the changes between the two projects are made. To perform this operation, open a project with the [FILE] [OPEN PROJECT] menu command, followed by the [FILE] [SAVE AS PROJECT] menu command.

When saving a project under a new project file name, the source files associated with the project are not changed. If the new project file is in the same directory as the old project file, the same source files will be used by both projects. However, if the directories are different, when the new project file is saved, it copies all of the source files to the new directory as well. It is important to note that all the files related to a project must be in the same directory. The source files that required changes should be saved under a new file name, in order not to make changes to the original project's source file, when they are stored in the same directory.

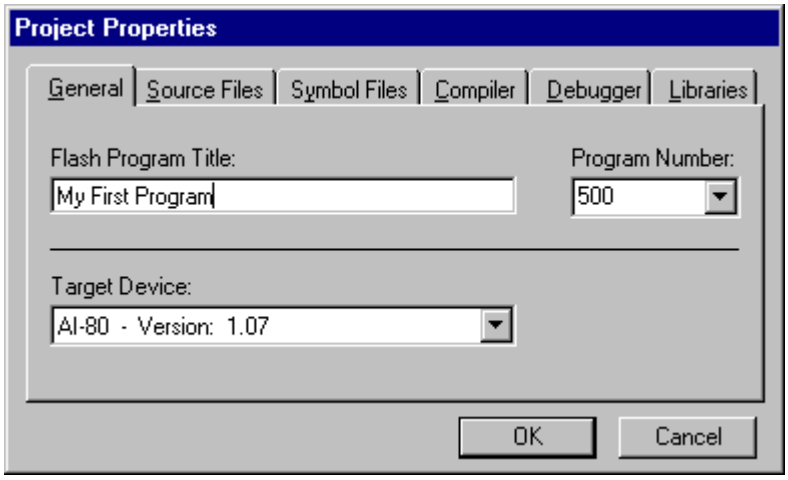


The last four projects opened are remembered by the A.I.WorkBench program, and upon restarting the program, it will immediately display a window similar to the figure above. To open any of the projects listed, simply select it and press the OPEN button, or double click the mouse on the desired project file.

Project Settings

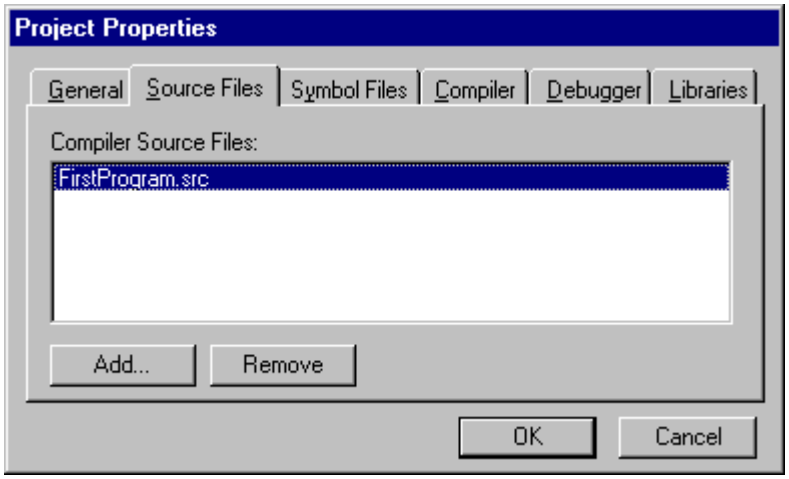
All of the project settings can be viewed and changed by selecting the [VIEW] [PROJECT SETTINGS] menu command. A window similar to the following figure will be displayed. It consists of five different 'tabs' containing various settings.

The first tab, termed General, shows the program title and number to use along with what device the compiler will generate code for. The program title is displayed in the Flash Memory Files window and can be up to 50 characters in length. The program number is used as the primary means for the AI-80 to identify and distinguish programs contained in the flash memory. As such, each program contained in the AI-80 must have a unique program number. The normal convention is for user program numbers to range from 10 to 99, and 500 to 599. The lower range of 10 to 99 is meant for default factory loaded programs, while 500 to 599 can be used for user developed programs. Any number in the range of 1 to 65535 is valid and can be used. However, program numbers below 10 should be reserved for system functions only.



The Target Device drop-down box indicates which AI-80 software versions are supported by the current compiler. The version selected should always match the software version of the AI-80, and never be higher. If the AI-80 software version is less than displayed in the drop-down box, it should be upgraded in order to take advantage of any new features. To upgrade the AI-80 software see section 4-6: Upgrading the AI-80 Software.

The following tab on the Project Properties window is labeled Source Files. Clicking it displays all the source files that will be compiled into a program file. New source files can be added by selecting the ADD button. To remove a source file, select it in the list and click the mouse on the REMOVE button.

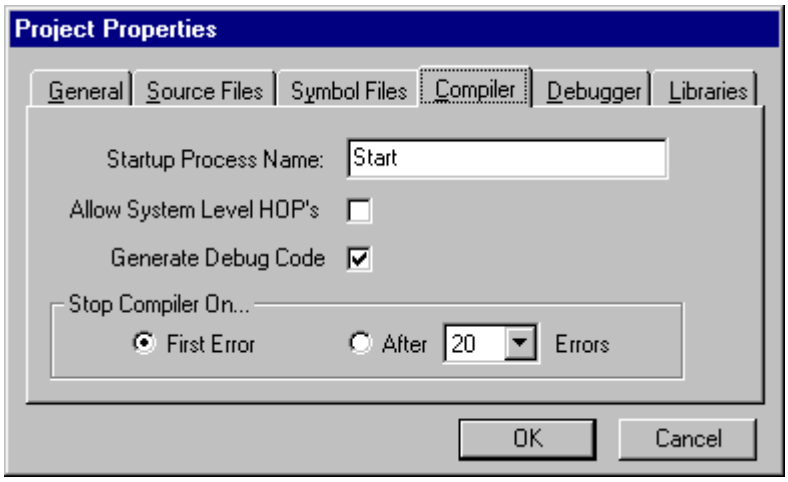


The Symbol Files tab is used to display any symbol tables from other projects that are included in this project. The purpose of using another project's symbol table is so that inter-process variables can be shared between different programs. This allows for the possibility to run multiple programs at the same time and share data between the programs.

Note: As of version 1.8 of the A.I.WorkBench program, the importation of other projects symbol tables is not supported. Though data can still be shared between programs by declaring IMPORT and EXPORT variables with identical register numbers.

The fourth tab, labeled Compiler, displays a number of compiler related options. The first item is the Startup Process Name. When a program consists of multiple processes, only one of them is initially started. The other processes within the program must be controlled by the first process. As such, the compiler must know which is the first process to be started. The name of that process is entered in the "Startup Process Name" text box, as shown below.

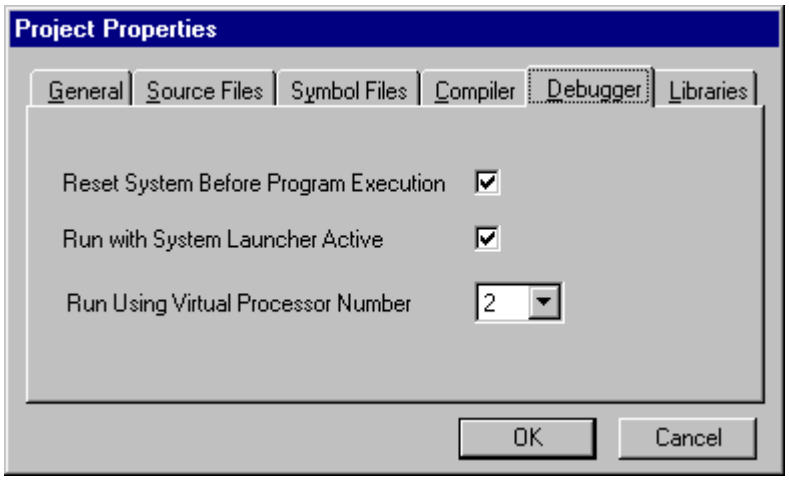
If no startup process is defined, or the startup process name does not exist, the compiler will then use the first process defined in the program listing as the startup process. For the majority of programs that only contain one process, it automatically becomes the startup process.



The two following check boxes allow the access to system level Hardware Object Properties (HOP's) and control the generation of debugging information respectively. Normally, access to the system level HOP's in only required by programs performing system level functions. Since improperly accessing these HOP's can cause unpredictable operation, access is normally disabled. The option to generate debugging code, while increasing code size, gives the ability to single step through a program for debugging purposes, along with setting breakpoints.

Note: As of version 1.8 of the A.I.WorkBench program, a built-in debugger capable of supporting breakpoints and single-step functions is not available. These features are planned for a future release.

The fifth tab, labeled Debugger, controls various options applicable when executing compiled programs. In testing a compiled program, it is normally downloaded to the AI-80's internal RAM and executed. The download time into RAM is generally much faster than saving the program into the flash memory storage. However in order to mimic the same conditions as the program would execute from flash, the following settings can be altered.



The first check box, if enabled, causes the AI-80 to execute the System Reset program before running any programs downloaded into its RAM. This ensures that every time the downloaded program is executed, all the AI-80 hardware settings (HOPs) are in their default state, and not the same as set by the last program executed. When executing programs from the flash memory via the front panel keys, the System Reset program is also used to maintain the default hardware settings before a program starts.

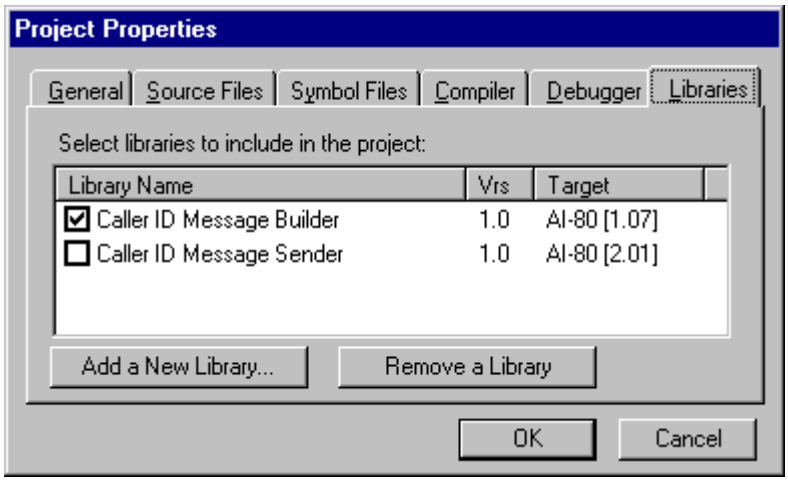
The second check box, if enabled, will make sure the System Launcher program is active before the downloaded program is started. The System Launcher program manages the execution of other programs and provides the user interface via the front panel keys. It allows other programs to be paused or stopped while in progress and resets all the hardware settings when a program finishes. Since in most applications, the System Launcher program will be active, the check box should be enabled. However, when developing system level programs or a custom user interface program, the standard System Launch program should not be executing at the same time. In these cases, the check box should be disabled.

The third setting controls which processor number the program will be started with. The AI-80 interpreter can execute up to four different processes. This can be broken into either multiple programs, each one using a process, or a single program using up to four simultaneous processes, or any combination in between as long as only four processes are executing at any one time.

Normally, programs will be started on process number two. Process one is used by the System Launcher program, if active. Processes three and four are free and can be utilized by the program. Only if writing system level programs should this setting be changed. If user programs do not use process number two, then the System Launcher will be unable to control the programs execution.

Project Libraries

The last tab in the Project Properties window is used to select which (if any) libraries are included with the project. Clicking the mouse on the tab shows a list of all the libraries that are currently available, as shown below. This includes the default libraries included with the A.I.WorkBench software and user created libraries.



The libraries are a collection of common subroutines and functions. They can simplify programming by providing routines that perform high level functions, freeing the user from developing common and repetitive low level functions. Two libraries are supplied with the A.I.WorkBench software. They are:

Caller ID Message Builder:

This library contains a collection of subroutines that simplify the creation of FSK Caller ID based message. Both single data and multiple data message formats can be created, including the visual message waiting variations. The subroutines provided can easily create complex multiple messages with various parameters, such as date & time, calling number, calling name, call qualifier, and visual indicator. The resulting data from the message is programmed in the AI-80's DATA object, which can then be used to generate the FSK modulated signal.

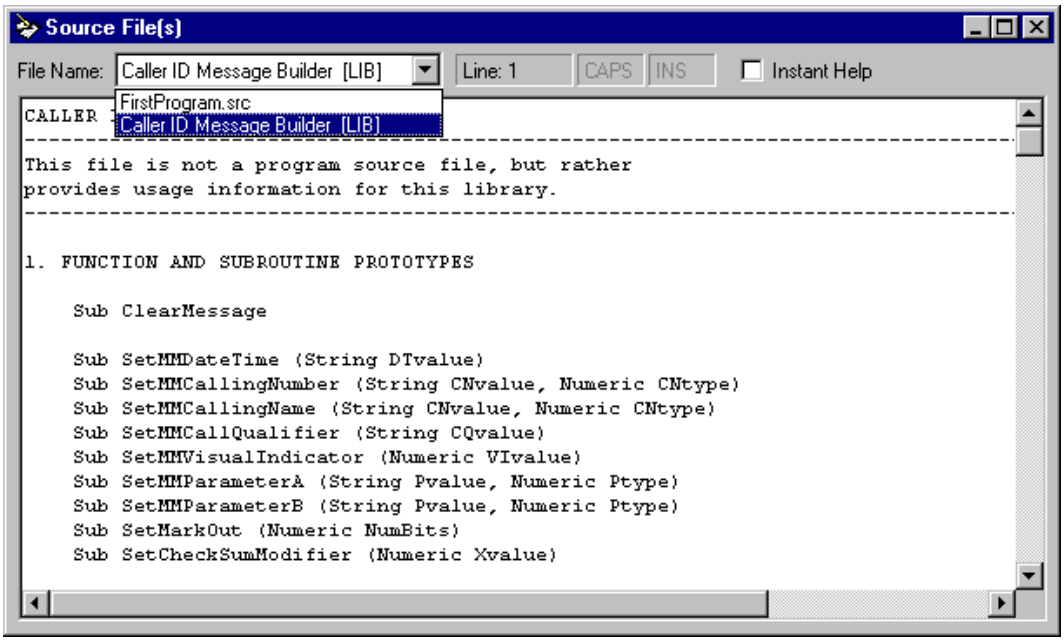
Caller ID Message Sender:

This library contains a collection of subroutines and functions that generate FSK Caller ID transmissions. Both type I (on-hook), and type II (off-hook) transmissions can be generated, with programmable levels and timing. Designed to be used with Caller ID Message Builder library, these routines can send the message created by that library.

To include a library with a project, click the mouse in the square box beside the library name. Any library with a check mark is included in the current project. In addition to the library name, its version and target information is included in the displayed list. The compiler will ensure that the target version of the library is compatible with the project. User created libraries can be added to the list by selecting the "Add a New Library" button. This opens a dialog window from which the library details and source file can be entered. Likewise, to remove a library from the list, select the library then click the mouse on the "Remove a Library" button.

Once a library is included in a project, usage information on that library is included in the source file editor. In the above Project Properties window, the Caller ID Message Builder library was checked and included in the project. Once the

above window is closed by pressing the OK button, the Source File window will include a new entry in its File Name list box. As shown in the following figure, a new entry called "Caller ID Message Builder [LIB]" is included in the drop-down list box. This is not the source file for the library, but rather a text file that explains the usage of the routines in the library. It describes each routine in detail and also provides examples.



Note: For information on how to create user libraries, see section 4-5.

Saving Projects

All the project settings and source files are automatically saved before the compiler is started. For new projects that are untitled, the user will be requested to supply a project file name and any source file project names. Projects can be saved at any time by selecting the [FILE] [SAVE PROJECT] menu command. This will save both the project file and all source files contained within the project. Selecting the [FILE] [SAVE AS PROJECT] menu command allows a new project file name to be used. If the new project file name is in a different directory as the previous file name, all the source files will be automatically copied to the new directory as well.

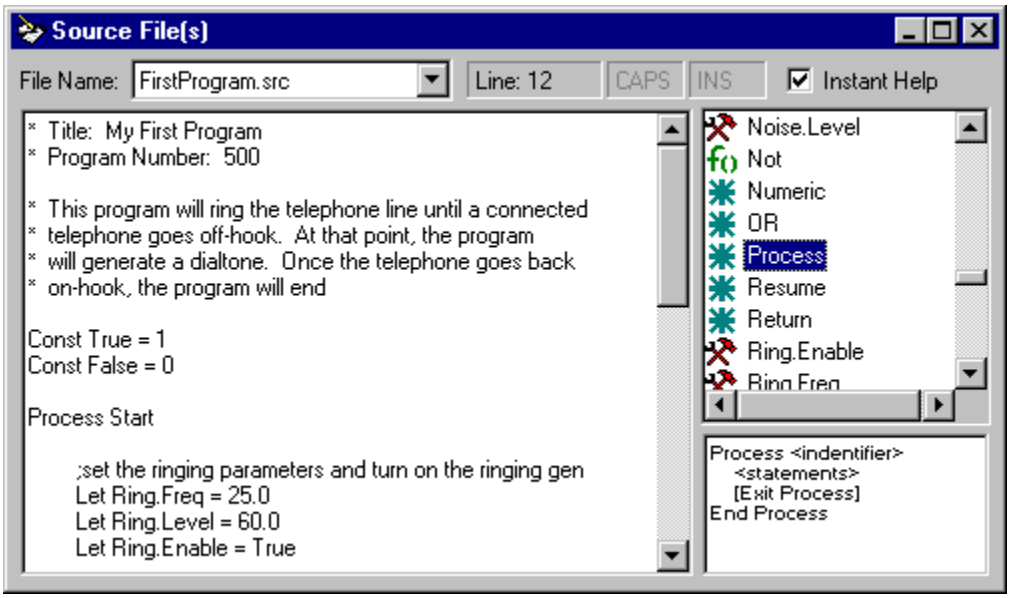
Section 3-5 Using the Source File Editor

The source files used with any project are plain ASCII text files. As such any editor can be used to create and change the files contents. The built-in source file editor includes some features designed to make developing AI-80 programs easier.

Working with Source Files

The source file editor is shown by selecting the [VIEW] [SOURCE FILE] menu command. If an existing project file is opened or a new project started, the source file editor is automatically displayed.

The following figure displays the FirstProgram source file (as developed in Section 3-2: Introduction to Programming. The window can be broken down into three different areas. The top section displays various status and control items. The left side displays the source file contents, and the right side displays optional help information.



The top status area shows the current file name being edited in the drop-down list box. If more than one source file is part of the project, the remaining source files can be selected from the list. Choosing a new source file will automatically save any changes made to the current file and then load and display the contents of the new file. If a project's source file is new, then the text " - new" is appended to the file name displayed. Once the file is saved, the appended text will be removed.

To change the current source file name, select the [FILE] [SAVE AS SOURCE FILE] menu command. A window will appear from which a new name can be chosen. Once the SAVE button is pressed, the file will be saved under the new name. A message will appear requesting if the project file should be updated with the new source file name. If yes is answered, then the project settings and the displayed file name in the drop-down list will reflect the new source file name. If no is answered, then the project settings and displayed file name remain as previous; however, a copy of the file has been saved under the new name.

The area to the right of the file name drop-down list box shows the current line number the cursor is at, and whether or not the caps lock is active and if the editing mode is overstrike or insertion.

Instant Help

The check box in the upper right portion of the window determines if the instant help information is displayed. The default setting is enabled. The help feature is composed of the top list box, which shows all the keywords and hardware properties (HOP's) that can be used within a program. Also included in the list are any variables, constants, subroutines, functions, and labels defined in the source files. Once the compiler has read all the project's source files, it creates a symbol file. This symbol file is read by the Instant Help feature and it updates the list with the various symbols defined in the program. The icon associated with each item in the list represents the type of symbol it is. The following list describes the meaning of each icon.

 Programming language keyword (reserved word)

 Functions, either built-in to the language or user defined

 Subroutines, either built-in to the language or user defined

 Hardware property (built-in variables used to control the AI-80's hardware settings)

 User defined variables (can be read from or writen to)

 User defined constants, or parameters in functions or subroutines (can only be read from)

 User defined labels, or program processes

The text area, just below the list of all keywords and symbols is used to provide information on the item selected in the list box. For programming keywords, this is syntax information and usage of the keyword. Subroutines and functions display usage information if they are built into the programming language. If user defined, the location and source file of their definition is shown. Hardware Object Properties (HOP's) display their data type, access type (if read only or write only), and any restrictions in use, including minimum and maximum values, if applicable. Variables and constants display their data type, their parent (who owns them), and their declaration location in the source file. Finally, labels show their parent and location in the source file.

As words are typed in the selected source file, the Instant Help feature attempts to matched the word with the closest item (alphabetically) in its list of symbols and keywords. It highlights this item and displays any relevant information in the text

area below the list. Pressing the TAB key will then complete the word in the source file with the selected item in the list.

For example, typing the letters “pr” will highlight the keyword “Process” in the Instant Help list box. By pressing the TAB key, the editor will complete the word and enter “Process” where “pr” had been. This helps to speed up writing programs, since once enough characters have been typed, the editor can finish the word by pressing the TAB key.

Alternatively, double clicking the mouse on the selected item in the Instant Help list box copies the item to source file window at the current insertion point. This allows simple program lines to be written by double clicking the mouse on the desired items. Clicking the right mouse button, when over the list box, inserts a new line into the source file.

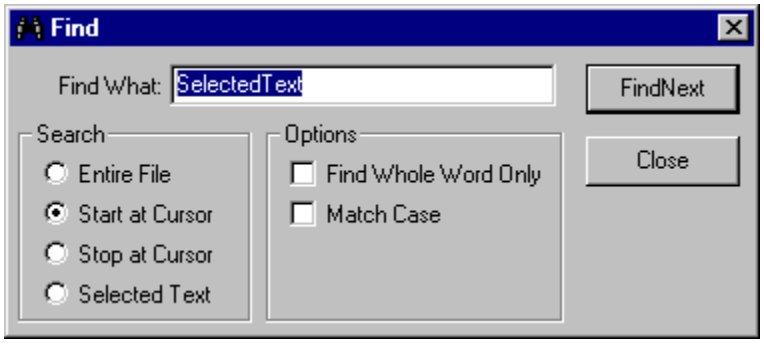
At the completion of the compilation process, the Instant Help feature will update the item list with any new symbols defined in the program. For very large programs, this can take a few seconds. If this becomes too time consuming, the Instant Help feature can be turned off by clicking the mouse on the check box in the upper right side of the window.

Editing Files

When editing the source files, basic clipboard functions can be used through either the [EDIT] menu or the keyboard shortcut keys. These include the standard Cut (Ctrl-X), Copy (Ctrl-C), and Paste (Ctrl-V) functions. Two additional operations available are Find and Replace text. These can be used to locate text phrases or words from within the source file.

To use the Find feature, select the [EDIT] [FIND] menu command, or press the Ctrl-F key. This displays a window similar to the following. The text to search for is entered in the top of the window. Various searching options can be selected from the two groups of control buttons. Pressing the FindNext button starts the search for the matching text.

If any text has been highlighted in the source file before the Find text window is displayed, then the selected text is automatically copied to the Find What field in the window.



The Replace feature is very similar to the Find feature, except that it will replace the located text with a different text string. Selecting the [EDIT] [REPLACE] menu

command displays a window similar to the above figure, from which the text replace function and be controlled.

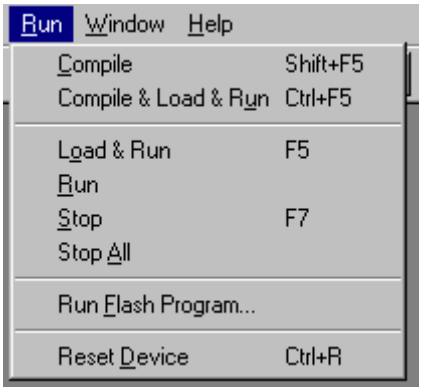
Section 3-6 Executing Programs

Once a project has been compiled without any errors, it can be downloaded into an AI-80 and executed. This requires that the AI-80 is connected to the host PC and has established a communications link.

The Run menu contains various commands to control the operation of AI-80 programs. Three basic steps are required to execute any program. They are:

- Compile the program without any errors
- Load the program in the AI-80 RAM
- Run the program

Though it is possible to load a compiled program into the AI-80's flash memory and then executed it from the flash memory, it is usually faster to load the program in the AI-80 internal RAM and execute from there. Once the program is finalized and debugged, it can be loaded into the flash memory.



The function of each of the commands in the Run menu is listed below.

Compile
The first command, Compile, generates the object code from the source files. If no errors were detected, the program can be loaded into the AI-80 and executed.

Compile & Load & Run
As an extension to the Compile command, this command also compiles the program, and if no errors were detect, it automatically loads the program into the AI-80's RAM and begins execution.

Load & Run
If the program has been previously compiled and no additional changes are required, the Load & Run command will transfer the program to the AI-80 and start its execution.

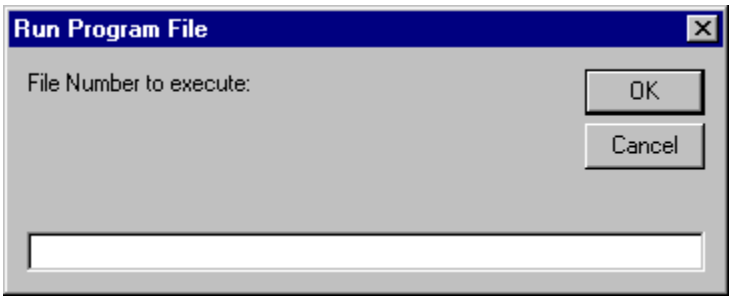
Run

Once a program has been loaded into the AI-80's RAM it will remain there unless certain flash memory files operations are performed. The Run command will execute any program that is contained in the AI-80's RAM.

Stop
The Stop command will terminate the program executing. However, this command assumes that the program is only operating from the startup process, as defined in the project settings. If the program starts other processes, those will not be stopped. To stop all processes running, use the Stop All command.

Stop All
This command terminates all processes active in the AI-80. Unlike the Stop command which terminates only the startup process of the program, this command will stop all of them. If the System Launcher was executing in one of the processes, it will also be terminated. As such the front panel keys may not function. As long as within the project settings the System Launcher option is enabled, it will automatically be restarted once the program is executed again with the Run command.

Run Flash Program
This command can be used to execute any program residing in the flash memory. Programs are referenced by their ID number, which is set as part of the project settings and can be view in the Flash Memory Files window. The command displays the following window, from which a program number can be entered.



Reset Device
The Reset command will restore all the AI-80's hardware settings to their default settings, along with executing the System Boot program.

Section 4 Reference Information

This section contains more detailed reference information on the programming language and software system of the AI-80.

Section Contents:

- Programming Language
- Hardware Abstraction Layer
- System Software Overview
- Auto-Config Command Files
- Creating User Libraries
- Updating the AI-80 Software

Section 4-1 Programming Language

Introduction

The AI-80 is entirely controlled by programs executing via a built-in command interpreter. These programs manage the user interface (keypad and display), along with controlling the hardware of the AI-80 during testing sequences. Since the interpretive language is very simplistic and difficult to program, the A.I. WorkBench software compiles high level language statements into the low level interpretive language used by the AI-80. All the program files stored in the AI-80's non-volatile flash memory are in the interpretive format.

Since even the system level programs are constructed using the A.I. WorkBench compiler, the entire user interface of the AI-80 can be altered and customized for a wide variety of applications. See section 4-3, System Software Overview for more information on the operation of the system software programs.

An important feature of the AI-80 interpreter is the ability to execute up to 4 processes independently. Each process can execute a different program, or a single program can use up to four different processes. Each process has its own local data space and operates completely independently from the other three. Using multiple processes can greatly simplify complex tasks, by breaking a program in logical operations that operate in exclusion of other operations.

Language Syntax

The basic unit of any program is a process. Programs are constructed from at least one process, and may use up to four. If a program uses less than the maximum number of processes, other independent programs can be executed using the unused processes.

Process Block

Each process runs completely independently from each other; however, a common data pool is available to be shared between the processes for inter-process communications. If a program contains only one process, it is started when the program is launched. When a program has multiple processes, then only the process tagged as the "startup" process will be started. The others must be started within the program. The "startup" process of a program is defined in the programs Project Settings window.

The syntax for a process is as follows:

PROCESS <identifier>
<statements>
[EXIT PROCESS]

END PROCESS

When the program reaches the END PROCESS or EXIT PROCESS command, that process is terminated. Process blocks can not be nested. If a program contains more than one process, the PROCESS statements must follow one after another.

Function and Subroutine Blocks

In addition to processes, functions and subroutines can be defined and called from within a process. Both the subroutines and functions can be passed data for processing. Subroutines do not return any data, while functions can return data of either the numeric of string data type. Subroutines are initiated by the CALL command while functions can 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

Any user defined functions or subroutines must be declared outside of any PROCESS block, and can not be nested (defined inside another). The identifier chosen for any 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. This parameter list indicates the data type for each parameter, along with an 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 are different, or the data types do not match, an error will be generated.

The syntax for calling a subroutine is as follows:

CALL <identifier> [(<identifier> [, <identifier>])]

While functions are called inside expressions as:

<identifier> [(<identifier> [, <identifier>])]

It is important to note a few restrictions when working with functions and parameters.

- When calling a function or subroutine, expressions can not be used inside a parameter list. Only variables and constants can be passed.

- All parameters are “read only” inside a function or subroutine. The called routine can not 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 in recursion, its variable contents will be identical to that of the calling instance, and changing its value will change 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, the WITH keyword is used after EXIT FUNCTION or END FUNCTION. Any expression following is computed and its results returned to the calling expression. The WITH keyword is optional, and if missing, the function will return ether zero or an empty string, depending if the function data type is numeric or string.

An END SUB or END FUNCTION statement must be included to mark the end of the subroutine. 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 and Constants

Variables can be defined under various conditions by using the LOCAL, GLOBAL, IMPORT, and EXPORT keywords.

The primary difference between the LOCAL/GLOBAL and IMPORT/EXPORT variables relates to the various processes that can be defined within a program. All LOCAL and GLOBAL variables are localized to a specific process. As such, each process has its own instance of a LOCAL or GLOBAL variable. To share data between processes, the IMPORT and EXPORT keywords must be used to define a variable. Export will allocated space for the variable in the inter-process data pool, and also make that variable available to other programs, by adding it to the program’s symbol table. A program’s symbol table may be added to another program and be used to access all the variables defined with the EXPORT keyword. The program that accesses an exported variable, must still define that variable with the IMPORT keyword. IMPORT and EXPORT variables can only be defined inside a PROCESS block, while LOCAL and GLOBAL variables can be declared inside or outside PROCESS, FUNCTION, or SUB blocks.

The use of LOCAL or GLOBAL to define a variable affects the scope of the variable. When GLOBAL is used, that variable is known, and can be accessed in any process, function, or subroutine within the project. Note that each process will have its own instance of a global variable. A LOCAL variable has a more restrictive scope. If defined inside a process, function, or subroutine that variable is only know inside the block that defined it. As such, two LOCAL variables can have the same name, provided they are declared and used inside different process, function, or subroutine blocks. If a LOCAL variable is defined outside a process, function, or subroutine block, then that variable’s scope is the current source file. The variable will be known, and can be used, anywhere within the source file that contains its definition.

Note: As of release 1.8 of the A.I. WorkBench program, symbol table importation is not supported. The EXPORT and IMPORT keywords can still be used to share variables between processes in a single program. However, the memory allocation is not performed automatically for exported variables. As such, a register location must be manually assigned between the range of 1 to 300. Note that string variables require 16 consecutive data register locations, where as numeric variables require only 1 data register.

The syntax for defining a variable of the four different types is as follows. The data type can be either NUMERIC or STRING, and each identifier used must be unique. The declaration of a variable does not have to come before that variable is used in the program. During the compiling process, an initial scan of the program will identify all variables defined.

LOCAL <type> <identifier>
GLOBAL <type> <identifier>
IMPORT <type> <identifier> < register >
EXPORT <type> <identifier> <register>

While variables can have three different scopes (within current block, within current source file, within current project), the scope of a constant is either within the current block, or within the current project. If a constant is defined inside a process, function, or subroutine block, the constant is only known within that block. Any reference to it in another process, function, or subroutine block will cause an error. If defined outside all program blocks, the constant is known throughout the project and can be used anywhere.

The syntax for defining constants is shown below. The literal may be either a numeric value or a string. If a string, it must be enclosed in quotation marks.

CONST <identifier> = <literal>

Note: All variables declared will be allocated storage space in the AI-80 internal registers regardless if they are used or not. Since the data space (per process) is restricted to 300 registers, excessive variable declarations can exhaust the supply very quickly. Each numeric variable requires a single register, while a string variable requires sixteen registers. Avoid declaring large numbers of variables inside subroutines or functions. Instead, if creating a collection of related routines (a source code module or library), use LOCAL variables with file scope that can be shared between the various functions and subroutines.

Program Statements

All the programs are constructed from fifteen basic programming statements. Many of these are similar in form to that of common high level programming languages. Unique statements are included for controlling the execution of the various processes. The basic program statements are:

- LET
- LOOP
- FOR-NEXT
- IF-THEN-ELSE
- SELECT-CASE
- CALL
- LABEL
- GOTO
- GOSUB
- RETURN
- LAUNCH
- STOP
- HALT
- RESUME
- ASM

LET:
The LET statement is used to compute the result of an expression and transfer the value to a variable. The expression itself may contain variables, constants, literals, or function calls. The data type of the left hand side variable must match that of the expression. As such, numeric variables can only be assigned numbers, and string variables can only be assigned strings. The syntax for an expression is defined in the following pages.

LET <variable> = <expression>

LOOP:
The LOOP statement can be used to repeat 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 will be repeated. If an expression is present, it must evaluate to a numeric value. If no expression is present, the statements will loop indefinitely. Optionally, the EXIT LOOP statement will immediately exit the loop.

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

Note: Various commands are not allowed inside a LOOP statement. They are GOTO, RETURN, EXIT SUB, and EXIT FUNCTION. In situations where these statements are required, substitute the FOR-NEXT statement for the LOOP statement.

FOR-NEXT:
The FOR-NEXT statement is an alternate method of repeating a block of statements, instead of using the LOOP command. The specified variable and expressions must be of a numeric data type. The variable is initially set to the value of expression1. The variable is then compared to the value of expression2. If less than or equal to, the following statements will be executed. At the NEXT statement, the variable is incremented by either the value of 1 or, if expression3 is

present, its value. If the variable value is still less or equal to expression2, then the statements between FOR and NEXT are repeated.

```
FOR  <variable> = <expression1> TO <expression2> [ STEP ...
                                                   <expression3> ]
        <statements>
        [EXIT FOR]
NEXT <variable>
```

If the EXIT FOR statement is encountered, the program will immediately jump to the statement following the NEXT command, and continue execution from that point.

IF-THEN-ELSE:
The IF statement can be used for conditional program control. The expression is evaluated, and if true, the following statements are executed. A true expression is one that evaluates to a numeric result, in which the result is not zero. An expression that evaluates to zero is considered false. Optionally, the ELSE statement can be used to define a block of statements that will be executed if the expression is false. In all cases, the END IF command is required to mark the end of the IF command.

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

SELECT-CASE:
The SELECT statement is an alternate way of conditional statement execution. The result of the expression1 is compared to the expressions after the CASE statements. If they are equal, the statements following the CASE keyword are executed. The expressions to be compared can be either numeric or string expressions, but must be consistent with each other (can not mix and match data types). The keyword ELSE can be used instead of an expression, in which, the following statements will always execute if none of the previous CASE expressions were equal.

```
SELECT <expression1>
        CASE <expression2> | ELSE
                <statements>
        ...

        [ CASE <expression>
                <statements> ]
END SELECT
```

Note: If CASE ELSE is used it should occur as the last CASE statement before the END SELECT command, since it will always evaluate as true.

GOTO & LABEL:

Unconditional program branching is performed with the GOTO command. Program execution jumps to the first statement following the specified label.

GOTO <identifier>
LABEL <identifier>

GOSUB & RETURN:
Simple subroutines (without parameter passing), are called using the GOSUB command. Program execution will jump to the specified label. To return to the next statement following the GOSUB command, use the RETURN command.

GOSUB <identifier>
RETURN

Subroutines using the GOSUB command may be nested. However, each subroutine jump requires one register on the process's stack in order to store the return address. As the stack size is limited, the number of nested subroutine calls is limited to a maximum of 40. If a RETURN command is encountered before any GOSUB command, a stack fault will be detected by the AI-80 program interpreter. This will cause an error and stop the current process.

CALL:
The CALL statement is used to execute a subroutine. Optional identifiers can be passed to the subroutine. The number and type of any identifiers must match that defined in the subroutine's SUB block.

CALL <identifier> [(<variable>|<literal> [, <variable>|<literal>]]

LAUNCH & STOP & HALT & RESUME:
The following statements are used to control the multiple processes that may be executing. The LAUNCH statement is used to start additional processes, either within the same program or within a new file (program). With the STOP, HALT, and RESUME statements, processes can be terminated, halted or resumed respectively. The keyword ME refers to the task number of currently executing process.

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

STOP <task number> | ME
HALT <task number> | ME
RESUME <task number>

The resume statement will only function if the specified process is currently in the halted state. If running or stopped, the resume statement will have no effect.

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

ASM:

The ASM statement allows the AI-80 interpreter to be programmed directly in its native language. The syntax is as follows:

ASM {native language statements}

Only under unusual conditions does a program require the ASM command. The most common use is when a process wishes to directly access its own, or another process's register file. Programs can use the ASM command to examine any process's stack registers, program counter register, status register, and local data space. The native language statements can also appear directly in expressions. The following example reads the program counter of the current process and transfer the result into a variable called PC.

LET PC = {VN1}

Built-in Subroutines and Functions

A number of subroutines and functions are included in the basic language syntax. Subroutines must be used with the CALL statement, while functions can be used as part of any expression. However, the data type returned by the function, must match the data type of the expression it is contained in.

Built-in subroutines:

WAIT (<expression>)
Expression must evaluate to a numeric value. The subroutine suspends the process for the specified number of milliseconds. All other processes will continue to execute.

Built-in functions:

VAL (<identifier>) Converts a string to a numeric value. The string must only contains the digits 0 to 9, or the decimal point, and may be preceded by the minus sign. If any other character is present in the string, it will be evaluated to zero.

STR (< identifier >) Converts a numeric to a string. The string output is formatted in scientific notation ([-]n[.n]e[-]n).

ISTR (< identifier >) Converts a numeric to an integer string. The string output is an integer in the range of -2^23 to 2^23-1.

INT (< identifier >) Returns the integer value of a numeric value. The passed value will be rounded to the nearest integer value.

ABS (< identifier >) Returns the absolute value of a numeric value.

LEN (< identifier >) Returns the number of characters in a string. The returned value will range from 0 to 64.

NOT (< identifier >) Returns the logical NOT of the numeric value. Non-zero values will return zero, while zero values will return 1.

NEG (< identifier >) Returns the negative of the numeric value.

LOG (< identifier >) Returns the logarithm of the numeric value. The base of the logarithm is 10.

EXP (< identifier >) Returns the result of the numeric value raised to the tenth power.

CHR (< identifier >) Converts an ASCII value into a string of one character length.

ASC (<ident1>, <ident2 >) Returns the ASCII character code of a single character in the *ident1* string at the position of *ident2*. For example, ASC("Hello",2) returns the ASCII code for the second character, which is "e" or 101.

MID (<ident1>, <ident2>) Returns a single character string with the character in the *ident1* string at the position of *ident2*. For example, ASC("Hello",5) returns the fifth character in "Hello", which is "o".

Note: The built-in functions can only evaluate a single variable, constant, or literal value. Expressions are not allowed inside the function parameter list.

Identifiers, Data Types, Variables, and other stuff

The restrictions and syntax of the identifiers, types, and variables used within the program statements, are defined here.

<type> Data Type

Data types can be one of two different kinds. They are either “NUMERIC” or “STRING”. Numeric data types are represented as a 32 bit single precision floating point value, while string data types can consist of a series of ASCII characters ranging from zero to a maximum of 64 characters. The data type is specified when defining variables, functions, and subroutines.

<identifier> Identifier

Identifiers are used to represent processes, functions, subroutine, variables, and labels. They consist of ASCII strings that can contain letters or numbers, but must start with a letter and cannot include any other characters except ‘_’. The maximum length of an identifier is 32 characters.

<literal> Literal

A literal is an ASCII string representing a value in a format compatible with the valid data types. For the numeric data type, it must be a number in either of the following formats:

decimal:	[-]n[.n][E[-]n]	i.e.	-12.34e5
hexadecimal:	0xr	i.e.	0x1f7c

Where n represents 1 or more digits between 0 and 9 and r represents hexadecimal characters 0 to 9, and A to F. For the string data types, literals must be enclosed in quotation marks as in the following example:

string: “hello world.”

The quotation marks are not part of the literal, but are needed to delimit it. If quotation marks are to be part of the literal, add two consecutive quotation marks to represent each quotation mark. For example, the literal:

“he said, “”never!””, then left the room.” represents the string:
he said, “never!”, then left the room.

<statements> Statements

Statements must begin with one of the following keywords:

LET, LOOP, FOR, IF, SELECT, GOTO, LABEL,
CALL, LAUNCH, STOP, HALT, RESUME, EXIT, and ASM

<variable> Variable

A variable is a type of identifier that has been defined with the LOCAL, GLOBAL, EXPORT, IMPORT keywords in the program. Variables are used to represent data in either a string or numeric representation.

Variables are also used to represent the hardware resources of the AI-80. These variables are included as part of the language and can be used anywhere a user defined variable is used. Referred to as HOPs (Hardware Object Properties), these variables are defined by the following structure:

<object name>.<property>

The HOP variables are mapped to hardware control registers within the AI-80 and control all the hardware functions. They can be of either NUMERIC or STRING data types. Some may have access restrictions, such as read only or write only. As such, writing to a read only HOP will cause an error, as will reading from a write only HOP. The next section (4-2) describes the usage and function of the HOPs.

<file ID number> Program File ID Number

The file ID number can be either a variable or literal of a numeric data type. It represents the file number of a process to be started with the LAUNCH command.

<task number> Task or Process Number
The task number must be a numeric literal. It represents the process number to be either started, stopped, halted or resumed. The valid range is from 1 to 4.

<expressions>
The structure of an expression is defined as follows. It must have at least one operand. Additional operands must be separated by operators. Depending on the data type of the variables, some operators are not supported.

<expression> == <operand> [<operator> <operand> ...]

<operands> == <variables> | <constants> | <literals> | <function>

where the operators are defined as:

< operator > == "AND" | "OR"
< operator > == "=" | "<>" | "<" | ">" | "=<" | ">="
< operator > == "+" | "-"
< operator > == "*" | "/"
< operator > == "bAND" | "bXOR"

For the string data type, only the following operators are valid:

< operator > == "=" | "<>" | "+"

Expressions are evaluated in a strict left to right fashion. Operator order of precedence is NOT followed.

The result of logical operators (AND, OR) and relational operators (=, <>, <, >, =<, and >=) is always a numeric value of either true (1), or false (0).

For string data types, only three operators can be used. The "+" operator concatenates the two string operands, while the "=" and "<>" operators will return a numeric true (1) or false (0).

The two operators bAND and bXOR perform a bit-wise AND and XOR operation. Accordingly the operands must be of the numeric type. Before the bit-wise operation is performed the operands are converted to integer values.

Note: Though, any non-zero value represents a true result, the logical and relational operators will always return the value of 1.

Program Limits and Language Restrictions

The program limits are a function of the command interpreter residing in the AI-80. The compiler will verify that these programming limits are not exceeded for any program.

Maximum code size:	65535 bytes
Variable data space:	300 registers per local process
	300 registers for inter-process data space
	1 register used for each numeric variable
	16 registers used for each string variable

Maximum # processes:	4 (at a given instant)
Stack depth per process	40 registers

Language restrictions are a result of both the compiler limitations and restrictions in the AI-80 interpreter. Future software versions of the AI-80 and compiler may remove these restrictions.

- User defined functions and subroutines are NOT re-entrant. The variables defined within a function or subroutine are allocated permanent space in the process's data space. Re-entrant calls will use the same data registers.

- Parameters passed to a user defined function or subroutine can be read by the statements in the function or subroutine. However, the parameters can NOT be written too. That is, they can not be modified by the function or subroutine.

- Unary operators are not supported in expressions. Function calls must be used instead.

- The maximum number of loops in the LOOP command is 2^{23} -1.

Note: The programming language is NOT case sensitive. As such, keywords or identifiers can be entered in either upper case, lower case, or a combination of both.

Section 4-2 Hardware Abstraction Layer

The hardware abstraction layer defines how programs can control and manage the hardware resources of the AI-80. While a program is running, it has full control over the AI-80 and can manipulate the various tone generators, telephone interface settings, display settings, along with monitoring key presses, timer values, and host communications.

Each hardware property is mapped to a built-in variable, that can be used in the same manner as any user defined variable, which is declared with the LOCAL, GLOBAL, EXPORT, and IMPORT statements. Writing a new value to the hardware property immediately changes the hardware settings. Likewise, reading from a hardware property returns its current status. The hardware properties are group into logical collections and can be referenced in a program (as a variable) in the following structure:

<object name>.<property name>

Referred to as HOPs (Hardware Object Properties), the variables are specified by the object name, followed by a period, then the property name. As with user defined variables, the HOP's can represent two different data types. These being numeric or string. Unlike user defined variables, some HOP's have access restrictions, in that they may be read only (like constants), or in some cases write only.

Since the HOP's are treated as variables within the AI-80 programming language, they can be used anywhere a user defined variable is used. The following example shows how to set the ring generator frequency and level, and how to monitor the hook switch status.

```
Const True = 1
Const False = 0

Let Ring.Freq = 25.0                ;set ringing freq to 25 Hz
Let Ring.Level = 60.0               ;set ringing level to 60 Vrms
Let Ring.Enable = True              ;turn on the ring generator

Loop
        If TelInt.HookStatus = True Then
                Let Ring.Enable = False
                Exit Loop
        End If
End Loop
```

The above program sets the ringing generator frequency and level to 25 Hz and 60 Vrms respectively. After enabling the ringing generator, it waits (indefinitely) for the hook switch status to detect an off-hook condition. At that time the program disables the ringing generator and ends the program.

The following tables describe in detail all the HOP's accessible by the AI-80 programming language. A number of properties represent binary controls (active

or inactive). The active state is always enabled by writing a non-zero value (true), while the inactive state is set by writing value of zero (false). Likewise, for reading the properties, a non-zero value represents active, while zero represents inactive.

RING Object

The ring generator is controlled with the RING object. When enabled, it will generate the specified level across the tip and ring leads of the active telephone interface port. Even though the ring generator will turn off if the port enters an off-hook state, the ENABLE property will continue to indicate active unless written to with the value of zero.

RING	.FREQ	type Numeric	Access R/W	Special
	Desc.	This property controls the frequency of the ring generator. Its value can be set to any value in the range of 10 Hz to 100 Hz.		
	.LEVEL	type Numeric	Access R/W	Special
	Desc.	Sets the level of the ringing generator between the limits of 0 to 80 Vrms.		
	.ENABLE	type Numeric	Access R/W	Special
	Desc.	Used to turn on or off the ringing generator. Setting to a non-zero value (true) will enable the generator, while a value of zero (false) will turn off the generator. The current status of the ringing generator is returned by reading the property. Enabling the ringing generator will turn off the tone A, tone B, and noise generators (if active).		

TONEA Object

The Tone A generator is a flexible signal source that can create either a single frequency tone, FSK modulated tone, or amplitude modulated tone. Its output will be present at the active telephone interface port in either the on-hook or off-hook states. The ring generator has priority over the tone generator. As such, turning on the ring generator will turn off the tone.

TONEA	.FREQ	type Numeric	Access R/W	Special
	Desc.	In single tone or AM mode of operation, this property controls the output frequency. In FSK mode, it controls the frequency of the space tone. Its valid range of values is from 20 Hz to 10,000 Hz.		
	.LEVEL	type Numeric	Access R/W	Special
	Desc.	In single tone or AM mode of operation, this property controls the output level. In FSK mode, it controls the level of the space tone. Its valid range of values is from 0 Vrms to 4.0 Vrms.		
	.PHASE	type Numeric	Access R/W	Special
	Desc.	The instantaneous phase angle of the tone generator can be set or read with this property. The property represents the phase angle in units of degrees between 0 and 360.		
	.ENABLE	type Numeric	Access R/W	Special
	Desc.	The tone generator is enabled by writing a true value. Likewise it is turned off by write a false value. If using ToneA as an FSK modulator, the FSK properties must be set along with the modulation mode before writing a true value, otherwise the FSK modulator will not be enabled.		

.FREQMARK	type Numeric	Access R/W	Special
Desc.	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 20.0 Hz to 10,000 Hz.		

.LEVELMARK	type Numeric	Access R/W	Special
Desc.	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 Vrms to 4.0 Vrms.		

.BITTIMESPACE	type Numeric	Access R/W	Special
Desc.	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 seconds to 1.0 seconds..		

.BITTIMEMARK	type Numeric	Access R/W	Special
Desc.	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 seconds to 1.0 seconds..		

.FSKBITINDEX	type Numeric	Access R/W	Special
Desc.	This property can be used to set or read the current pointer value for the FSK data array. The FSK data sent is stored in a buffer created with the DATA object. This property acts as the index pointer to the buffer. While the FSK modulator is active, this property will increment from its starting value to the maximum number of bits in the buffer. Writing to this property will change the index pointer and effect which data bits are being generated.		

.FSKNUMBITS	type Numeric	Access Read	Special
Desc.	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 object. This property is read only.		

.FSKCONTINUOUS	type Numeric	Access R/W	Special
Desc.	Writing a true value to this property before the FSK modulator is enabled, will cause the modulator to indefinitely repeat the bit pattern contained in the buffer. Writing a false value, disables this option		

.FSKHOLDCARRIER	type Numeric	Access R/W	Special
Desc.	Writing a true value to this property will cause the tone generator to maintain the frequency and level of the last FSK bit generated indefinitely.		

.MODULATION	type Numeric	Access R/W	Special
Desc.	The modulation mode of the tone generator is set by this property. The valid modes are: 0) Single Tone 1) FSK Modulation 2) Amplitude Modulation (Tone B is the modulating source) If in the AM mode, tone generator B will not produce any output, except as the modulating source for tone A.		

.FSKACTIVE	type Numeric	Access Read	Special
Desc.	The current state of the FSK modulator is returned in this property as either true (on) or false (off). If the FSKCONTINUOUS property is active, FSKACTIVE will return true until the FSK modulator is turned off with the ENABLE property. If the FSKHOLDCARRIER property is active, FSKACTIVE will return true until the last bit has been sent. After the last bit, this property will return false. However the mark or space tone will still be on until turned off with the ENABLE property.		

	.AMDEPTH	type Numeric	Access R/W	Special
	Desc.	This property controls the amplitude modulation depth between the limits of 0 to 100 percent. If the modulation mode is set to AM (2), the ToneB generator is used as the modulating frequency, while ToneA is the carrier frequency. The level of ToneB is ignored, as the modulation depth is controlled by the AMDEPTH property.		

TONEB Object

Less flexible than tone generator A, ToneB can be used to create a single frequency tone or act as the amplitude modulating source for ToneA, when in the AM mode of operation. As with ToneA, ToneB's output will be present at the active telephone interface port in either the on-hook or off-hook states. The ring generator has priority over the tone generator. As such, turning on the ring generator will turn off the tone.

TONEB	**.FREQ**	type Numeric	Access R/W	Special
	Desc.	This property controls the output frequency. Its valid range of values is from 20 Hz to 10,000 Hz.		
	.LEVEL	type Numeric	Access R/W	Special
	Desc.	This property controls the output level. Its valid range of values is from 0 Vrms to 4.0 Vrms		
	.PHASE	type Numeric	Access R/W	Special
	Desc.	The instantaneous phase angle of the tone generator can be set or read with this property. The property represents the phase angle in units of degrees between 0 and 360.		
	.ENABLE	type Numeric	Access R/W	Special
	Desc.	The tone generator is enabled by writing a true value. Likewise it is turned off by write a false value. Note that if ToneA's modulation has been set to AM, then ToneB will not produce any output, except to act as the modulation source for ToneA.		

NOISE Object

A broad band white noise generator is controlled with the following two properties. The noise spectrum is flat over the bandwidth of 20 Hz to 10 kHz with the output level representing the total noise voltage generated over the entire output bandwidth. If enabled, the noise output will be present at the active telephone interface port in either the on-hook or off-hook states. The ring generator has priority over the noise generator. As such, turning on the ring generator will turn off the noise.

NOISE	**.LEVEL**	type Numeric	Access R/W	Special
	Desc.	This property controls the output level. Its valid range of values is from 0 Vrms to 2.0 Vrms		
	.ENABLE	type Numeric	Access R/W	Special
	Desc.	The noise generator is enabled by writing a true value, and disabled by writing a false value.		

DATA Object

The data properties are used to manipulate the FSK data bit buffer, which is used for sending FSK modulated data. In the common application of sending Caller ID information to a telephone, the following properties are used to compose a message, which is sent with the FSK modulator built into tone generator A.

DATA

.CLEAR	type Numeric	Access Write	Special
Desc.	Writing any value to the CLEAR property will clear the data bit buffer. The number of bits in the buffer can be read at any time with the TONEA.FSKNUMBITS property.		

.PARITY	type Numeric	Access R/W	Special
Desc.	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 effects data added to the buffer with the ADDCHAR and ADDSTRING properties.		

.STOPBITS	type Numeric	Access R/W	Special
Desc.	When bytes, characters, or 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 100.		

.ADDMARK	type Numeric	Access Write	Special
Desc.	Writing to this property adds the specified number of mark bits to the FSK data bit buffer. The valid range is from 0 to 4096.		

.ADDSPACE	type Numeric	Access Write	Special
Desc.	Writing to this property adds the specified number of space bits to the FSK data bit buffer. The valid range is from 0 to 4096.		

.ADDALTERNATE	type Numeric	Access Write	Special
Desc.	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. The valid range is from 0 to 4096.		

.ADDBYTE	type Numeric	Access Write	Special
Desc.	Writing to this property adds a serially encoded 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.		

.ADDCHAR	type Numeric	Access Write	Special
Desc.	Similar to ADDBYTE, writing to this property adds a serially encoded 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 will effect the encoded bit pattern.		

.ADDSTRING	type String	Access Write	Special
Desc.	Similar to ADDCHAR, writing to this property adds a string of characters to the FSK data buffer. Each character is encode with a startbit, 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 effects the encoded bit pattern.			
	.ADDXSUM	type Numeric	Access Write	Special
	Desc.	Writing any value to this property will add the current checksum value to the FSK data buffer. The number of bits and format of the checksum depends on the XSUMTYPE value.		
	.XSUMENABLE	type Numeric	Access R/W	Special
	Desc.	This property enables or disables the checksum calculations. As part of many Caller ID standards, a checksum value is included in the Caller ID message. If enabled, the current checksum value is updated anytime the ADDBYTE, ADDCHAR, or ADDSTRING properties are written too. The checksum is enabled by writing a true value. Likewise, it is disabled by writing a false value.		
	.XSUMTYPE	type Numeric	Access R/W	Special
	Desc.	This property controls the checksum method used to update the checksum value. There are two settings: 0) Inverted Modulus 256 Conforms to Bellcore and ETSI FSK Caller ID standards 1) 16 Bit CRC (x16+x12+x5+1) Conforms to NTT (Japan) FSK Caller ID standards		
	.XSUMVALUE	type Numeric	Access R/W	Special
	Desc.	The current value of the checksum counter can be read or modified by accessing this property. Depending on the XSUMTYPE settings, this value will range from either 0 to 255 (8 bit), or 0 to 65535 (16 bit). Before creating a Caller ID message, the XSUMVALUE should be set to zero.		

MFGEN Object

The Multi-Frequency Generator (MFGEN) object implements a flexible and easy to use DTMF tone generator. In addition to creating DTMF tones, it can be modified to generate a wide variety of dual tone signals. Key to the generator is an internal data table containing frequency, level, and duration information for up to 20 arbitrary symbols. Each symbol can represent a single or dual tone signal with independent frequency, level, and duration. Once the symbol table has been set, tones can be generated one at a time, or as a string of tones. Before enabling the MF generator, both ToneA and ToneB must be disabled, as the MF generator takes a lower priority.

MFGEN	**.INDEX**	type Numeric	Access R/W	Special
	Desc.	This property is used to access the symbol data table. The data table consists of 100 entries (5 per symbol with 20 symbols). Each symbol has two entries for frequencies (in Hz), two for levels (in Vrms), and one for the tone duration (in msec). To read or write to any entry, set the INDEX property to the entry number, then use the VALUE property. See the following 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..		
	.VALUE	type Numeric	Access R/W	Special
	Desc.	The VALUE property works in conjunction with the INDEX property in reading and writing 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 changed by reading or writing to the VALUE property.		

.LEVEL	type Numeric	Access R/W	Special
Desc.	Writing to this property will set both tone levels, for symbols 1 to 16, to the specified value. This acts as a shortcut to manually setting the symbol table levels to a common value. The valid range for the level is from 0 to 4.0 Vrms		

.FREQADJUST	type Numeric	Access R/W	Special
Desc.	The FREQADJUST property is a shortcut to setting the frequency entries in the symbol table, when using standard DTMF tones. Writing to the property will set the frequencies for symbols 1 to 16 to the standard DTMF frequency, plus or minus the specified adjustment in percent. The frequency adjustment range is from -20 % to +20%.		

.ONTIME	type Numeric	Access R/W	Special
Desc.	Writing to this property will set 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 duration's to a common value.		

.OFFTIME	type Numeric	Access R/W	Special
Desc.	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 for this time interval is in milliseconds.		

.SYMBOL	type Numeric	Access Write	Special
Desc.	Writing to this property, specifies the next symbol number to be generated. The valid range for this property is from 1 to 20. To start the tone, write a true value to the ACTIVE property.		

.STRING	type String	Access R/W	Special
Desc.	The STRING property is used to generate a series of tone symbols. The 20 different symbols correspond to a different ASCII character (see table below). The time interval between each symbol is set by the OFFTIME property.		

.ACTIVE	type Numeric	Access R/W	Special
Desc.	This property is used to enable and disable the tone generator. Writing a true value will start the generator with either the last SYMBOL or last STRING value written to it. Once the generator has finished, reading the ACTIVE property will return false. While it is active, it will return true. If either ToneA or ToneB is enabled, the MF generator can not be started and will always return false.		

Symbol #	Freq 1	Freq 2	Level 1	Level 2	On-Time
1	1	2	3	4	5
2	6	7	8	9	10
--	--	--	--	--	--
--	--	--	--	--	--
20	96	97	98	99	100

Symbol Data Table Index Values

Symbol #	ASCII		Symbol #	ASCII
1	1		11	*
2	2		12	#
3	3		13	A
4	4		14	B
5	5		15	C
6	6		16	D
7	7		17	E
8	8		18	F

9 10	9 0		19 20	G H

Relationship between Symbol Number and ASCII String Characters

TELINT Object

The TELINT properties control the AI-80's telephone interface circuitry, such as port selection, line polarity, and line impedance. The properties can also return status information concerning the telephone hook switch status, and line balance condition.

TELINT

	type	Access	Special
.PORTB	Numeric	R/W	
Desc. The active telephone interface port and CPE interface port is set to B when writing a true value to the PORTB property. Writing a false value sets the active port to A (default).			
.REVERSE	Numeric	R/W	
Desc. Writing a true value to the REVERSE property reverse's the tip and ring leads within the active telephone interface port. Likewise, writing a false value, returns the polarity to its normal state.			
.OSI	Numeric	R/W	
Desc. The OSI property is used to disconnect the active telephone interface port from the front panel RJ-11 connector, by writing a true to it. Setting the OSI property to a false value re-connects the RJ-11 connector and restores the DC and AC line conditions.			
.CURRENT	Numeric	R/W	
Desc. The telephone interface delivers either a high (45 mA) or low (26 mA) loop current to an off-hook telephone, depending on the CURRENT property. Writing a true value sets it to the high current mode, while a false value uses the low current mode.			
.LINEIMP	Numeric	R/W	Option
Desc. The LINEIMP property controls the AC telephone interface source line impedance according to the following table. 0) 600 Ohms 1) 900 Ohms 2) Complex Impedance (optional)			
.HOOKDETECT	Numeric	Read	
Desc. The HOOKDETECT property is read only, can not be written to. It returns the status of the active telephone interface hook switch detect circuitry. If the port is off-hook, then HOOKDETECT returns true. If in the on-hook state, then HOOKDETECT returns false.			
.BALANCE	Numeric	Read	
Desc. The BALANCE property is read only, and will return true if the telephone interface becomes unbalanced (unequal DC loop currents). Under normal operating conditions, this condition should not occur and usually represents a hardware fault.			
.LENGTH	Numeric	R/W	Option
Desc. If an optional hardware line length module is installed, the LENGTH property is used to select 1 of 8 different line lengths. Writing a value between 1 and 8 selects the different line lengths. If this options is not installed, then reading the LENGTH property will always return zero.			

	.MEASPOINT	type Numeric	Access R/W	Special Option
	Desc.	As with the LENGTH property, the MEASPOINT property is used with the optional hardware line length module. If installed, it selects the signal measuring point as follows: 0) Measure signals before the line lengths (default) 1) Measure signals after the line lengths (at the RJ-11 connector) If not installed, reading the MEASPOINT property always returns zero.		

CPE Object

The CPE object only contains one property (HOOKSWITCH), which is used to set the CPE load to either the on-hook or off-hook state.

CPE	.HOOKSWITCH	type Numeric	Access R/W	Special
	Desc.	If writing a true value to the HOOKSWITCH property, the CPE load will be set to the off-hook state. Likewise, writing a false value sets the CPE load to the on-hook state.		

MEASURE Object

The MEASURE properties are used to measure the broad band voltage levels at either the active telephone interface, or the active CPE load.

MEASURE	.SOURCE	type Numeric	Access R/W	Special
	Desc.	The SOURCE property controls the measurement point for the level meter. It can be set to one of three values. 0) Measure signals at the telephone interface 1) Measure signals at the CPE load port (normal) 2) Measure signals at the CPE load port (ringing) For settings 0 and 1, the maximum signal level that can be measured is 4 Vrms (before compression); however, when using setting 2, levels of up to 100 Vrms can be measured at the CPE load port.		
	.SMOOTHING	type Numeric	Access R/W	Special
	Desc.	The SMOOTHING property controls the level meter's settling speed and level accuracy. If measuring low frequency signals, such as ringing, a smoothing factor closer to unity will improve measurement accuracy. For measuring voice band frequencies, the smoothing factor can be lowered, resulting in faster settling times. The following values can be used as a guideline. 0.9998 for measuring signals down to 10 Hz (+/- 0.1 dB) 0.998 for measuring signals down to 100 Hz (+/- 0.1 dB) 0.985 for measuring signals down to 500 Hz (+/- 0.1 dB)		
	.LEVEL	type Numeric	Access Read	Special
	Desc.	The LEVEL property is read only, and returns the current AC voltage reading of the level meter (in units of Vrms).		

DTMF Object

The DTMF object controls a DTMF tone detector that can be used to detect and measure any one of the 16 standard DTMF digits.

DTMF

.ENABLE	type Numeric	Access R/W	Special
Desc.	Setting the ENABLE property to a true value turns on the detector and starts the frequency and level measurements. If no DTMF digits are expected, the detector can be turned off by writing a false value to ENABLE.		

.SOURCE	type Numeric	Access R/W	Special
Desc.	The SOURCE property selects the signal source for the DTMF detector according to the following table: 0) Measure from the active telephone interface port 1) Measure from the active CPE load port		

.DIGIT	type Numeric	Access Read	Special
Desc.	The DIGIT property is read only and returns the value of any DTMF digit detected. If zero is read, no digit has been detected. Non-zero values correspond to standard DTMF digits as follows: 1-9) DTMF digits 1 to 9 respectively 10) DTMF digit 0 11) DTMF digit * 12) DTMF digit # 13-16) DTMF digits A to D respectively The DIGIT property is not “debounced” and during DTMF start up and shutdown the DIGIT property may give spurious readings. As such, any program using the DIGIT value should perform some basic debouncing to ensure the DTMF digit is valid.		

.FREQTOL	type Numeric	Access R/W	Special
Desc.	This property sets the DTMF acceptance tolerance for frequency error. The valid range of values is between 0 and 2 %. If the frequency counters detects a DTMF digit within this range, and the MINLEVEL property value is below both the low and high group tone levels, the DIGIT property will be set to the corresponding DTMF digit detected.		

.FREQTIME	type Numeric	Access R/W	Special
Desc.	The FREQTIME property controls the integration time of the frequency counters for the high and low group tones. This value can range between 2 and 20 milliseconds. The default time is 5 milliseconds. The longer the integration time, the more stable the frequency measurement will be. Since many telephone DTMF tones are produced digitally, the tones can have large amounts of short term instability along with high distortion levels. When using short integration times, this results in readings that vary between successive measurements. Longer integration times minimize the instability and distortion effects and results in more stable readings.		

.MINLEVEL	type Numeric	Access R/W	Special
Desc.	This property sets the minimum low and high group tone level limit. Unless a DTMF digit exceeds the minimum level the DIGIT property will always return a zero value (no digit detected). The minimum level is specified in Vrms.		

.LOWFREQ	type Numeric	Access Read	Special
Desc.	The LOWFREQ property is read only and returns the instantaneous frequency reading (in Hz) of the low group tone.		

	.LOWLEVEL	type Numeric	Access Read	Special
	Desc.	The LOWLEVEL property is read only and returns the instantaneous level reading (in Vrms) of the low group tone.		
	.HIGHFREQ	type Numeric	Access Read	Special
	Desc.	The HIGHFREQ property is read only and returns the instantaneous frequency reading (in Hz) of the high group tone.		
	.HIGHLEVEL	type Numeric	Access Read	Special
	Desc.	The HIGHLEVEL property is read only and returns the instantaneous level reading (in Vrms) of the high group tone.		

FSK Object

The FSK properties allows access to the optional AI-80 FSK decoding routines. These routines can decode up to 700 bytes of information encoded in a bit serial fashion. Note that the FSK decoder can not be active at the same time the DTMF decoder is active. If both are set to active the DTMF decoder will take precedence.

As the FSK decoder is an optional component to the AI-80, it can only be accessed if a software key is entered using the A.I.WorkBench software (vrs 1.6 and above).

FSK	**.ACTIVE**	type Numeric	Access R/W	Special
	Desc.	Setting this property to a non-zero value enables the FSK decoding routines. Likewise, writing a value of zero turns off the decoding routines. Note that the DTMF decoding routines take precedence over the FSK decoding routines. As such if DTMF.ENABLE is non-zero, then the FSK routines will be disabled regardless of the value set to FSK.ACTIVE. Also, if the required software key has not been programmed into the AI-80, reading the ACTIVE property always returns zero.		
	.SOURCE	type Numeric	Access R/W	Special
	Desc.	The SOURCE property is used to specify the signal pickup point for the FSK decoder. If zero, then the FSK decoder uses the signals present at the telephone interface as its input source. Otherwise, the signals present at the CPE interface are used as the input source.		
	.COUNT	type Numeric	Access R/W	Special
	Desc.	The COUNT property serves two purposes. First it returns the number of bytes received by the FSK decoder. Second, as up to 700 bytes are buffered by the FSK decoder, the COUNT property represents the index to the buffer of the last byte received. Once the 700 byte buffer has been filled, any additional bytes received are lost. Writing a value to the COUNT property sets the buffer index value for the FSK decoder. Normally, this value should be set to zero before enabling the FSK decoder.		
	.LASTBYTE	type Numeric	Access R/W	Special
	Desc.	This property represents the value (0 to 255) of the last byte received by the FSK decoder.		
	.MARKTIME	type Numeric	Access R/W	Special
	Desc.	The MARKTIME property returns the amount of time that a continuous mark signal is present. Once either a space signal is detected, or the signal level drops below the detection threshold, this property is set to zero. The value		

	returned is in units of seconds from 0 to a maximum of 100.			
	.INDEX	type Numeric	Access R/W	Special
	Desc.	The INDEX property is used to access the buffer that stores up to 700 bytes received by the FSK decoder. To access a byte, set the INDEX property to a value between 1 and 700. Then use the BYTEVALUE, BYTETIME, or BYTESTATUS properties to read the information stored in the buffer.		
	.BYTEVALUE	type Numeric	Access Read	Special
	Desc.	This property is read only, and returns the value of the byte stored in the FSK decoder buffer pointed to by the INDEX property. The value returned will be between 0 and 255.		
	.BYTETIME	type Numeric	Access Read	Special
	Desc.	When the FSK decoder receives a byte, it stores the byte value and the value of TIMER.FAST to a buffer. This property is used to return the timer value for the byte stored in the buffer pointed to by the INDEX property. The value returned will be between 0 and 100 seconds.		
	.BYTESTATUS	type Numeric	Access Read	Special
	Desc.	The BYTESTATUS property returns the status flags stored with each byte by the FSK decoder. The status flags are defined as follows: bit 0: Received Stopbit Value (1=mark, 0=space) bits 1 to 7: (Reserved, will read as zero)		

TIMER Object

The TIMER properties are used for generic time keeping functions along with detecting transitions in the telephone interface hook switch detector.

TIMER	**.SLOW**	type Numeric	Access R/W	Special
	Desc.	The SLOW property returns a value in seconds that automatically updates itself at a rate of once per millisecond. Its value will increment from zero to a reading of 10000 seconds, at which point it saturates. It can be written to any value between 0 and 10000, at which point it will start to increment from the new setting.		
	.FAST	type Numeric	Access R/W	Special
	Desc.	The FAST property is similar in function to the SLOW timer, except that the FAST timer updates itself once every 25.6 microseconds. The quicker update rate gives better timing resolution than the SLOW timer. The maximum value for the FAST timer is 100 seconds.		
	.ONHOOK	type Numeric	Access R/W	Special
	Desc.	The ONHOOK property will be automatically updated with the SLOW timer value whenever the active port's telephone interface hook switch detector records an off-hook to on-hook transition.		
	.OFFHOOK	type Numeric	Access R/W	Special
	Desc.	The OFFHOOK property will be automatically updated with the SLOW timer value whenever the active port's telephone interface hook switch detector records an on-hook to off-hook transition.		

DISPLAY Object

The DISPLAY properties control the AI-80 front panel display indicators. All of the properties are write only (can not be read) and allow programs to change the 4 digit 7 segment display and the individual LED's above the start, pause, and stop keys.

DISPLAY				
	.NUM	type Numeric	Access Write	Special
	Desc.	Writing to the NUM property displays the specified number on the 7 segment display. The range of numbers that can be displayed depends on the decimal point setting (DP property), as follows: DP=0 (display integers) Range = -999 to 9999 DP=1 (display tenths) Range = -99.9 to 999.9 DP=2 (display hundredths) Range = -9.99 to 99.99 DP=3 (display thousandths) Range = -0.999 to 9.999 Writing any values outside the valid range cause the display to show four consecutive dashes "- - - -", which represent an out of range condition.		
	.DP	type Numeric	Access Write	Special
	Desc.	The DP property sets the decimal point position to one of four possible settings. The range of values that can be displayed depend on the DP value, as shown above.		
	.LEDON	type Numeric	Access Write	Special
	Desc.	The LEDON property can is used to turn on the LED associated with either the start(1) , pause(2) , or stop(3) key. The value written to this property must be between 1 and 3.		
	.LEDOFF	type Numeric	Access Write	Special
	Desc.	The LEDOFF property can is used to turn off the LED associated with either the start(1) , pause(2) , or stop(3) key. The value written to this property must be between 1 and 3.		
	.BLINK	type Numeric	Access Write	Special
	Desc.	The 7 segment display can be flashed at various rates by using the BLINK property. The property value specifies the blink rate (in units of milliseconds) for the display. To turn off the flashing, a value of zero is written to the BLINK property. The maximum flash interval is 5000 milliseconds.		
	.SEGMENTA	type Numeric	Access Write	Special
	Desc.	The SEGMENTA property can be used to turn on and off the LED display segments that make up the right most digit. Each of the 7 segments, plus decimal point, is assigned a bit in a 8 bit (0 to 255) value. If the bit is set to 1, the corresponding LED segment is turned on.		
	. SEGMENTB	type Numeric	Access Write	Special
	Desc.	Similar to the SEGMENTA property, this value controls the second right most digit.		
	. SEGMENTC	type Numeric	Access Write	Special
	Desc.	Similar to the SEGMENTA property, this value controls the second left most digit.		
	. SEGMENTD	type Numeric	Access Write	Special
	Desc.	Similar to the SEGMENTA property, this value controls the left most digit.		

KEY Object

The KEY properties return the state of the AI-80 front panel keys. They are read only and can not be written to.

KEY				
	.START	type Numeric	Access Read	Special
	Desc.	The START property returns the number of seconds the front panel Start key has been held down. Once the key is pressed, the START property will being incrementing from zero to a maximum of 100 seconds. If the key is released, the START property immediately returns to a value of zero. The time resolution of the value is approximately 20 milliseconds.		
	.PAUSE	type Numeric	Access Read	Special
	Desc.	Similar in function as the START property, but returns the amount of time the Pause key has been pressed.		
	.STOP	type Numeric	Access Read	Special
	Desc.	Similar in function as the START property, but returns the amount of time the Stop key has been pressed.		
	.UP	type Numeric	Access Read	Special
	Desc.	Similar in function as the START property, but returns the amount of time the Up key has been pressed.		
	.DOWN	type Numeric	Access Read	Special
	Desc.	Similar in function as the START property, but returns the amount of time the Down key has been pressed.		

SPEAKER Object

The SPEAKER properties control the built-in speaker within the AI-80. This can be useful for monitoring the tones being generated by the AI-80 or from any CPE connected to the AI-80.

SPEAKER				
	.VOLUME	type Numeric	Access R/W	Special
	Desc.	The overall volume level of the speaker can be set to one of four different volume settings by writing to the VOLUME property. The range of valid values is from 1 (quietest) to 4 (loudest).		
	. BEEPENABLE	type Numeric	Access R/W	Special
	Desc.	Simple beeps can be created by writing a true value into the BEEPENABLE property. The duration and frequency of the beep is set by the BEEPTIME and BEEPFREQ properties. Once a beep has been started (by writing true to BEEPENABLE), it can be stopped anytime by writing false to BEEPENABLE, or will automatically stop once the programmed duration has been reached.		
	.BEEPTIME	type Numeric	Access R/W	Special
	Desc.	The BEEPTIME property sets the duration of the beeps. The acceptable range of values is from 1 to 10,000 milliseconds.		
	.BEEPFREQ	type Numeric	Access R/W	Special
	Desc.	The BEEPFREQ property sets the frequency of the beeps. The acceptable range of values is from 100 to 5000 Hz.		
	. SIGNALGAIN	type Numeric	Access R/W	Special
	Desc.	In order to monitor the signals generated by the AI-80 internally with the speaker, the SIGNALGAIN property can be set to a non-zero value. The higher the value, the higher the signal gain. The valid range for this property is from 0 (off) to		

	10.0.			
	. TELINTGAIN	type Numeric	Access R/W	Special
	Desc.	Similar to the SIGNALGAIN property, the TELINTGAIN property can be used to monitor signals with the speaker, the signals present at the active telephone interface port. This represents the signals generated by the AI-80 in addition to those generated by any connected device. The valid range for this property is from 0 (off) to 10.0.		
	. CPEGAIN	type Numeric	Access R/W	Special
	Desc.	Similar to the SIGNALGAIN property, the CPEGAIN property can be used to monitor with the speaker, the signals present at the active CPE load port. The valid range for this property is from 0 (off) to 10.0.		

The high volume settings and high gain settings may be needed to effectively monitor faint or low level signals. However, the presence of strong or high level signals may cause excessive distortion from the speaker. In this case, the gain and/or volume settings should be decreased.

COMM Object

The COMM object's properties control the AI-80's serial RS-232 interface. While normally connected to a host computer for downloading and developing programs, the serial port can used by programs for communicating to other devices, such as serial printers. Note that the AI-80's serial interface is fixed to communicate with other devices using 8 bits, no parity, 1 stop bit. The baud rate is programmable, but limiting the other common serial settings minimizes possible troubleshooting.

COMM	**.BAUD**	type Numeric	Access Write	Special
	Desc.	The BAUD property sets the serial interface's baud rate to 1 of 5 different values according the following table: 0) 9600 baud (default setting on power up) 1) 19200 baud 2) 38400 baud 3) 57600 baud 4) 115200 baud Note that the AI-80 will automatically reset the baud rate to its default setting (9600) when a break signal is detected. As such to ensure communication with a host system, the host should initially send a break signal. Once communications have been established at 9600 baud, the rate can be changed to other values.		
	.RXCOUNT	type Numeric	Access Read	Special
	Desc.	The RXCOUNT property is a read only value that reflects the number of data bytes present in the serial receive buffer. Under normal operation, system monitoring software within the AI-80 will always empty the receive buffer before any programs detect data present.		
	.GETBYTE	type Numeric	Access Read	Special
	Desc.	The GETBYTE property returns the next byte available from the serial receive buffer and decrements the RXCOUNT property. If no bytes are contained within the receive buffer, reading GETBYTE will return a copy of the last byte received.		
	.SENDBYTE	type Numeric	Access Write	Special
	Desc.	The SENDBYTE property sends a single byte value to the serial transmit buffer. Any bytes in the transmit buffer will automatically be transmitted at the selected baud rate. Before writing to the SENDBYTE property, the TXFREE		

	property should be checked to ensure there is enough space in the transmit buffer.			
	.SENDSTRING	type String	Access Write	Special
	Desc.	Similar to the SENDBYTE property, the SENDSTRING property is used to send a string of ASCII data to the serial transmit buffer. Before writing to the SENDSTRING property, the TXFREE property should be checked to ensure there is enough space in the transmit buffer		
	.RXSTATUS	type Numeric	Access Read	Special
	Desc.	The RXSTATUS property returns the status of the receive serial buffer. The values it returns and their meanings are as follows: 0) No error detected 1) Parity or framing errors detected 2) Overflow in the receive data buffer 3) Break condition detected Only the highest condition code is returned, and reading the RXSTATUS property will zero the property value until the next error condition is detected.		
	.TXFREE	type Numeric	Access Read	Special
	Desc.	The TXFREE property returns the number of bytes unused in the serial transmit buffer. If TXFREE returns zero, then no more data should be sent until the buffer clears itself.		
	.CTS	type Numeric	Access Write	Special
	Desc.	The Clear to Send (CTS) output signal can be set active by writing a true value to the CTS property. Likewise, writing a false value to the CTS property will deactivate the CTS output signal.		
	.RTS	type Numeric	Access Read	Special
	Desc.	The RTS property returns the state of the Request to Send (RTS) input signal. If the RTS signal is active, the RTS property will return true. If the signal is inactive, the RTS property returns false.		

FILE Object

The FILE properties provide limited access the AI-80 file system. They can be used to determine whether or not certain files exist within the flash memory.

FILE	**.IDLOW**	type Numeric	Access R/W	Special
	Desc.	The IDLOW property is used to specify the low 16 bits of the 32 bit file ID code being sought. The lower 16 bits are used to represent the file number, while the upper 16 bits represent the file type.		
	.IDHIGH	type Numeric	Access R/W	Special
	Desc.	The IDHIGH property is used to specify the high 16 bits of the 32 bit file ID code being sought. The high 16 bits are used to represent the file type, while the lower 16 bits represent the file number.		
	.EXIST	type Numeric	Access Read	Special
	Desc.	The EXIST property is read only, and returns either true or false depending on whether the file ID (specified by IDLOW and IDHIGH) exist. If the file exists in the flash memory, the EXIST property will return true.		

SYSTEM Object

The SYSTEM properties return various version strings along with what hardware options have been installed in the AI-80. All the properties are read only.

SYSTEM				
	.UNITID	type String	Access Read	Special
	Desc.	The UNITID property returns an ASCII character string that represents the AI-80 ID code.		
	.SOFTID	type String	Access Read	Special
	Desc.	The SOFTID property returns an ASCII character string that contains the primary software version number.		
	.HALID	type String	Access Read	Special
	Desc.	The HALID property returns an ASCII character string that contains the hardware abstraction layer version number.		
	.FFSID	type String	Access Read	Special
	Desc.	The FFSID property returns an ASCII character string that contains the flash memory file system version number.		
	.VTPID	type String	Access Read	Special
	Desc.	The VTPID property returns an ASCII character string that contains the virtual interpretive processor version number.		
	.OPTIONS	type Numeric	Access Read	Special
	Desc.	The OPTIONS property returns a numeric value that represents what hardware options are installed in the AI-80. Each option is represented by a binary bit. If the specific option's bit is set in the returned value, that option is present. The presently assigned option bits are: Bit 0) Complex & External Line Impedances Bit 1) Telephone Line Length Module Bit 2) n/u Bit 3) I/O Expansion Module		
	.HALTCMDS	type Numeric	Access R/We	Special Restrict.
	Desc.	The HALTCMDS property is used to prevent the AI-80 from receiving any RS-232 data sent from the PC. Writing a non-zero value stops the AI-80 from listening to any commands sent from the PC. All the bytes received from the PC are stored in the received data buffer, which is accessed via the COMM.GETBYTE property. This mode allows the AI-80 programs to communicate with external devices over the RS-232 port without the AI-80 attempting to interpret the data as commands. Note that if a line break condition is detected, this property is returned to its default value of zero. As such, normal communications can be restored by asserting a line break condition.		

IO Object

The optional IO Module extends the hardware capabilities of the AI-80, by adding a number of features. These include digital input and output signals, audio signal I/O, and DC voltage measurement. Programming control over the IO module is performed in the same manner as the other AI-80 hardware resources. The "IO" object represents all the hardware properties associated with the IO Module.

Programs that make use of the IO object, should check that the IO Module is installed in the AI-80 before accessing its properties. This is done by reading the "DEVICEID" property. If it returns a non-zero value, the IO Module is installed.

IO

.DEVICEID	type Numeric	Access Read	Special
Desc.	This read-only property returns an ID number for the IO Module. Normally this value is 802. However, if the IO Module is not installed in an AI-80, reading this value will return zero. All programs should first read the device ID before accessing the IO Module to determine that the IO Module is present.		

.VERSION	type Numeric	Access Read	Special
Desc.	The IO Module version property returns a single number that represents its revision code. Currently this value is fixed at 1; however, future enhancements may return a different value.		

.NAME	type String	Access Read	Special
Desc.	This property returns a string representing the name of the IO Module.		

.SERIAL	type String	Access Read	Special
Desc.	The SERIAL property is the serial number for the IO Module. Each serial number string is unique and this is a read-only property.		

.DOUT	type Numeric	Access R/W	Special
Desc.	The eight digital output signals are controlled by the DOUT property. Writing a value between the limits of 0 and 255 will set the output signals to eight a high or low state depending on their bit position. Bit 0 in the written value controls the logic state of output 1, while bit 1 controls output 2, and so on. If the bit is set, the output logic level will be a one, or high state. Likewise, for a clear bit, the output logic level will be zero, or low state. Reading the DOUT property returns the last value written.		

.DIN	type Numeric	Access Read	Special
Desc.	Reading the DIN property returns the logic level present at the eight digital input signals. Each digital input is represented by one of eight different bit positions in the returned value. Bit 0 represents input 1, bit 1 represents input 2, and so on. If the input signal is at a logic one (high state), its bit value will be set. The value returned by DIN will always range between 0 and 255.		

.BITSET	type Numeric	Access Write	Special
Desc.	BITSET can be used to change any digital output signal, or digital in/out signal to a logic one, or high state. Writing a value of 1 to 8 will set digital output signal 1 to 8 to a logic one level. Writing a value of 9 to 15 will force digital in/out signals 1 to 7 to be an output with a logic level of one (high state). Note that digital in/out signals 1 and 2 can also be used for asynchronous serial communications. If enabled (IO.COMMBAUD <> 0), then the BITSET property will have no effect on these two digital in/out signals.		

.BITCLEAR	type Numeric	Access Write	Special
Desc.	Similar to the BITSET property, BITCLEAR will change any digital output signal, or digital in/out signal to a logic zero, or low state. Values 1 to 8 represent digital outputs 1 to 8, while values 9 to 15 represent digital in/out signals 1 to 7. As with the BITSET property, if the asynchronous serial communication function is enabled (IO.COMMBAUD <> 0), then this property has no effect on digital in/out signals 1 and 2.		

.BITINPUT	type Numeric	Access R/W	Special
Desc.	This property performs two functions. First it will force a digital in/out signal to be an input, if specified. Second, it determines which bit is read by the GETBIT property. Writing a value of 1 to 8 specifies that the GETBIT property will return		

	the logic state of digital input 1 to 8. A value of 9 to 15 returns the logic state of the digital in/out signal 1 to 7. Finally, a value of 17 to 24 causes GETBIT to return the state of the digital output signal 1 to 8. If the asynchronous serial communication function is enabled (IO.COMMBAUD <> 0), then this property has no effect on digital in/out signals 1 and 2.		
.GETBIT	type Numeric	Access Read	Special
Desc.	GETBIT is a read-only property that returns the logic state (one or zero) of the digital signal specified by the BITINPUT property. BITINPUT values or 1 to 8 return the digital input signal 1 to 8 state, 9 to 15 return the digital in/out signal 1 to 7 state, while values 17 to 24 return the digital output signal 1 to 8 state.		
.AINCHANNEL	type Numeric	Access R/W	Special
Desc.	Four of the digital input signals can also be used as analog input channels for DC voltage measurement. The AINCHANNEL property specifies which digital input signal's DC voltage will be measured when reading AINLEVEL. The channel value can range from 1 to 4, representing digital input signals 5 to 8 respectively.		
.AINLEVEL	type Numeric	Access Read	Special
Desc.	This property returns the DC voltage present at digital input signals 5 to 8, as specified by the AINCHANNEL property. The measurement range for the DC voltage is from -10 volts to + 10 volts. The IO Module performs continuous DC voltage measurements on the selected channel at a rate of once every 1.1 msec. Reading this property returns the results of the last measurement.		
.AINCOMPARE	type Numeric	Access Read	Special
Desc.	In addition to digital inputs, signals 0 and 1 can be used as an analog voltage comparator. If the voltage level at digital input 0 is higher than the voltage at digital input 1, then reading AINCOMPARE will return TRUE (-1). Otherwise, it returns FALSE (0). The voltages present at digital inputs 0 and 1 must be between 0 and 5 volts.		
.DCLEVEL	type Numeric	Access Read	Special
Desc.	This property returns the measured DC voltage present at the rear banana jacks of the AI-80. The voltage measured is returned in units of volts and can range from -200 to +200. It is important to wait until the DC measurement is complete before reading this property. If the measurement is still in progress, the returned value may be inaccurate.		
.DCTRIGGER	type Numeric	Access R/W	Special
Desc.	The DCTRIGGER property is used to control and monitor the progress of the DC voltage measurement. Writing a non-zero value to this property will start a DC measurement cycle. The time required to complete the measurement is controlled by the DCTIME property. During the measurement cycle, reading DCTRIGGER will return the value of 1. Once the measurement is complete, this property will return a value of zero. The normal procedure for performing a measurement is to write to this property with a value of 1. Then wait until it returns zero. At that time the DCLEVEL property can be read with the measurement results.		
.DCTIME	type Numeric	Access R/W	Special
Desc.	This property specifies DC measurement cycle duration in units of seconds. The valid range of values is from 0.001 to 1.0 seconds, with a default value of 0.1 seconds. Longer measurement times are useful in rejecting AC signals, such as 50/60 Hz hum noise. If 50/60 Hz power line hum is causing measurement instability, the DCTIME value should be increased in multiples of the power line cycle time (20		

msec for 50 Hz, and 16.7 msec for 60 Hz).

.AUDIOOUT	type Numeric	Access R/W	Special

Desc.

The AUDIOOUT property controls the function of the BNC Audio Output. Its value can range from 0 to 5 with the following functionality:

0) Off. Turns off the BNC Audio Output.

1) Signal Generator Monitor. The BNC Audio Output provides a monitor for the signals being sent to the telephone interface. The level at the BNC output is 6 dB less than the level at the telephone interface port (unterminated). For example, if ToneB is generating a 1 Vrms tone, the level at the BNC output will be 0.5 Vrms.

2) Telephone Interface Monitor. In this setting the BNC output reflects the signal present on the tip and ring leads of the telephone interface. Both the signals generated by the AI-80 and any signals generated by a connected CPE will be present at the BNC output. Like mode 1 above, the signal level at the BNC output is 6 dB less than the level at the telephone interface port.

3) CPE Load Monitor. The BNC output monitors the signals present at the CPE Load tip and ring leads. The output level is 6 dB less than the level at the tip and ring leads.

4) Speaker Output Monitor. In this mode, the BNC Audio Output monitors the signals routed to the AI-80's speaker. The speaker signals can be a combination of the tone generators, telephone interface, and CPE load signals, as controlled by the SPEAKER properties.

5) Signal Generator Output. Similar to mode 1 above, the output of the signal generated is routed to the BNC output. However, the signal is NOT route to the telephone interface as well. The TONEA, TONEB, and MFGEN tones are only present at the BNC Audio Output connector. In this mode, the signal levels are as specified. For example, if the TONEB level is specified at 1 Vrms, the level at the BNC output will be 1 Vrms. Note the maximum output level at the BNC connector is 2 Vrms.

.AUDIOIN	type Numeric	Access R/W	Special

Desc.

The AUDIOIN property is used to enable measurements of the signals fed into the BNC Audio Input. Writing a TRUE value (non-zero) modifies the SOURCE property of the MEASURE and DTMF objects, such that if the SOURCE value is set to 1, the signal source will be the BNC input as opposed to the CPE Load. The following table summarizes the possible signal sources for measurement:

Signal Source	SOURCE prop.	AUDIOIN prop.
1) Telephone Interface	0	X (don't care)
2) CPE Load	1	0 (False)
3) CPE Load (low gain)	2	0 (False)
4) Audio BNC Input	1	1 (True)

The maximum input level that can be applied to the BNC input is 2 Vrms. Note that the input is DC coupled.

.AUDIOMIX	type Numeric	Access R/W	Special

Desc.

Audio signals present at the BNC input can be mixed with the tone generator output and routed to the telephone interface. When the AUDIOMIX property is set to TRUE (non-zero), signals applied to the BNC input are increased in gain by 6 dB and routed to the telephone interface. Any tones being generated are mixed with the signals from the BNC input. Setting the property to FALSE (zero), disables the mixing function.

.PMODE	type Numeric	Access R/W	Special

Desc.	The PMODE and PTIME properties can be used to measure the duration of digital pulses present at digital in/out 7. To use this function, in/out pin 7 must be set as an input with the BITINPUT property. Writing to the PMODE property starts a timing measurement in one of the following modes: 0) Off - Turns off pulse timing mode 1) +ve Pulse - Measures the time between the first rising edge and falling edge. 2) -ve Pulse - Measures the time between the first falling edge and rising edge 3) Rising Edges - Measures the time between the first rising edge and second rising edge. 4) Falling Edges - Measures the time between the first falling edge and second falling edge. The results and status of the measurement can be read by the PTIME property.

.PTIME	type Numeric	Access Read	Special
Desc.	Working with the PMODE property, reading PTIME returns the pulse timing measurement in seconds. After writing to PMODE with a value from 1 to 4, the IO Module will start a pulse timing measurement. While the measurement is taking place, PTIME returns the value of zero. When the measurement is complete, PTIME will return a non-zero value. The ranging in timing is from 0.1 msec to 100 seconds with an accuracy of +/- 0.2 msec. Note, if a pulse time is measured to be less than 0.1 msec, then PTIME will return a value of 0 even though the measurement is complete. As such, it is always prudent to employ a time-out such that programs do not wait endlessly for a non-zero value.		

.PULSEMODE	type Numeric	Access R/W	Special
Desc.	The PULSEMODE property is used to control a number of pulse counting and pulse generation features. By writing a value from 0 to 5, the following modes can be enabled: 0) Off - No pulse counting or pulse generation active 1) Pulse Counting (free run) - In this mode, a 24 bit counter is incremented at every rising edge detected on digital in/out pin 6. The value of the 24 bit counter can be read or written to by accessing the PULSECOUNT property. The maximum pulse frequency applied to digital in/out pin 6 should not exceed 1 MHz. 2) Pulse Counting (gated) - Similar to the free run pulse counting mode; however, the pulses are only counted for a specified duration. When in this mode, writing to the PULSEGATE property clears the PULSECOUNT value and counts the number of rising edges at digital in/out 6 for the duration specified by PULSEGATE. Once the gating time has expired the PULSECOUNT value is frozen. 3) Pulse Generation (single shot) - In this mode of operation, the timer output pin generates a single positive going pulse of a specified duration by writing to the PULSEDURATION property. 4) Pulse Generation (continuous) - Enabling this mode causes the digital timer output pin to generate a continuous square wave with a frequency defined by the PULSEFREQ property. 5) Pulse Width Modulator - Two independent pulse width modulators can be enabled for generating analog output values. Each PWM has a resolution of 10 bits with a repetition rate of 1.8 kHz. The digital timer output is the source for the first PWM channel, while digital in/out pin 4 is the source for the second PWM channel. If a low pass filter is applied to the PWM output, the result is an analog output voltage between 0 and 5 Volts, with 10 bits of resolution.		

Note: to use the second PWM channel, the digital in/out pin 4 must be configured as an output with either the SETBIT or CLEARBIT properties.

Note: for the pulse counting modes (1 and 2), the digital in/out pin 6 must be set to be an input signal by using the BITINPUT property.

.PULSECOUNT	type Numeric	Access R/W	Special

Desc. Used in the pulse counting modes (1 - free run, 2 - gated), reading this property returns the number of counts (rising edge) detected at digital input pin 6. The counter is 24 bits wide representing values from 0 to approximately 16.7 million.

.PULSEGATE	type Numeric	Access R/W	Special

Desc. The PULSEGATE property is used to specify the gating time for the pulse counter. When PULSEMODE is set to 2, writing to this property will clear the pulse counter and allow counting of the rising edges at digital in/out 6 for the time specified. The gate time can range from 0.001 to 100 sec. The PULSEGATE value is decremented to zero while the pulse counting gate is open. As such, a program can write a value to the PULSEGATE property and then wait till the PULSEGATE value reaches zero. At that time the pulse counting is completed and the PULSECOUNT property can be read. The accuracy of the pulse gate value is +/- 0.2 msec.

.PULSEDURATION	type Numeric	Access R/W	Special

Desc. This property is used to specify the duration of a single-shot positive pulse generated at the digital timer output pin. If the PULSEMODE property is set to 3, then writing a value between 0.001 and 100 seconds generates a pulse of that duration. Reading the PULSEDURATION property returns the amount of time left until the pulse is complete. As such, to generate a pulse of 100 msec, write the value of 0.1 to PULSEDURATION and then continuously read PULSEDURATION until the value returned is zero. That will mark the end of the pulse. The accuracy of the pulse generated is +/- 0.2 msec.

.PULSEFREQ	type Numeric	Access R/W	Special

Desc. The PULSEFREQ property is used to control the frequency of a square wave generator. Active only if the PULSEMODE property is set to 4, the digital timer output pin can generate a square wave with a frequency between the limits of 0.05 Hz and 100,000 Hz.

Note: The accuracy of the frequency generated is inversely proportional to its frequency. The IO Module uses a master clock frequency of 3.6864 MHz and divides it by an integer value to create the square wave output. As such, the frequency resolution at progressively higher frequencies worsens. The following table shows the accuracy given various frequencies:

100 Hz Output:	Accuracy < 0.1 %
1 kHz	< 0.1 %
10 kHz	< 0.3 %
100 kHz	< 3.0 %

.PWM1	type Numeric	Access R/W	Special

Desc. This property represents the pulse width modulator output value for channel #1. The valid range of values is from 0 to 1023. The PWM output is only active if the PULSEMODE property is set to 5. If enabled, the PWM output is present at the digital timer output pin.

.PWM2	type Numeric	Access R/W	Special

Desc.	This property represents the pulse width modulator output value for channel #2. The valid range of values is from 0 to 1023. The PWM output is only active if the PULSEMODE property is set to 5. If enabled, the PWM output is present at the digital in/out pin 4, provided that it has been set as an output with the SETBIT or CLEARBIT property.		
.COMMBAUD	type Numeric	Access R/W	Special
Desc.	An alternate function to the digital in/out pins 1 and 2 is an asynchronous receiver and transmitter. If enabled, this function can send and received serial data at a number of different baud rates. The COMMBAUD property is used to enable this function and set the baud rate (bits per second) as follows: 0) Off - This function is disabled 1) Enabled at 1200 baud 2) Enabled at 2400 baud 3) Enabled at 9600 baud 4) Enabled at 19200 baud Note that if the value written to COMMBAUD is non-zero (function enabled), then digital in/out pin 1 is forced as an input (for receiving data) while digital in/out pin 2 is forced as an output (for transmitting data).		
.COMMPARITY	type Numeric	Access R/W	Special
Desc.	The COMMPARITY property controls the parity mode and number of data bits used for both receiving data and transmitting data. The valid settings are as follows: 0) no parity, 8 data bits 1) odd parity, 7 data bits 2) even parity, 7 data bits In all modes, the least significant data bit is both received and transmitted first. For data framing, 1 start bit and 1 stop bit are used.		
.COMMSENDBYTE	type Numeric	Access Write	Special
Desc.	Writing to the COMMSENDBYTE property places a single byte (value 0 to 255) into a buffer for transmission. The byte is then modified according to the parity setting and transmitted at the specified baud rate. This property is write only and should only be written to when the transmitter buffer is empty. The state of the transmitter buffer is returned by reading COMMTXEMPTY. If the returned value is TRUE (non-zero), then the buffer can accept more data.		
.COMMSENDSTRING	type String	Access Write	Special
Desc.	Similar in function to the COMMSENDBYTE property, writing to COMMSENDSTRING will transfer an entire string into a buffer for transmission one character at a time. Each character in the string is modified to match the current parity setting. The maximum string length is 64 characters. This property is write only and should only be written to when the transmitter buffer is empty. The state of the transmitter buffer is returned by reading COMMTXEMPTY. If the returned value is TRUE (non-zero), then the buffer can accept more data.		
.COMMTXEMPTY	type Numeric	Access Read	Special
Desc.	This read-only property returns either TRUE (non-zero) or FALSE (zero) depending on whether or not the transmitting buffer is empty. Unless the returned value is TRUE, no data should be written to the transmitting buffer.		
.COMMRXCOUNT	type Numeric	Access Read	Special
Desc.	Reading the COMMRXCOUNT property returns the number of bytes that have been received and placed into a buffer. If		

	no bytes have been received, then a value of zero is read. The maximum number of characters that can be stored in the receive buffer is 63 characters. Beyond that amount, an over-flow error is reported by COMMRXERROR. Bytes are removed from the buffer by reading COMMGETBYTE.		
.COMMRXERROR	type Numeric	Access Read	Special
Desc.	This read-only property returns the error status of the data receiver. A value of zero represents no error detected, while a non-zero value indicates an error as follows: 0) No error 1) Parity - a byte was received without the correct parity 2) Framing - a byte was received without a proper stop bit 3) Over-flow - the receive data buffer has been over filled and data has been lost. Reading the COMMRXERROR property will reset the error status back to zero.		
.COMMGETBYTE	type Numeric	Access Read	Special
Desc.	COMMGETBYTE is used to read data from the receive data buffer. If the COMMRXCOUNT is greater than zero, reading COMMBETBYTE will return a value between 0 and 255. If no data is contained in the buffer (COMMRXCOUNT=0), then reading COMMGETBYTE will return a value of -1.		
.MEMREGISTER	type Numeric	Access R/W	Special
Desc.	The IO Module can be used to store either 32 numeric values or 128 characters in non-volatile memory. This memory space is available for user applications and its contents will remain intact even if the AI-80 is turned off or reset. Access to this memory space is performed thought the MEMREGISTER property. This property is used to specify which register location is being accessed. The valid range of values is from 1 to 32. Each register can contain either a single numeric value or 4 characters. Strings longer than 4 characters use successive registers. To write or read from any of the registers, set MEMREGISTER to the register number. Then access the MEMWRITENUM, MEMWRITESTRING, MEMREADNUM, or MEMREADSTRING properties. If the access was successful, reading the MEMREGISTER value will return -1. Note: Strings are always terminated by the null character (0). As such, writing a string of 4 characters requires 2 registers. The first register stores the four characters, while the second register stores the null character.		
.MEMWRITENUM	type Numeric	Access Write	Special
Desc.	This property is used to write a numeric value to a memory register in the IO Module's non-volatile storage. First a register value between 1 and 32 must be specified with the MEMREGISTER property. Then writing a numeric value to this property will transfer that value to the memory register.		
.MEMWRITESTRING	type String	Access Write	Special
Desc.	This property is used to write a character string to a memory register(s) in the IO Module's non-volatile storage. First a register value between 1 and 32 must be specified with the MEMREGISTER property. Then writing a string to this property will transfer that value to the memory register. Each register can store 4 characters. If the string is longer, then successive registers are over-written until the entire string is stored.		
.MEMREADNUM	type Numeric	Access Read	Special
Desc.	This property is used to read a numeric value from a memory register in the IO Module's non-volatile storage. First a memory register value between 1 and 32 must be specified with the MEMREGISTER property. Reading this property		

	returns the value stored in the register.			
	.MEMREADSTRING	type String	Access Read	Special
	Desc.	This property is used to read a character string from a memory register in the IO Module's non-volatile storage. First a memory register value between 1 and 32 must be specified with the MEMREGISTER property. Reading this property returns the string store in the register. Since strings are terminated with the null (0) character, if no null character exists in the specified register, successive registers are read until the null character is detected.		

Section 4-3 System Software Overview

The AI-80 allows for a large degree of flexibility and customization in order to support a wide range of applications. Key to this flexibility is the structure of the system program files stored in the on board non-volatile memory. This section presents an overview of the system files, along with showing how to customize the AI-80 functionality.

System Files

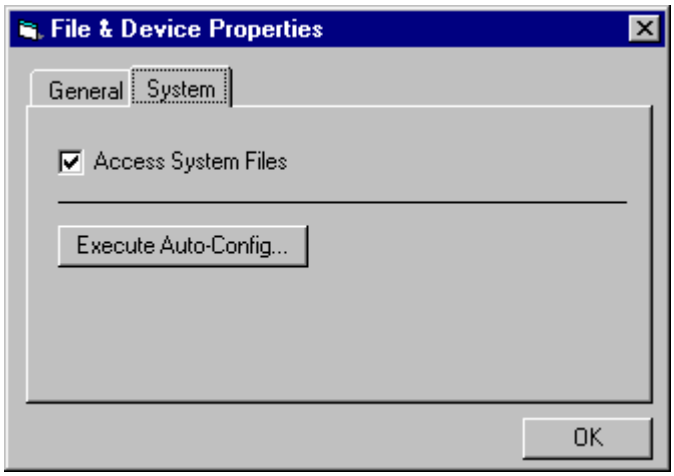
All of the software used to operate the AI-80 is contained in non-volatile flash memory. Being re-programmable, it allows field upgrading of the software, as new features become available. However, since the software can be changed, there is a potential for unintentionally modifying and erasing key system files. This can cause improper operation or result in an AI-80 that is effectively unusable. Fortunately, the AI-80 has been designed to be recoverable from a corruption of any of the key flash memory files.

Three different types of files are stored within the AI-80's flash memory. They are:

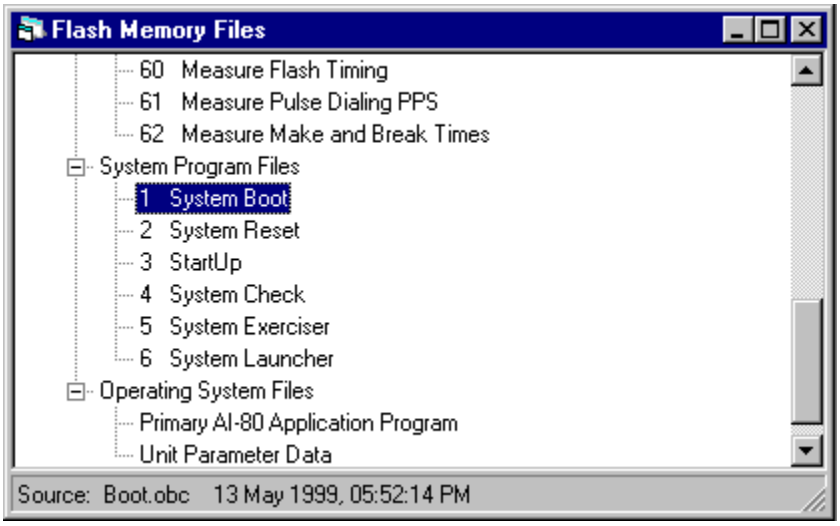
- User Program Files
- System Program Files
- Operating System Files

Normally, the A.I.WorkBench software will only display the User Program Files. These are the program files created with the A.I.WorkBench software and perform the various telephone related high level testing functions. The System Program Files, though also created with the A.I.WorkBench software, perform various system house keeping related functions. These includes resetting the AI-80 hardware properties, performing system checks, and controlling the execution of user developed programs. All of the source code for the system program files are included in the A.I.WorkBench software. This allows for customization of these programs for special applications. The third category of files is the Operating System Files. These files provide the low level code and data that the AI-80's internal processor runs with. Missing operating system files will generally prevent the AI-80 from executing properly, and usually requires the complete reloading of all the files in the flash memory.

All the files contained in the AI-80 flash memory, including the system files, can be viewed in the Flash Memory Files window. Before the system files can be accessed or modified, the Access System Files option must be enabled. This is done by selecting the AI-80 Caller ID Generator item at the top of the Flash Memory Files window, and then choosing the [FILE] [PROPERTIES] menu command (or right click the mouse). The displayed windows will show various properties of the AI-80. Click the mouse on the System tab and enable the Access System File check box, as shown below.



After clicking the OK button, the file list in the Flash Memory Files window is refreshed with all the files present in the AI-80 non-volatile flash memory. Following the User Program Files, the System Program Files and Operating System Files are displayed in format similar to the figure below.



In order to understand the purpose and functionality of each of the system files, the AI-80 “boot”, or initialization, process needs to be described.

System Initialization

Before any customization of the system files is performed, it is important to understand how they are used and work with each other. Many of the files are only accessed during the initialization of the AI-80, while other system files (like the System Launcher) is continually executing as a separate process monitoring and controlling the user programs.

The sequence of steps that occur in the AI-80 when the unit is turned on, is as follows:

1. Following a hardware reset of the AI-80 internal processor, a “System Loader” program (stored in the flash memory) is executed. Though contained in the flash memory with all the other system and user files, the System Loader is contained in a protected section that can not be overwritten. Unless a hardware fault exists within the AI-80, the System Loader program will always execute after the unit is turned on.

2. The System Loader program searches the flash memory for the Primary AI-80 Application program. If located, the System Loader retrieves the program, loads it, and then passes execution to it. If for any reason, the System Loader can not load the program it will display “Err1” on the front panel display.

3. The Primary AI-80 Application Program implements the low level system functionality controlling all the AI-80 hardware aspects. This includes the host PC communications and the multiple virtual interpreter processors that execute the user and system programs.

4. One of the initial tasks of the primary program is to retrieve the Unit Parameter Data file from the flash memory. This file contains various calibration parameters used to improve the accuracy of the AI-80 measurements. If this file can not be found, the program will display “Err2” on the front panel display. In the situation the Unit Parameter Data file is present in the flash memory, but some or all of the correction factors are invalid or corrupted, the AI-80 will display “CAL” on the front panel and flash the Start LED. This indicates that the correction factors could not be loaded. Pressing the Start will resume normal operation without the correction factors present.

5. Once the Unit Parameter Data file is loaded, the AI-80 will search for system file #1 and execute it. If this file is not present in the flash memory, the front panel display shows “Err3”.

6. The System Boot program (file #1), is the first program file that can be customized by the A.I.WorkBench software. All of the source code files for the system files are included in the A.I.WorkBench software. The Boot program performs two simple functions. Firstly, it launches another program called System Reset. System Reset sets all the hardware properties of the AI-80 to their default state. Once System Reset has finished, the boot program decides what program to execute next dependent on any keys being pressed. If no keys are pressed, the boot program launches the Startup program, and terminates itself. If the Pause key is held for more than 2 seconds, the boot program launches the System Check program. Likewise if the Stop key is held for more than 2 seconds, the System Exercise program is launched.

7. Assuming no keys have been pressed during the System Boot program, it launches the Startup program. The Startup Program is very short and is meant to be easily modified. It sets the initial program number for the System Launcher program. Normally this is 10. It then launches the System Launcher program and terminates itself.

8. The System Launcher program is one of the most complex system programs, in that it manages the execution of all the User Programs. Using the initial program number from the Startup program, the System Launcher displays the program number on the front panel and monitors

the key pad for any user initiated actions. Pressing the Start key launches the selected user program, while the Pause key and Stop key suspend and terminate the user program respectively. The System Launcher program monitors the progress of the user program and once the user program has finished, the System Launcher will reset all the AI-80 hardware properties.

Customization

Since the source code for the system program files are included with A.I.WorkBench software, the programs can be modified to suit custom applications. Three of the most common and useful customization techniques are:

- Changing the default hardware settings
- Changing the default startup program number
- Changing the user interface and operation

Changing the Default Hardware Settings

In many cases, the AI-80 is required to use a hardware setting that is different from the default value. Examples of this include the telephone interface settings. If restricted to testing certain types of telephone, the line impedance may require a setting of 600 ohms instead of the default 900 ohm. Or Port B is used as the active port instead of Port A (for testing two line telephones).

Changing the default hardware settings, requires modifying the System Reset program. This program is launched when the AI-80 is first initialized and once any user program has finished its execution. Thus any changes made in the System Reset program are always reflected in the AI-80 hardware settings. The steps to modifying the program are as follows:

1) Load the project “projects\standard\system\reset.prj” file
2) Modify the source file with the built-in editor
3) Compile the program (Shift-F5)
4) Load the modified program into the flash memory (Ctrl-W)

Changing the Default Startup Program Number

A second common customization technique is to change the default startup program number. As a default value, user program number 10 (Bellcore Type I - Multiple Message Format Caller ID) is always selected at power up. If other programs are used more often, the default startup program number can be changed.

The setting for the default program number is contained in the Startup program file. It can be changed as follows:

1) Load the project “projects\standard\system\startup.prj” file
2) Using the source file editor modify the line:
   const LauncherStart = 10
   with the new default program number.
3) Compile the program (Shift-F5)

4) Load the modified program into the flash memory (Ctrl-W)

Changing the User Interface and Operation

The AI-80 user interface is controlled by the System Launcher program. It allows the selection of different programs along with their controlled execution. However for some application environments, such as production and manufacturing of telephone devices, usually a single specialized program is executed. In these environments, the user interface may require changes to support specific production requirements.

For these applications, the System Launcher may not be required and could be replaced by a custom control program. In this case, the custom program should be started on power up, instead of the System Launcher. This can be accomplished by modifying slightly the Startup program. It should be changed as follows:

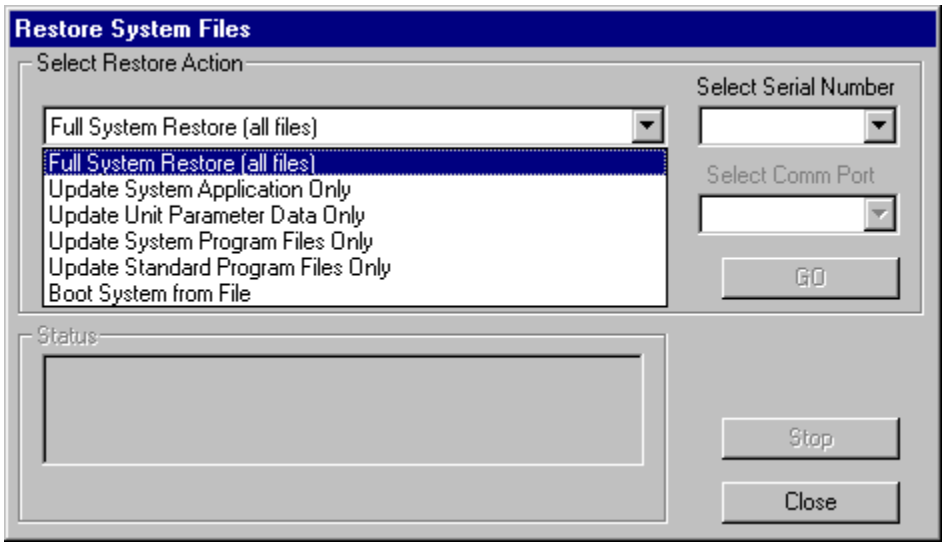
1) Load the project "projects\standard\system\startup.prj" file
2) Using the source file editor modify the line:
   const StartProgram = 6
   with the new custom control program number.
3) Compile the program (Shift-F5)
4) Load the modified program into the flash memory (Ctrl-W)

Restoring System Files

Due to the flexible software architecture of the AI-80, certain situations exist that could cause the flash memory file system to become corrupted. These include accidentally modifying or deleting system files, power loss during file transfers, and serial communication faults during file transfers. Under all of these conditions, the files required to restore the AI-80 are included with the A.I.WorkBench software.

As updates are made available, new data files can be transferred from the A.I.WorkBench software to the AI-80. This section deals only with restoring a damaged file system or individual files. For information on updating the AI-80 files with a newer version, see section 4-6: Updating System Files.

Selecting the [HELP] [RESTORE SYSTEM FILES] menu command displays a window similar to the following figure. Within this window, several update options can be selected, depending on the desired action and fault experienced. One of six different restore actions can be performed, ranging from a complete re-initialization of all files to simply updating the standard program files.



The following table lists various fault conditions along with suggested remedies.

Fault: ERR1 on display	Cause: The primary AI-80 application program can't be found or is corrupted. This will prevent the A.I.WorkBench software from communicating with the AI-80, and prevent any file transfers.
	Remedy: Select the "Full System Restore" action. This re-initializes the AI-80 file system and loads all the factory default files. All user programs store in the flash memory are lost. If it is important to recover the user programs before re-initializing the file system, perform the following actions: a) Select the "Boot System From File" option. This will transfer the AI-80 application program via the serial port, and allow the A.I.WorkBench software to establish a connection to the AI-80. b) Select the [FILE] [READ ALL FILES FROM FLASH] menu item and select all the files to save. c) Once the files have been saved, select the "Full System Restore" action to restore all the system files. d) Load the user program files that were saved in step b. Note that it may not be possible to recover any of the user program files if the AI-80's file system is severely damaged.

Fault: ERR2 on display, or CAL flashing on the display	Cause: The Unit Parameter Data file is missing or corrupted. This file contains the calibration data for the AI-80 and can not be loaded by the primary appellation program.
	Remedy: Select the "Update Unit Parameter Data Only" action. This will restore the file provided the file system is not damaged. If this operation does not work due to a damaged file system, then select "Full System Restore" and follow the procedure listed for "ERR1"

Fault: ERR3 on the display	Cause: One or more of the system program files are missing or corrupted.
	Remedy: Select the "Update System Program Files Only" action. This restores all the system files provided the file system is not damaged. If this operation does not work due to a damaged file system, then select "Full System Restore" and follow the procedure listed for "ERR1"

Fault: Programs P10 through P99 fail to operate	Cause: One or more of the standard programs may have been deleted or is corrupted.
	Remedy: Select the "Update Standard Program Files Only" action. This restores all the standard program files (P10 through P99) provided the file system is not damaged. If this operation does not work due to a damaged file system, then select "Full System Restore" and follow the procedure listed for "ERR1"

Full System Restore Action

This action completely restores all the files contained within the AI-80. It should be used if the AI-80 display shows ERR1 on power up, or the A.I.WorkBench is unable to establish communications with the AI-80. It returns the AI-80 to a factory default state and erases any user programs stored in the non-volatile memory.

Before this action can be started, the unit's serial number must be selected from the drop down list box. It is important that the correct serial number is selected, as it determines which Unit Parameter Data File is loaded into the AI-80's memory. This file contains the calibration data for the AI-80 and is unique for each unit. Anytime the A.I.WorkBench software connects with an AI-80, it will read its calibration file and store it to disk. If the calibration data has never been read by the A.I.WorkBench software, contact technical support for assistance, as we can supply this file from our records.

Note that as a large amount of data is transferred over the serial port, this procedure can take a number of minutes to complete, and progress can appear slow at times.

Update Unit Parameter Data Only Action

The Unit Parameter Data file contains all the calibration data for the AI-80. If it becomes is missing the AI-80's display shows ERR2. If the data contained within the file is corrupted, the display will flash CAL. In either case, selecting the above action will restore the file.

As with the "Full System Restore" action, the unit's serial number must be selected from the drop down list box. If the serial number is not listed, then the A.I.WorkBench software has never been connected to the unit and been able to read its calibration data. In this case contact technical support for assistance, as we can supply this file from our records.

If this action fails, the AI-80 file system may be damaged, which may require a Full System Restore to correct.

Update System Program Files Only Action

This action restores the system program files to the AI-80's non-volatile memory. If these files are corrupted or damaged, the display may show ERR3. As these files provide the user interface, the front panel keys may be ineffective.

If this action fails, the AI-80 file system may be damaged, which may require a Full System Restore to correct.

Update Standard Program Files Only Action

The standard program files (P10 through P99) are factory loaded programs that provide basic Caller ID and telephony related testing programs. This action will reload all of these files into the AI-80's memory.

If this action fails, the AI-80 file system may be damaged, which may require a Full System Restore to correct.

Update System Application Only Action

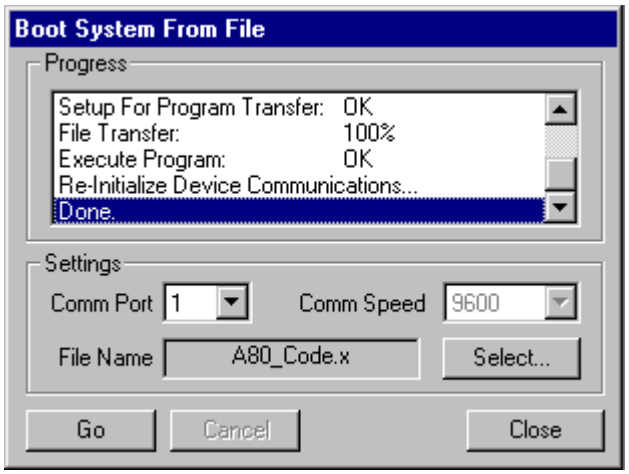
This selection will update the AI-80's primary application program. This is the main program running on the AI-80's internal processor, which allows for communication with the PC and execution of the system and standard program files. The primary appliation program stored in the AI-80's memory is overwritten with the latest copy distributed with the A.I.WorkBench software.

Note: It is not recommended to use this action to update an AI-80 with the latest software release. This is because other files may require an update at the same time, or changes in the file size can cause fragmentation in the AI-80's memory with this action. When performing compete updates of the AI-80 software, see section 4-6: Updating AI-80 Software.

Boot System From a File

The last action that may be selected from the "Restore System Files" window is "Boot System From a File". While normally at power up the AI-80 loads the primary application program from the flash memory, it is possible to load the program via the serial port. Bypassing the flash memory, the primary application program is loaded directory into the AI-80 RAM and executed. Once running the normal operations can be resumed in order to restore the program in the flash memory.

To perform the system boot, select the "Boot System From a File" action and click the mouse on the GO button. This displays the following window.



To proceed with the process of transferring the application program, follow these steps:

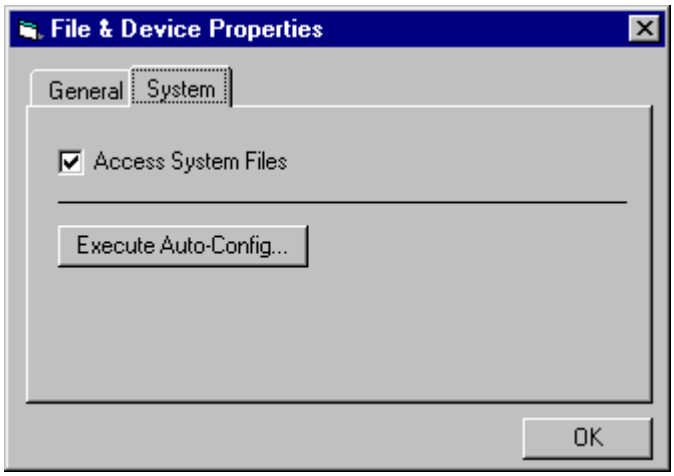
1. Click the SELECT button in order to specify the program to load into the AI-80. Choose the file named A80_Code.x and press the OPEN button.
2. Choose the communications port the AI-80 is connected to from the drop down list box.
3. Click the GO button. Then once instructed to, turn on the AI-80. This will initiate a process that loads the program file into the AI-80. Once the transfer is complete (may take a few minutes), the window should display the same information as above.

Depending on the state of other system files in the flash memory, error messages may be displayed by the AI-80. However, the A.I.WorkBench program should be able to read and modify the flash memory contents in order to replace the damaged or missing system files.

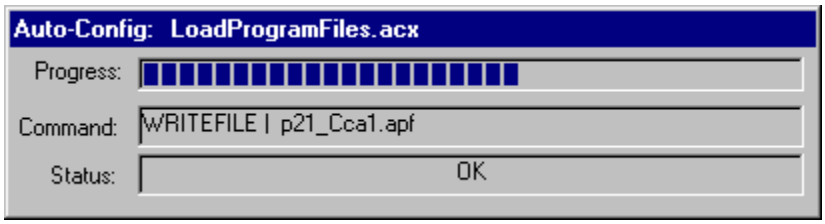
Section 4-4 Auto-Config Command Files

Auto-Config command files are shortcuts to performing significant file manipulations within the AI-80 flash memory files. If a large number of programs must be loaded into an AI-80, or its flash memory restored to factory default settings, using the auto-config commands files provides a method to automate the process. These command files contain a list of simple commands that the A.I.WorkBench follows to either read, load, or delete files from the flash memory.

To use the auto-config files, select the first entry displayed in the Flash Memory Files window. This is usually labeled “AI-80 Caller ID Generator”. Once selected, click the right mouse button, or select the [FILE] [PROPERTIES] menu command. This will display the following window.



To execute an auto-config file, select the SYSTEM tab and click the EXECUTE AUTO-CONFIG button. Once the desired command file is selected, click the OPEN button and the commands will be executed. As the file is being processed, a progress window displays the current command being executed.



Once finished, the window will be removed, and the Flash Memory File window is refreshed reflecting any changes made.

The auto-config command files are ASCII files and can be created or edited with an ASCII editor, such as the common Notepad program. The files can use five different commands to modify the flash files. Any lines in the file starting with '*' are treated as comment lines and are ignored. The supported commands are:

INITFLASH
This command requires no other parameters and completely erases the flash memory contents including all user programs and system files.

Syntax: *INITFLASH*

READFILE
This command will read the contents of the specified file ID number from the flash memory and save it to the specified file name on the host computer.

Syntax: *READFILE* | *<file ID number>* | *<file name to save to>*

WRITEFILE
This command transfers the specified file from the host computer to the flash memory. The file's ID number (used to identify it within the AI-80 flash memory files) does not need to be specified, as it is encoded in the specified file.

Syntax: *WRITEFILE* | *<file name to read from>*

WRITEAPP
This command is a special variation of the WRITEFILE command. It transfers the low level system applications to the AI-80.

Syntax: *WRITEAPP* | *<app. ID number>* | *<title>* | *<code file>*

DELETEFILE
This command deletes the specified file ID number.

Syntax: *DELETEFILE* | *<file ID number>*

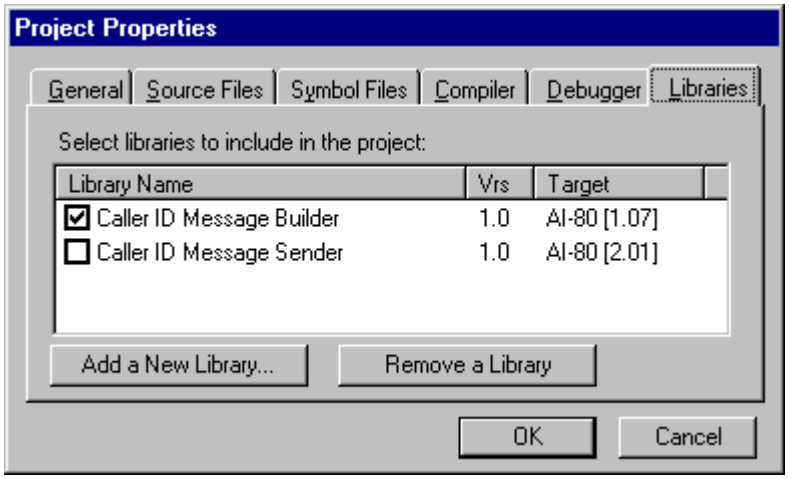
Note: Auto-Config files can be executed upon program startup by including /ACX followed by the file name on the A.I.WorkBench command line.

Section 4-5 Creating User Libraries

Libraries are collections of subroutines and functions combined into a single source file. They can simplify programming by performing common or repeatedly used operations. Basic libraries are supplied with the A.I.WorkBench software while additional libraries can be created by the user.

All of the libraries supported by the A.I.WorkBench software are listed in the Library tab of the Project Properties window. Along with the library name, its version and target information are displayed. If the box to the left of the library name is checked, then that library will be included into the current project.

As shown in the figure below, the Caller ID Message Builder library is included in the current project



The following steps outline the procedure to create a new library:

- Start a new project
- Write the library source code
- Compile and check for errors
- Write the library usage file
- Add the library to the A.I.WorkBench

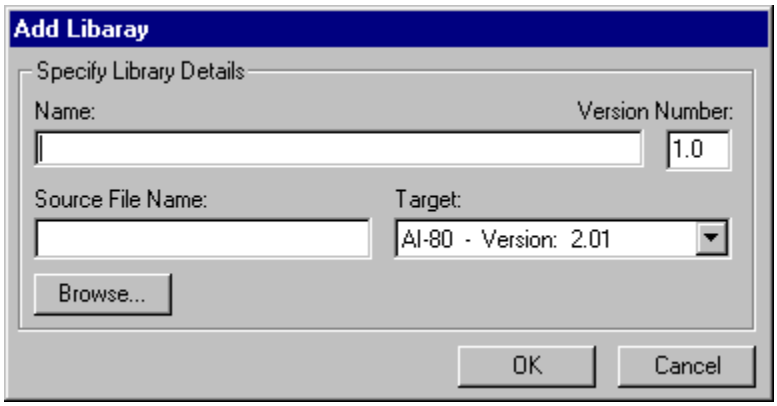
Start a New Project:
To create a user library, the first step is to create a new project by selecting the [FILE] [NEW PROJECT] menu command. The project's properties are not critical to the library, as the library is not compiled and loaded into the AI-80 as a separate file. It is important that only one source file is used in the project. All of the functions and subroutines in the library must be contained within one source file. Both the project file and source code file must be saved to the LIB sub-directory.

Write the Library Source Code:
The source code file contains all the functions and subroutines that make up the library. In order to maintain a large degree of portability, library routines should not use EXPORT or IMPORT variables to share data between processes, unless they are very clearly documented in their use. Also, as the variable data space is limited in the AI-80, the number of variables declare should be keep to a minimum. String variables consume a large amount of space and should be used sparingly.

Compile and Check for Errors:
Use the compiler to check and correct any errors in the source file.

Write the Library Usage File:
An optional usage file can be created with any ASCII file editor. This file will be displayed in the source file editor when a project uses the library. It should contain information on how to use the routines contained with the library and any additional data that is relevant. The file must have the same file name as the library source file except with the .def extension. As well, it must be located in the LIB sub-directory with the source and project files.

Add the Library to the A.I.WorkBench:
Finally, from within the Libraries tab of the Project Properties window (see above), click the “Add a New Library...” button. This shows a window similar to below.

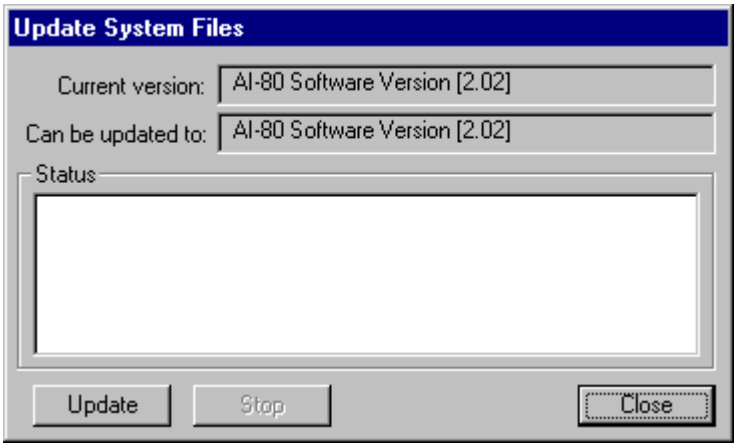


In the displayed window, all the blank fields must be filled out. The library name is the title displayed in the Project Properties window. The library version number can be used to track updates or changes to the library. Clicking the “Browse” button displays the list of source files present in the LIB sub-directory. Select the appropriate file and click the “Open” button. Lastly, the target device must be selected from the drop-down list. The target device information is used by the compiler to make sure any project using the library is compatible with the AI-80 it is being developed for.

After entering the information and clicking the OK button, the new library is added to the list in the Project Properties window.

Section 4-6 Updating the AI-80 Software

As new versions of the A.I.WorkBench software are released, they may contain updates to the AI-80's internal software. By selecting the [HELP] [UPDATE SYSTEM FILES] menu command, the AI-80's software can be updated to match the latest release. Displayed in the following figure are the AI-80's current version information along with the version the A.I.WorkBench software supports. Normally, the update should only take place if the current version number is less than the version number supported by the A.I.WorkBench software.



To start the update, click the mouse on the Update button. This starts a procedure that performs the following tasks:

- Read the AI-80's calibration data and save it
- Read all the AI-80's user programs and save them
- Re-initialize the AI-80's file system by erasing all the stored files
- Load the latest version of the primary application program
- Load the calibration data file
- Load the latest version of the system programs
- Load the latest version of the standard program files
- Load all the user program files that were previous saved

Depending on the connected baud rate, the procedure can take several minutes to complete. As the tasks are completing, the status area in the above window is updated. When the calibration data and user programs are read from the AI-80, they are stored in the following temporary directory:

<A.I.WorkBench Program Files>\projects\standard\system\tempSN<xxxxxx>

Where <xxxxxx> is the AI-80's serial number. At the end of the procedure and if no error was detected, the temporary directory is deleted along with all the files contained in it. However, if an error is detected, the update procedure is halted

and any temporary files are kept in the directory. These files may be needed to restore the AI-80.

To stop the updating procedure at any time, click the mouse on the Stop button. The current task will run to completion, at which time the user is asked to confirm aborting the update. Note that stopping the update procedure once the AI-80's flash memory has been erased will leave it in an unusable state. If this occurs, follow the procedures in section 4-3 to perform a full system restore.

Appendix A A.I. WorkBench Compiler Errors

The compiler will generate various error messages if it can not understand the syntax of any program line, or detects a mistake in the use of any identifiers. Each error message is associated with a 4 digit code. All of the possible error messages are listed here in numerical order.

Code	Error
1000	**Can't Find Target** The data files for the specified target device (Project Settings) can not be found. Check that the project settings are correct and/or the A.I.WorkBench software is of at least the same version that created the project file.
1001	**Target Version Mismatch** The specified target device is not supported by this version of the compiler. This may occur when a project is created with a compiler version that is newer than this one.
1002	**Can't Find VTP data file** The VTP data file, which contains information on the target device's command set, can not be located. A file might be missing, or it was never installed correctly.
1003	**Can't Find HAL data file** The HAL data file, which contains information on the target device's hardware abstraction layer, can not be located. A file might be missing, or it was never installed correctly.
1004	**Too Many Errors** The maximum number of compiler errors has been reached and the compilation process has stopped. This limit can be adjusted in the project settings.
1005	**Unsupported Command** A command or function used in the program is not supported by the target version. This error can occur if the project is using commands or functions only available for a newer software version. Either the offending command should be removed, or project's target version changed.
1010	**Invalid Program Line** The program line does not start with a reserved keyword and is not treated as a comment line (starts with * or ;). As such the program line in invalid.
1011	**Incorrectly Formatted Quotation** The text string used in this program line is not terminated correctly inside quotation marks. If quotation marks are used inside a text string, they must appear twice.
1012	**Unknown or Invalid Symbol** The program line contains an invalid symbol. This means that the compiler could not determine whether the line contains a keyword, identifier, numeric literal, text string, or operator. This can occur if no spaces are added between numbers and identifiers.
1020	**Statements Outside Program Block** All program statements, other than constant and variable definitions must be

	inside either a PROCESS, SUB, or FUNCTION block.

1021	**Statement Syntax Error** The syntax of the program line does not match the required form for the command. Check the structure of the command, and ensure that comments are preceded with ';'.

1022	**Invalid Data Type** The data type specified is unknown to the compiler and must be either NUMERIC or STRING to denote variables containing floating point numbers or text.

1023	**Duplicate Identifier** The declared identifier has already be declared elsewhere in the program. Variable, subroutine, and label names must be unique.

1024	**Out of Variable Space** The number of declared variables exceeds the amount of memory storage allocated to the program. Try to reduce the number of variables. Note that string variables require 16 times more space than numeric variables.

1025	**Too Many Labels** The maximum number of labels has been exceeded. These are not labels created with the LABEL command, but rather internally generated labels used by the target's interpretive processor. Under normal circumstances this error should never occur. Please contact technical support for assistance.

1026	**Invalid Register Specified** The register location specified with the IMPORT or EXPORT command is outside the valid range.

1027	**Register Location is Allocated** The register location specified with the IMPORT or EXPORT command has been previously allocated and can not be used.

1030	**Program Too Big** The compiled program's object code exceeds the maximum size limit for the target device. The program could be split into multiple programs in order to not exceed the maximum program size.

1031	**Unknown Identifier** The identifier referenced in the program line has not been declared by either of the LOCAL, GLOBAL, EXPORT, IMPORT or CONST keywords.

1032	**Unknown HOP** The specified Hardware Object Property (HOP) is unknown to the specified target device. This can occur if the HOP is either incorrectly spelled, or the project's target device does not support the specified HOP.

1033	**Can't Write to Read Only HOP** The specified HOP can not be written too. As such it may not be assigned a value by the LET command.

1034	**Can't Read from Write Only HOP** The specified HOP can not be read from. As such it may not appear inside any expressions. It can only be assigned values by using the LET command.

1035	**HOP Access Restricted** The specified HOP is restricted to system level programs. Access to the system level HOP's can be set in the Project Settings window. Incorrect use of the system level HOP's can cause unstable operation.

1036	**Can't Write to a Constant** A constant value can only be read from and not written too. As such, a defined constant can not be assigned a new value with the LET command.

1037	**Can't Write to a Parameter** Parameters used to pass values into subroutines and functions are read only. There value can not be changed and are treated as local constants.
1038	**Data Type Mismatch** The data types of operands in the expression are mismatched. This results when numeric and string operands are mixed in an expression.
1039	**Unknown Operator Symbol** An operator used in the expression is unknown.
1040	**Reserved Key Word Used** The identifier specified in the program line is the same as a reserved keyword. All identifiers must be different from the reserved keywords.
1041	**Invalid Function Usage** The function can not be used as an operand in this context. The operand must be a variable, HOP, or literal. A possible cause of this error is including a function in the parameter list of a calling subroutine.
1050	**Undefined Subroutine** The subroutine name given in the CALL statement is not defined. Check that the spelling of the subroutine name is correct, and that it has been defined.
1051	**Missing END LOOP Statement** No END LOOP statement can be found. All LOOP statements must have a matching END LOOP statement in order to define the bottom of the program loop.
1052	**Missing END PROCESS Statement** All PROCESS blocks must be terminated with the END PROCESS statement. This marks the end of a process.
1053	**Missing END IF Statement** No END IF statement can be found. All IF statements must have a matching END IF statement in order to define the end of the conditional statements.
1054	**Missing NEXT <identifier> Statement** No NEXT statement can be found. All FOR statements must have a matching NEXT statement in order to define the end of the loop.
1060	**Bad use of IF Statement** The program line's syntax does not match the requirements for the IF-THEN-ELSE command. Check the syntax and ensure that any comments after the statement start with ';'.
1061	**Bad use of END Statement** The END statement is either of incorrect syntax, or it may not be used in its present location in the program.
1062	**Bad Use of EXIT Statement** The EXIT statement is either of incorrect syntax, or it may not be used in its present location.
1063	**Bad Use of FOR Statement** The syntax of the FOR statement is incorrect. Check the syntax of the program line and ensure that any comments after the statement start with ';'.
1064	**Can't Locate Specified Label** The label specified can not be found. Any labels used must be defined with the LABEL command. Check the spelling of the specified label such that it matches the defined label.
1070	**Not Numeric Data Type**

	The variable specified must be of numeric data type. String variables are not allowed in this context.

1071	**GOTO Not Allowed Here** GOTO statements are not allowed inside a LOOP statement. If an unconditional branch is needed, use the FOR-NEXT looping structure instead.

1072	**RETURN Not Allowed Here** RETURN statements are not allowed inside a LOOP statement. If a subroutine return is needed, use the FOR-NEXT looping structure instead.

1073	**EXIT SUB, EXIT FUNC Not Allowed Here** Subroutines and functions can not be exited inside a LOOP statement. If such an exit is required, use the FOR-NEXT looping structure instead.

1074	**NEXT Without FOR** The program line contains a NEXT statement without a matching FOR statement. A FOR statement must always precede a NEXT statement.

1075	**Bad use of SELECT Statement** The syntax of the SELECT statement is invalid. Check the syntax of the statement and ensure that any comments after the program line start with ';'.

1076	**CASE Without SELECT** The CASE statement can only be used inside the SELECT structure. Check the syntax of the SELECT-CASE statements.

1080	**Bad Parameter List Syntax** The parameter list definition for the subroutine or function is incorrect. Check the syntax of the parameter list and ensure that each parameter is defined with a valid data type and are separated with commas, if more than one parameter is used.

1081	**Parameter Data Type Mismatch** The data type of a parameter passed to a subroutine or function does not match the data type in the routines definition statement.

1082	**Missing END SUB Statement** All subroutine program blocks must be terminated with the END SUB statement. This marks the end of the subroutine, and the point where program control returns to the calling statement.

1083	**Missing END FUNCTION Statement** All function program blocks must be terminated with the END FUNCTION statement. This marks the end of the function, and the point where program control returns to the calling statement.

1084	**Unknown Function** The function being calling in the expression is undeclared. Check the spelling of the function to ensure it matches the declaration.

1085	**GOSUB and RETURN Not Allowed** The GOSUB and RETURN statements can not be used inside any SUB or FUNCTION blocks.

1090	**Invalid Library Target Device** The project's target device name does not match the device name required by an included library file. Check the project settings to make sure any selected libraries match the project's target device.

1091	**Library Version Not Supported** A library used in this project requires a device target version number that is greater than specified for this project. Check the project settings to ensure that the project's target version number is at least equal to the version number of any libraries used.

Appendix B AI-80 Built-in Programs

The description and operational details of the AI-80 built-in programs are shown below. All of the source file(s) and project files for these programs are included with the A.I.WorkBench software. They are located in a sub-directory of the A.I.WorkBench files, under the \projects\standard\ directory. These programs can be modified to suit special applications, or can serve as examples in performing various telephony functions.

No. 10	Title: **Bellcore Type I - Multiple Message**	Project: p10_Bel1
Description:	Ring Port A for 2 seconds, then send a FSK Caller ID message containing the date/time, calling number, and calling name in the Bellcore Multiple Message Format.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Ring for 2 seconds at 22 Hz and 80 Vrms 2) Wait 500 msec 3) Send FSK (Bell 202 at -13 dBm (600 ohms)) 300 Preamble bits, 180 Mark bits Date & Time = 10:24 AM March 26 Calling Number = 5556789 Calling Name = John Smith	

No. 11	Title: **Bellcore Type I - Single Message**	Project: p11_Bel2
Description:	Ring Port A for 2 seconds, then send a FSK Caller ID message containing the date/time and calling number in the Bellcore Single Message Format.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Ring for 2 seconds at 22 Hz and 80 Vrms 2) Wait 500 msec 3) Send FSK (Bell 202 at -13 dBm (600 ohms)) 300 Preamble bits, 180 Mark bits Date & Time = 7:39 PM October 3 Calling Number = 5551212	

No. 12	Title: **Bellcore VMWI - Activate**	Project: p12_Bel3
Description:	Send a FSK Caller ID message containing Visual Message Waiting Indicator (Activate) message (Bellcore Multiple Message Format).	
Usage:	1) Connect CPE to port A 2) Start program	
Details:	1) Send FSK (Bell 202 at -13 dBm (600 ohms)) 300 Preamble bits, 180 Mark bits Message Type = 0x82 (VWMI) Visual Indicator Parameter = Activate	

No. 13	Title: **Bellcore VMWI - Deactivate**	Project: p13_Bel4

Description:	Send a FSK Caller ID message containing Visual Message Waiting Indicator (Deactivate) message (Bellcore Multiple Message Format).
Usage:	1) Connect CPE to port A 2) Start program
Details:	1) Send FSK (Bell 202 at -13 dBm (600 ohms)) 300 Preamble bits, 180 Mark bits Message Type = 0x82 (VWMI) Visual Indicator Parameter = Deactivate

No. 15	Title: **Bellcore Type II - Multiple Message**	Project: p15_Bel5
Description:	Send a FSK Caller ID Type II (off-hook) message containing the date/time, calling number, and calling name in the Bellcore Multiple Message Format.	
Usage:	1) Connect CPE to port A 2) Set CPE to the off-hook state 3) Start program	
Details:	1) Generate SAS tone at 440 Hz for 300 msec at 100 mVrms (unterminated) 2) Wait 25 msec 3) Generate CAS tone at 2130/2750 Hz for 80 msec at 100 mVrms (unterminated) 4) Wait for CPE to send DTMF ACK tone (time-out at 165 msec) 4) If ACK tone received, wait 100 msec and send FSK data 5) FSK Message details (Bell 202 at -13 dBm (600 ohms)) 80 Mark bits Date & Time = 10:24 AM March 26 Calling Number = 5556789 Calling Name = John Smith	

No. 20	Title: **UK BT Type I - CLIP Message**	Project: p20_Bt1
Description:	Send a Caller ID message using the UK BT signaling method. With Port A, reverse the line polarity, then send the DTAS tone followed by the FSK data (V.23). After the FSK data, ring the telephone twice.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Reverse the line polarity 2) Wait 200 msec 3) Send DTAS tone (2130/2750 Hz for 80 msec at 100 mVrms) 4) Wait 150 msec 5) Send FSK (V.23 at -14 dBV) 300 Preamble bits, 180 Mark bits Date & Time = 11:05 AM July 29 Calling Number = 071 250 7587 Calling Name = John Bull 6) Wait 500 msec 7) Ring for 700 msec seconds at 22 Hz and 80 Vrms 8) Wait 700 msec 9) Ring for 700 msec seconds at 22 Hz and 80 Vrms	

No. 21	Title: **UK CCA Type I - CLIP Message**	Project: p21_Cca1
Description:	Send a Caller ID message using the UK CCA signaling method. With Port A, briefly ring the telephone, then send the FSK data (V.23). After the FSK data, ring the telephone twice.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Ring for 350 msec seconds at 25 Hz and 60 Vrms 2) Wait 600 msec 3) Send FSK (V.23 at -14 dBV) 300 Preamble bits, 180 Mark bits Date & Time = 4:21 PM on January 31 Calling Number = 1234567890	

	Calling Name = John Bull 6) Wait 500 msec 7) Ring for 400 msec seconds at 25 Hz and 60 Vrms 8) Wait 200 msec 9) Ring for 400 msec seconds at 25 Hz and 60 Vrms

No. 22	Title: **France Type I - CLIP Message**	Project: p22_Fr1
Description:	Send a Caller ID message preceded with a short ringing burst. With Port A, briefly ring the telephone, then send the FSK data (V.23). After the FSK data, ring the telephone twice.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Ring for 250 msec seconds at 25 Hz and 70 Vrms 2) Wait 600 msec 3) Send FSK (V.23 at -13 dBm (347 mVrms unterminated)) 300 Preamble bits, 180 Mark bits Date & Time = 2:09 AM on December 15 Calling Number = 0115551234 Calling Name = John Smith 6) Wait 500 msec 7) Ring for 600 msec seconds at 25 Hz and 70 Vrms 8) Wait 400 msec 9) Ring for 600 msec seconds at 25 Hz and 70 Vrms	

No. 23	Title: **Australia Type I (Ring Burst Alert)**	Project: p23_Aus1
Description:	Send a Caller ID message preceded with a short ringing burst. With Port A, briefly ring the telephone, then send the FSK data (Bell 202 - Bellcore Multiple Message Format). After the FSK data, ring the telephone twice.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Ring for 400 msec seconds at 25 Hz and 70 Vrms 2) Wait 800 msec 3) Send FSK (Bell 202 at -13 dBm (347 mVrms unterminated)) 300 Preamble bits, 180 Mark bits Date & Time = 11:45 PM on June 7 Calling Number = 5551234 Calling Name = John Smith 6) Wait 500 msec 7) Ring for 400 msec seconds at 25 Hz and 70 Vrms 8) Wait 200 msec 9) Ring for 400 msec seconds at 25 Hz and 70 Vrms	

No. 24	Title: **Australia Type I (Line Reverse Alert)**	Project: p24_Aus2
Description:	Reverse the line polarity, then send a Caller ID message in the Bellcore Multiple Message Format (containing Date&Time, Calling Number, and Calling Name). After the message has been sent, generate ringing.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Reverse the line polarity 2) Wait 600 msec 3) Send FSK (Bell 202 at -13 dBm (347 mVrms unterminated)) 300 Preamble bits, 180 Mark bits Date & Time = 2:00 AM April 1 Calling Number = 035551111 Calling Name = Bill Jones 6) Wait 500 msec 7) Ring for 400 msec seconds at 20 Hz and 80 Vrms 8) Wait 200 msec	

	9) Ring for 400 msec seconds at 20 Hz and 80 Vrms

No. 25	Title: **China Type I (Odd Parity)**	Project: p25_Chn1
Description:	Ring Port A for 2 seconds, then send a FSK Caller ID message containing the date/time, calling number, and calling name using odd parity encoding for the parameter data fields.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Ring for 2 seconds at 22 Hz and 80 Vrms 2) Wait 500 msec 3) Send FSK (Bell 202 at -13 dBm (600 ohms)) 300 Preamble bits, 180 Mark bits Date & Time = 5:25 PM on August 17 Calling Number = 81081338 Calling Name = Charley Heung	

No. 30	Title: **Japan NTT Type I**	Project: p30_Ntt1
Description:	Send a Caller ID Message using the Japanese NTT Type I signaling method. This consists of reversing the line polarity, then generating up to 6 seconds of CAR (CPE Alerting Ring). If the CPE goes off-hook during the CAR, send the FSK data. Then wait for the CPE to go back on-hook and ring the CPE normally.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Reverse the line polarity 2) Wait 500 msec 3) Generate CAR (17.5 Hz, 75 Vrms, 500 msec on, 500 msec off) 4) If CPE does not go off-hook within 6 seconds, goto step 9 5) If CPE goes off-hook, then... 6) Wait 1 second 7) Send FSK (V.23 at -13 dBm (600 ohms)) NTT Type 1 message format Calling Number = 5551212 8) Wait for CPE to go on-hook (within 7 seconds) 9) Ring CPE at 17.5 Hz, 75 Vrms, for 1 second	

No. 40	Title: **DTMF Caller ID (Line Reverse Alert)**	Project: p40_Mfd1
Description:	Send a DTMF based Caller ID number by first reversing the line polarity, then sending the calling number using DTMF digits, followed by ringing.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	
Details:	1) Reverse the line polarity 2) Wait 300 msec 3) Send DTMF Caller ID message (DTMF level of 300 mVrms, on/off time of 70 msec) Start code = D, Calling Number = 7132920, Stop Code = C 4) Wait 500 msec 5) Ring for 600 msec seconds at 20 Hz and 60 Vrms 6) Wait 600 msec 7) Ring for 600 msec seconds at 20 Hz and 60 Vrms	

No. 41	Title: **DTMF Caller ID (Ring Burst Alert)**	Project: p41_Mfd2
Description:	Send a DTMF based Caller ID number by first generating a short ringing burst, then sending the calling number using DTMF digits, followed by normal ringing.	
Usage:	1) Connect CPE to port A 2) Set CPE to the on-hook state 3) Start program	

Details:	1) Ring at 22 Hz, 60 Vrms, for 500 msec 2) Wait 500 msec 3) Send DTMF Caller ID message (DTMF level of 300 mVrms, on/off time of 70 msec) Start code = D, Calling Number = 7132920, Stop Code = C 4) Wait 500 msec 5) Ring for 600 msec seconds at 22 Hz and 60 Vrms 6) Wait 600 msec 7) Ring for 600 msec seconds at 22 Hz and 60 Vrms

No. 50	Title: **Dial Tone Generation**	Project: p50_dt
Description:	When a CPE on Port A goes off-hook, wait 200 msec and generate a dial tone signal (350/440 Hz) until the CPE goes back on-hook.	
Usage:	1) Connect CPE to port A 2) Start the program 3) Going off-hook starts the dial tone 4) Going on-hook stops the dial tone 5) The program will continue to run until the Stop key is pressed	
Details:	1) Wait till the CPE goes off-hook 2) Wait 200 msec 3) Generate dial tone (350 and 440 Hz at 347 mVrms (unterminated)) 4) Wait till the CPE goes on-hook 5) Stop the dial tone 6) Repeat	

No. 51	Title: **Stutter Dial Tone Generation**	Project: p51_sdt
Description:	When a CPE on Port A goes off-hook, wait 200 msec and generate a stutter dial tone signal (350/440 Hz) until the CPE goes back on-hook.	
Usage:	1) Connect CPE to port A 2) Start the program 3) Going off-hook starts the stutter dial tone 4) Going on-hook stops the dial tone 5) The program will continue to run until the Stop key is pressed	
Details:	1) Wait till the CPE goes off-hook 2) Wait 200 msec 3) Generate stutter dial tone 350 and 440 Hz at 347 mVrms (unterminated), 10 pulses, 100 msec on/off time after the 10th pulse, generate continuous dial tone 4) Wait till the CPE goes on-hook 5) Stop the dial tone 6) Repeat	

No. 60	Title: **Measure Flash Timing**	Project: p60_flsh
Description:	Measure and display the duration of any CPE line flashes. The program measures the time from when the CPE went on-hook to off-hook. The time interval is displayed in milliseconds up to a maximum of 9999 msec. While the CPE is on-hook, the display will show “F”.	
Usage:	1) Connect CPE to port A 2) Start the program 3) Flash the CPE’s hook switch (display will show “F” while on-hook) 4) Once the CPE goes off-hook, the flash time is displayed in msec 5) The program will continue to run until the Stop key is pressed	
Details:	1) Zero the on-hook and off-hook timers 2) Wait for the CPE to go on-hook 3) Display “F” 4) Wait till the CPE goes off-hook 5) Display the flash time 6) Repeat	

No. 61	Title: **Measure Pulse Dialing PPS**	Project: p61_pps
Description:	Measure and display the average PPS reading for a CPE performing pulse dialing. The program will calculate the average pulses per second (PPS) for any digit dialed (greater than 1). No checking for valid make and break times is performed, only the average PPS is reported	
Usage:	1) Connect CPE to port A 2) Start the program 3) Pulse dial any digit from 0 to 9 4) Once the dialing is complete, the average PPS is displayed. 5) The program will continue to run until the Stop key is pressed	
Details:	1) Wait for the CPE to go on-hook (record the timer value) Marks the start of the first digit 2) Wait for up to 700 msec for the start of another digit 3) If another digit is detected, increment the digit count displayed 4) Once all the digits have been dialed, calculate and display the PPS 5) Repeat	

No. 62	Title: **Measure Make and Break Times**	Project: p62_mbr
Description:	Measure and display the average pulse dialing make time and break time. The program will calculate the average make and break times for any digit dialed. Note that no checking of minimum and maximum make/break times is performed. Only the average is calculated.	
Usage:	1) Connect CPE to port A 2) Start the program 3) Pulse dial any digit from 0 to 9 4) Once the dialing is complete, the average make time is displayed for 1 second 5) Then the break time is displayed for 1 second 6) Additional digits can be dialed at any time 7) The program will continue to run until the Stop key is pressed	
Details:	1) On program start, wait for the CPE to go off-hook 2) Wait for the CPE to go on-hook 3) If on-hook for more than 200 msec, then restart program 4) If on-hook for less than 200 msec, record on-hook time (break time) 5) Record the make time when the next digit starts 6) Repeat 3 to 5 until all digits have been received 7) Display the break time (in msec) for 1 second 8) Display the make time (in msec) for 1 second 9) Repeat if more digits are dialed	

Appendix C AI-80 Error Messages & Codes

System Power-Up Error Codes

Once turned on, the AI-80 undergoes a number of steps to check its internal systems and load its system level software. In the event an error is detected, it will be displayed on the front panel. The possible error codes and there meaning are:

Err1 Missing System Program
This error occurs if the AI-80 can not locate the primary system program within the non-volatile flash memory. This can happen if the flash memory has become corrupted. Without these files the AI-80 can not establish normal communications with the A.I.WorkBench software. To resolve this program, the program must be re-loaded into the flash memory. This is done by first booting the AI-80 via a serial connection from the host PC and then writing the missing file to the flash memory. See sections 4-3 and 4-4 for more information.

Err2 Missing Unit Parameter Data File
If the parameter data file is missing or corrupted within the flash memory, this error is displayed. This file contains various factory calibration settings and is required for proper operations. It can be re-loaded into the flash memory using the A.I.WorkBench software. If a copy of the unit's parameter file is not available, contact technical support for assistance.

CAL Invalid or Missing Calibration Factors
If the parameter data file is located within the flash memory, but the calibration factors are either missing or invalid, the display will show 'CAL' and flash the Start LED. The AI-80 will be suspended until the Start key is pressed. Once the Start key is pressed, normal operation will resume; however, the errors within the parameter data file should be corrected to ensure proper operation

Err3 Missing Program Boot File
One of the last steps in the AI-80's startup procedure is to begin execution of the Boot program (program ID number 1). If this file is missing from the flash memory, the error message will be displayed. Without the Boot program the System Launcher program (which manages the user programs and implements the user interface) will not be started. Though the AI-80 will be able to communicate with A.I.WorkBench software, which allows the Boot program to be re-written into the flash memory.

Note: During the operation of the AI-80, on additional error may be displayed on the front panel as **Err**. This represents an internal stack fault, which is fatal to the AI-80's operation. In this case, the AI-80 will be halted from executing any user programs and must be turned off and on to reload its system software. If this error occurs, please contact technical support for assistance.

Functional Check Error Codes

The AI-80 can be forced to execute a functional check of itself by holding down the Pause key when the unit it first turned on. The Pause key must be held for approximately two seconds, until a second beep is heard. Releasing the key will then start the functional check.

If the AI-80 reports a failure, it will display both an error code and its measurement value that failed. The display alternates between the error code (shown with a leading 'F' character) and the measurement value. Since the AI-80 is operating outside its specified limits, it should be returned for repair or calibration.

See Section 1, Functional Check of the AI-80 for more information on performing the check.

Error Code	Description of Error
F 1	Unable to verify CPE on-hook and Tel. Line on-hook state.
F 2	Unable to verify proper tone generator output level on the telephone line in the on-hook state with a normal polarity setting.
F 3	Unable to verify proper tone generator output level on the telephone line in the on-hook state with a reversed polarity settings.
F 4	Unable to verify proper ringing level on the telephone line with a reversed polarity setting.
F 5	Unable to verify proper ringing level on the telephone line with a normal polarity setting.
F 6	Unable to detect the CPE load off-hook state.
F 7	Unable to verify proper tone generator output level on the telephone line in the off-hook state with normal polarity as measured by the CPE load.
F 8	Unable to verify proper operation of the telephone line OSI condition.
F 9	Unable to verify proper tone generator output level on the telephone line in the off-hook state with normal polarity as measured by the Telephone Interface.
F 10	Unable to verify proper tone generator output level on the telephone line in the off-hook state with reversed polarity as measured by the CPE load.
F 11	Unable to verify residual noise level present at the telephone interface.
F 12	Unable to verify the CPE load hook switch status in the on-hook state
F 13	Unable to verify the CPE load hook switch status in the off-hook state
F 14	Unable to verify the Telephone Interface transient suppression circuitry during the reception of DTMF tones.

Program Execution Error Codes

Programs developed with the A.I.WorkBench software execute within the AI-80's native language built-in interpreter. Though the compiler is responsible for generating error free object code, it is possible that the compiler uses commands or structures unknown to the AI-80. This can occur if the AI-80 software version does not match the compiler's device target setting. The error code of any detected fault for the interpreted programs are displayed in the Processor Status window.

Since these are internal errors to the AI-80 and not compiler source code listings, they should not occur. If any of the following errors are reported, please contact technical support for assistance.

Error Type	Code	Description
Data Reference	501	Invalid reference location (must be either H,G,V, or I)
"	502	Invalid reference data type specified.
"	503	Invalid encoded reference (internal reference coding error)
"	504	Invalid global/local data register location specified
"	505	Invalid VTP register number or VTP data type mismatch
VTP Error	1000	Program counter exceed program length (no program end command detected)
"	1001	Unknown program command or bad command syntax
"	1002	Reference data type mismatch
"	1003	Specified label is an illegal value
"	1004	Can't find the specified label
"	1005	VTP stack underflow (no data to pop)
"	1006	VTP stack overflow (no room to push)
"	1007	Data type mismatch with data popped from stack
"	1008	Bad return program address popped from the stack
"	1009	Illegal VTP number specified with task control commands
HOP	10xxxx	Set HOP. Invalid HOP ID number
"	11xxxx	Set HOP. Invalid source data address
"	12xxxx	Set HOP. Can't write to a read only HOP
"	13xxxx	Set HOP. Data type mismatch
"	15xxxx	Get HOP. Invalid HOP ID number
"	16xxxx	Get HOP. Invalid destination data address
"	17xxxx	Get HOP. Can't read from a write only HOP
"	18xxxx	Get HOP. Data type mismatch
		xxxx = HOP number passed to the Set/Get HOP command

Command strings used by the A.I.WorkBench software to control the AI-80 may also return error messages due to version mismatches. The following table lists the various error codes that can be returned by the AI-80. These error messages should not normally appear. If they are displayed, please contact technical support for assistance.

Error Type	Number	Description
Command Interpreter	100	Unknown CSI command
"	101	Can't find the "=" with the ">" (set) command

“	102	No value specified with the “>“ (set) command
“	110	Invalid sector specified with the flash commands
“	111	Invalid address specified with the flash commands
“	112	Invalid byte count specified with the flash commands
“	113	No “|” character found with the flash commands
“	114	Invalid data with the flash write command
“	115	Flash device program fail with flash save command
“	116	Flash device erase fail with flash erase command
“	120	Bad VTP number specified with program commands
“	121	Invalid VTP program start command format
“	122	Can’t find file or invalid memory program run command

Appendix D Host Serial Communications Check

Normally, the A.I.WorkBench program will scan for any AI-80 connected to the host PC at program start. However under some circumstances the program may be unable to locate and communicate with an AI-80. RS-232 serial communications can be prone to problems, due to the large number of settings that must match between the DTE and DCE before communication can proceed.

The AI-80 was designed to simplify many of the problems that can occur with RS-232 serial links. Usually, the A.I.WorkBench fails to connect to the AI-80 because of these two problems:

- Cross over serial cable was used instead of a straight-through cable.
- Communications port is unavailable or incorrectly setup within the Windows operating system.

The AI-80 Host Serial port on the rear panel is configured as a standard 9 pin female DCE (Data Communications Equipment) interface. The connection from the PC can be either a 9 pin or 25 pin male DTE (Data Terminal Equipment) interface. If a 9 pin port is available on the PC, connect the supplied 9 pin cable between the AI-80 and the host PC. If only a 25 pin port is available, attach a 25 pin to 9 pin adapter (not supplied) to the PC, before connecting the supplied 9 pin cable to the AI-80 and PC. Since the AI-80 is configured as a DCE (Data Communications Equipment), a “straight-through” serial cable is required. Do not use a “null-modem” or “cross-over” cable when connecting the PC to the AI-80.

The serial port used on the PC must be recognized and configured by the Windows 95/98 operating system as either COM 1, COM 2, COM 3, or COM 4.

The AI-80 serial data transmission settings are fixed at 8 data bits, with no parity, and 1 stop bit. The Request-To-Send (RTS) and Clear-To-Send (CTS) signals have no effect on communications, as do Data-Terminal-Ready (DTR) and Data-Set-Ready (DSR). The baud rate of the AI-80 is adjustable from a default (power up) setting of 9600 bps to 115200 bps. If a break signal is received, the AI-80 will immediately return the baud rate to its default value.

If the A.I.WorkBench is unable to connect with the AI-80, follow these steps in order to help isolate where the problem is located.

1. Turn off the AI-80.
2. Turn on the AI-80 and hold down the Minus key until the display shows four decimal points: “. . . .”. This forces the AI-80 to a baud rate of 9600 and it will echo every byte it receives from the PC.
3. Connect a serial cable from the host PC to the AI-80.
4. Run the Windows 95/98 HyperTerminal Program (or any other terminal program).
   a) This is done by pressing the START button on the Task bar, followed by PROGRAMS > ACCESSORIES > HYPERTERMINAL.

b) Double click the “Hypertrm.exe” Icon
c) Enter a name for the connection. I.E. Test
d) Set the “Connect Using” drop-down list box to be the serial port number connected to the AI-80. I.E. for port number 1, select “Direct to Com 1”.
e) Press OK, and set the following in the next window:
Bits per second = 9600
Data bits = 8
Parity = None
Stop bits = 1
Flow Control = None
f) Press OK.

5. If all the settings are correct, the AI-80 will echo back all characters typed. Thus any characters typed will be displayed in the terminal window. The A.I.WorkBench will then be able to connect to the AI-80, provided, no other program (like the HyperTerminal program) is using the communications port. If no characters are displayed, then check the following...
a) If the COMM LED illuminates on the AI-80 when characters are typed, then the correct port has been selected. Verify that the baud rate, parity, and stop bits are correct. Use the [FILE] [PROPERTIES] menu command to change the terminal settings.
b) If the COMM LED does not illuminate, then repeat using a different port number. Use the [FILE] [PROPERTIES] menu command to change the terminal settings.
c) If none of the ports work, verify that the cable is a straight-through type and not a cross-over cable. Also verify that the communications port is reported as working properly in the Windows Control Panel.

Appendix E General Specifications

AI-80 Telephone Line Interface

Tone Generator

Output Level [*1]	0 to 2.0 Vrms +/- 0.5 dB
Frequency Range	50 Hz to 10 kHz
Flatness	100 Hz to 5 kHz +/- 0.75 dB
THD+N	0.09% C-message (1 kHz)
Harmonic Distortion	> 65 dBc
Frequency Accuracy	0.015%

FSK Generator

Output Level [*1]	0 to 2.0 Vrms +/- 0.5 dB
Flatness	+/- 0.75 dB
Mark & Space Frequency	100 Hz to 5 kHz
Frequency Accuracy	0.015%
Baud Rate	100 Hz to 5 kHz
Baud Rate Accuracy	0.015%

Noise Generator

Output Level [*1]	0 to 1.0 Vrms +/- 0.75 dB

Ring Generator

Output Level	0 Vrms to 80 Vrms.
Frequency Range	10 Hz to 100 Hz
Flatness	+/- 0.3 dB
THD+N	0.1% (22 Hz)
Frequency Accuracy	0.015%
Ringer Load	5 REN

Telephone Line

Output Impedance	600 or 900 ohms +/- 5% (200 Hz to 4 kHz)
Loop Voltage	48 Volts +/- 2V
Loop Current	26 or 45 mA +/- 15%

Level Meter

Level Accuracy	+/- 0.3 dB @ 1 kHz
Frequency Range	10 Hz to 10 kHz
Flatness	100 Hz to 5 kHz +/- 0.2 dB
Maximum Input	4.0 Vrms
Residual Noise	<-60 dBmC

[*1] Maximum output level with an unterminated or terminated telephone line

AI-80 CPE Load Interface

DC Characteristics

On-hook Impedance	> 1 Meg-ohm (0 to 200 Volts)
Off-hook Impedance	230 ohms +/- 10% (at 26 mA)
Maximum Loop Current	100 mA

AC Characteristics

On-hook Impedance	> 0.5 Meg-ohm (0 to 200 Volts, 10 Hz to 5 kHz)
Off-hook Impedance	600 ohms +/- 10% (1 kHz)

Level Meter

Level Accuracy	+/- 0.3 dB @ 1 kHz
Frequency Range	10 Hz to 10 kHz
Flatness	100 Hz to 5 kHz +/- 0.5 dB
Maximum Input	4.0 Vrms (normal gain) 150 Vrms (low gain)
Residual Noise	<-60 dBmC

AI-80 Optional Complex & External Impedance

Telephone Line Impedance

Complex #1 [*2]	220 ohm + (820 ohm || 115 nF) +/- 5%
Complex #2 [*2]	270 ohm + (750 ohm || 150 nF) +/- 5%
Complex #3 [*2]	370 ohm + (620 ohm || 310 nF) +/- 5%

[*2] Only one complex impedance can be installed in the AI-80 and it must be installed at the time of order.

AI-80 Optional I/O Module

Analog Output

Output Level [*3]	0 to 2.0 Vrms +/- 0.75 dB
Output Impedance	600 ohms +/- 5%
THD+N	0.09% C-message (1 kHz)
Harmonic Distortion	> 65 dBc
Signal Source	Internal Tone Generators, or Telephone Interface Monitor, or CPE Load Monitor

Analog Input

Maximum Input	0 to 2.0 Vrms
Input Impedance	100 kohms +/- 10%
Level Accuracy	+/- 0.75 dB
Residual Noise	<-60 dBmC

DC Voltage Measurement

Max. Differential Input	+/-200 VDC
Max. Common Mode	+/-200 VDC (w.r.t. earth ground)
Input Impedance	> 1 Meg Ohm
Accuracy	+/- 1% (0.2 to 200 Volts)
CMRR (DC)	> 60 dB

Analog Input Channels [*4]

Input Voltage Range	-10.0 Volts to +10.0 Volts
Input Impedance	> 100 kOhm
Accuracy	+/- 1% (1.0 to 10.0 Volts)

Digital I/O

Logic Levels	5 Volt TTL Compatible
Fixed Digital Outputs	8 bits
Fixed Digital Inputs	8 bits
Digital Input or Output	7 bits
Timer Output (PWM mode)	10 bit, 1.8 kHz rep. rate
Pulse Timing Measure [*5]	0 to 100 s +/- 0.2 msec

		+ve pulse, -ve pulse, rising or falling edge
Pulse Counter Input	*5	0 to 500 kHz
Gating Time		0.001 to 100 s +/- 0.2 msec
Pulse Generator		0.001 to 100 s +/- 0.2 msec
Async Serial Comm	*5	1200, 2400, 9600, 19200 baud 8 data bits no parity, 7 data bits odd parity, or 7 data bits even parity

*3 Level accuracy w.r.t. internal tone generators
*4 Analog input channels shared with digital inputs
*5 Function shared with digital input/output channels

Appendix F Technical Support

If you encounter problems while using the AI-80 or the A.I. WorkBench software, please contact us so that we can provide assistance. You may reach us in any one of the following manners:

Email: techsupport@adventinst.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) Limited
Unit No. 7, 9/F, Shatin Galleria
18 - 24 Shan Mei Street
Fotan, Shatin, N.T.,
Hong Kong

Software updates and future releases for the AI-80 and A.I.WorkBench software are available on our website at: *www.adventinst.com*