FAQ's for Controlling an AI-80

Table of Contents

1. Does a Windows DLL or driver exist for the AI-80? 1
2. How do I talk to an AI-80? 1
3. How do I control the AI-80? 2
4. How do I write to or read from the AI-80's properties? 2
5. How can data be exchanged between the AI-80's hardware settings and programs created with A.I.WorkBench? 3
6. What are the A.I.WorkBench variable types? 5
7. How do I control the programs generated by the A.I.WorkBench compiler? 5
8. Is it possible to write to the AI-80's flash memory? 7
9. Can the AI-80 decode Bell 202 or V.23 FSK signals? 7
10. How do I change the AI-80 RS-232 port baud rate? 7
11. The AI-80 returned an error code in response to a command. What does the number mean? 8
12. How can I get the status of any running programs? 9
13. Is there a summary list of all the AI-80 hardware properties? 9

1. Does a Windows DLL or driver exist for the AI-80?

No. Controlling an AI-80 is performed directly by sending commands from the PC's RS-232 serial communications port.

2. How do I talk to an AI-80?

By sending commands and receiving data over its RS-232 serial port. The serial port operates at one of the following baud rates: 9600, 19200, 38400, 57600, 115200. The serial format is fixed at 8 data bits, 1 stop bit, and no parity. The baud rate at power up is 9600 bps and can be changed by sending a command. If the AI-80 detects a line break (at least 11 consecutive space bits), it resets the baud rate back to the default value of 9600 bps. Only three connections to the 9 pin connector are required. They are TX data, RX data, and Ground. The AI-80 can sense the status of the RTS signal and set the CTS signal as either active or in-active. While the RTS and CTS signals are not required for communication, they can be utilized by programs running on the PC and AI-80 for controlling data flow or other purposes.

All commands sent to and data received from an AI-80 use ASCII characters. An AI-80 command is simply a series of ASCII characters followed by a <CR> character (ASCII 13). Optionally, a <LF> character (ASCII 10) may follow the <CR> character. The AI-80 ignores the trailing <LF> character. For every command sent, the AI-80 responds with either an acknowledgement, error code, or the requested data. Because ASCII characters are used for the commands, any terminal program (i.e. Window's HyperTerminal) may be used to send commands to the AI-80. Please note that the AI-80

does not echo any sent characters. In addition, all responses sent by the AI-80 are terminated with a <CR> character.

Please note that AI-80 commands are case sensitive. All commands use only upper case characters.

3. How do I control the AI-80?

In general there are two methods used to control the AI-80 via the RS-232 port. The first is to send very simple commands to the AI-80 that control its various tone generators and telephone interface settings. The AI-80 has approximately 170 different "properties" that can be read or written to. Each property controls a different aspect of the AI-80. For example if you want to change the frequency of a tone generator, you write to its frequency property. Or if you want to read the signal level on the telephone line, you can read the level meter's property.

The second method is to write a program for the AI-80 and let the program control the AI-80 properties. If you have used the A.I.WorkBench software included with the AI-80, then you know how to write AI-80 programs using a simplified high level language. The programs can be either stored in the AI-80's flash memory or loaded into the AI-80's RAM via the RS-232 port. Then commands can be sent from the PC to start or stop these programs. In addition, any executing program has the ability to send or receive data from the serial port directly.

For timing critical operations it is usually better to create an AI-80 program, load it into either flash or RAM, and then control it from the PC. This eliminates any timing uncertainties causes by communication delays.

4. How do I write to or read from the AI-80's properties?

To control the AI-80 hardware settings via the RS-232 serial port, use the Set Data ">" command. This command has the following syntax:

>(ref)=(data)

where "(ref)" represents the hardware setting to change and "(data)" is the new value for the hardware property. The syntax for the "(ref)" is as follows:

H(data type)(id)

where (data type) is either "N" or "S" depending of whether or not the hardware settings requires a numeric or string value. The "(id)" value represents an integer number identifying the specific property setting to change.

As an example, to set tone generator B to 1.23kHz at an output level of 0.413 Vrms, type the following three Set Data commands using any terminal program:

>HN50=1230***<CR>[<LF>]****
>HN51=0.413***<CR>[<LF>]****
>HN52=1***<CR>[<LF>]****

(*) Each commands ends with a carriage return characters (ASCII 13) and optionally a line feed characters (ASCII 10).

After sending each command, the AI-80 should respond with "OK".

The first command writes the value 1230 to hardware property setting number 50 (Tone Generator B frequency). The second command writes 0.413 to hardware property setting number 51 (Tone Generator B level). Finally the third command writes the value 1 to hardware property setting number 52 (Tone Generator B enable). This causes the AI-80 to output a 1.23 kHz tone at 0.413 Vrms from the telephone interface port.

To query hardware settings from the AI-80, use the Get Data command “?”. For example, to read the frequency setting for tone generator B, type the following command:

?HN50***<CR>[<LF>]***

This causes the AI-80 to return the string “1.23e3”. All values returned by the AI-80 are in scientific notation.

Note: When specifying numeric values, they must be in the following format:
[-]n[.n] where: n is 1 or more digits 0 to 9
[ ] indicates optional

A description of each hardware property can be found in the AI-80 reference manual. For a listing of all the property numbers, see question: 13 "Is there a summary list of all the AI-80 hardware properties?"

5. How can data be exchanged between the AI-80's hardware settings and programs created with A.I.WorkBench?

From the PC's point of view, the AI-80 looks like a large collection of registers. These registers control the operation of the AI-80's hardware properties. Additional registers control the execution of the programs compiled by the A.I.WorkBench software and are used to hold the data stored in the program's declared variables. The registers hold either 32 bit floating point numbers, or up to a 64 character string. Access to these registers vary. Most have read and write access, while some are read only, and others are write only.

When the PC wants to read a register is must use the get data command "?".

For example, to read the level meter on the AI-80, the PC should send:

?HN86***<CR>[<LF>]***

The first letter "H" means hardware property register, while the second letter "N" means numeric value. The number 86 is the register number of the AI-80 level meter.

To read the AI-80 software version string, the PC can send the following command:

?HS2***<CR>[<LF>]***

In this case, the letter "S" means a string register is being read, and the register number is 2.

There are over 170 different hardware related registers that are used to control the AI-80. For example, when you create a program using A.I.WorkBench to start the ring generator:

```
Let RING.LEVEL = 80
Let RING.FREQ = 22
Let RING.ENABLE = 1
```

The three commands are used to set register HN48 (level) to 80, HN47 (frequency) to 22, and HN49 (enable) to 1.

You can read back the ring generator settings by sending the command:

?HN48:?HN47:?HN49***<CR>[<LF>]***

to the AI-80 from the PC. Note that the colon character ':' can be used to send multiple commands to the AI-80 at the same time. The three responses from the AI-80 are also sent back as one text line, with each response separated by the colon character.

In situations were you require the PC to pass information back and forth to a A.I.WorkBench program, the simplest method is to use the variable registers.

If you declare variables in your AI-80 script program using the EXPORT or IMPORT keywords, then they can be used to exchange data with the PC. The following A.I.WorkBench program sets a declared variable to a value of 3.14.

```
Export  Numeric  MyVariableName  1
Let  MyVariableName = 3.14
```

To read this variable from the PC, use the following command:

?GN10001***<CR>[<LF>]***

The first letter "G" refers to general variable register, while the second letter "N" means you want to read a numeric value. A special block of 300 registers is reserved for all script variables declared with the EXPORT or IMPORT key words. The PC can read or write to any of these variables by adding an offset of 10000 to the register number.

For example, if the PC sends:

>GN10001=43.7***<CR>[<LF>]***

It sets register #1 to 43.7. A program can read this value and use it for various purposes. The following example sets the ring generator frequency to the value stored in the variable.

```
Import Numeric MyVariableName  1
Let RING.FREQ = MyVariableName
```

The program reads the value of 43.7 and sets the ring generator frequency to that value.

Using registers is simplest method to exchange data between the AI-80 and the PC. It keeps the PC as the communications master by always initiating commands. A second method exists where an AI-80 program can directly send back data to the PC via the RS-232 port. However special precautions must be observed to prevent collisions between any data the AI-80 sends and any responses the AI-80 automatically generates due to receiving a command from the PC.

Note: It is also possible for the PC to read the status of the A.I.WorkBench program (if it is running, paused, or stopped). This information is stored in registers VN103, VN203, VN303, and VN403.

6. What are the A.I.WorkBench variable types?

The AI-80 can execute up to four program processes simultaneously. Each process has a private data pool from which variables are allocated. In addition, all of the processes can access a common data pool. The common data pool is normally used to share data between the processes and also the PC.

In a script program you can declare four different types of variables. They are: LOCAL, GLOBAL, IMPORT, EXPORT.

The first two (LOCAL and GLOBAL) are always allocated in the process's private data pool. Though possible, it is not recommend that the PC read and write directly to these variables. This is because they may change location every time the program is compiled within A.I.WorkBench.

The last two (IMPORT and EXPORT) are always allocated in the common data pool. Since these variables are meant for sharing data between processes or the PC, the compiler must be told which register number to used. The syntax for these two variable types are:

Import <vartype> <varname> <register number>
Export <vartype> <varname> <register number>

Where the register number is from 1 to 300.

7. How do I control the programs generated by the A.I.WorkBench compiler?

In the A.I.WorkBench software, you can write programs using high level statements like LOOP, IF-THEN-ELSE, FOR-NEXT, and SELECT. These high level statements are compiled into a much simpler instruction set that the AI-80 executes. When you compile a program using the A.I.WorkBench software, it creates a number of output files. One of these files has a .obc extension. This file contains the instructions that the AI-80 executes. It is a complex string of ASCII characters that allows the AI-80 to perform the high level statements.

A brief summary of the commands used to control AI-80 programs are as follows:

1. Clear Program Memory: PC

Send this command before loading any programs into the AI-80's memory.

ie. PC***<CR>[<LF>]***

2. Load Program: PL"(program)"
Loads a program into the AI-80's RAM. The program must be enclosed in quotation marks. Note that if the program itself contains quotation marks, then they must be "doubled up". For example, to load the following program:

TIS"hello"GS1

The command syntax is:

PL"TIS""hello""GS1"***<CR>[<LF>]***

If programs are longer than 128 characters, they should be broken up into text strings no longer than 128 characters. Then load each text string in sequence by using the "PL" command.

The maximum program size that can be loaded into the AI-80's RAM is 16384 bytes.

3. Start Program Execution from RAM: PS(n)M
This command starts executing the program loaded into RAM. The value (n) must range from 1 to 4. This value specifies which "virtual" processor is used to execute the program. The AI-80 can run up to 4 programs at the same time. Each program executes on a virtual processor identified with a number from 1 to 4.

ie. Run program loaded into RAM on virtual processor #1

Send: PS1M***<CR>[<LF>]***

4. Start Program Execution from Flash Memory: PS(n)F(file number)
This commands starts executing a program from flash memory specified by the (file number) parameter. As with the above command, the value (n) specifies which processor use (valid range is from 1 to 4). Programs files are loaded into the flash memory by using the A.I.WorkBench software.

ie. Run program #500 stored in flash memory on virtual processor #2

Send: PS2F500***<CR>[<LF>]***

5. Halt Program Execution: PH(n)
Suspends program execution. The value (n) specifies which virtual processor. If n=0, then all active processors are suspended.

ie. Halt virtual processor #1

Send: PH1***<CR>[<LF>]***

6. Resume Program Execution: PR(n)
Resumes program execution. The value (n) specifies which virtual processor. If n=0, then all halted processors resume program execution.

ie. Resume programming running on virtual processor #1

Send: PR1***<CR>[<LF>]***

7. Stop Program Execution: PX(n)
Stops program execution. The value (n) specifies which virtual processor. If n=0, then all halted processors are stopped.

ie. Stop all processors

Send: PX0***<CR>[<LF>]***

8. Is it possible to write to the AI-80's flash memory?

While the AI-80 can read and execute programs from flash memory, it can not write programs to the flash memory by itself. It does support very low level commands for writing to the flash on a byte-by-byte level. The A.I.WorkBench software uses these commands to manage the directory structure and allocates space within the flash memory. It can be quite complex to perform this task manually. It is not recommend to attempt to write to the flash memory in the AI-80, as any mistakes may render the file structure unreadable.

The A.I.WorkBench software should be used to write programs into the flash memory. To automate tasks, you can create a simple file list that the A.I.WorkBench software uses to "batch" load multiple programs. In addition, this file list can be executed when the A.I.WorkBench starts if it is included in the command line.

9. Can the AI-80 decode Bell 202 or V.23 FSK signals?

An FSK decoder is an option for the AI-80. Unless the option is enabled, the AI-80 is unable to decode FSK signals. The exception to this is when using the TRsSim PC software. The TRsSim software automatically enables the FSK decoder in the AI-80 so that it can be used to perform SMS testing.

If an AI-80 has the FSK decoder option enabled, it will display "SO 1" briefly upon power up. To enable this option, a software key must be entered into the AI-80 by using the A.I.WorkBench software. Once this key is entered and the AI-80 reset, the FSK decoder will be activated.

10. How do I change the AI-80 RS-232 port baud rate?

On power up or reset, the AI-80's serial port is configured to 9600 baud, 8 data bits, 1 stop bit, no parity. The baud rate can be changed to either 19200, 38400, 57600, and 115200 bps by sending the following command.

>HN15=x where x represents the baud rate as follows:

x=0	9600
x=1	19200
x=2	38400

x=3 57600
x=5 115200

If the AI-80 detects a line break (more than 11 consecutive space bits), it automatically resets the baud rate to 9600 bps.

11. The AI-80 returned an error code in response to a command. What does the number mean?

If the AI-80 does not understand a command sent to it, it responds with an error number in the format of:

ERR=(x)

where (x) is an integer number indicating the error code. The following table lists the possible error codes.

Type	Error #	Description
Command	100	Unknown 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
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 of data type mismatch
Program	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
Hardware Properties	10xxxx	Set property command. Invalid number
	11xxxx	Set property command. Invalid source data address
	12xxxx	Set property command. Can't write to a read only property
	13xxxx	Set property command. Data type mismatch
	15xxxx	Get property command. Invalid property number
	16xxxx	Get property command. Invalid destination data address
	17xxxx	Get property command. Can't read from a write only property
	18xxxx	Get property command. Data type mismatch

xxxx = property number passed to the Set/Get command

12. How can I get the status of any running programs?

The AI-80 can execute up to four processes simultaneously. A program for the AI-80 contains at least one process block, though it may contain more. Each of the four virtual processors contained in the AI-80 can be accessed by a collection of registers by using the set data ">" and get data "?" commands.

For example, to read the program counter of process number 1, send the following command:

?VN101***<CR>[<LF>]***

To read the status register for process number 4, send the following command:

?VN403***<CR>[<LF>]***

The register number is always a three digit number in the form of: xyy

x = processor number from 1 to 4
yy = register number

Each virtual processor has a group of registers than control program execution. The following table lists these registers.

Register	Syntax	Access	Description
Source	VNx00	read only	Source of program. If running from RAM, return -1. If running from flash memory, return file number.
Program Counter	VNx01	read only	Returns processor program counter.
Stack Counter	VNx02	read only	Returns number of items placed on processor stack
Status	VNx03	read only	Returns program status: 0=stop, 1=running, 2=halted, other=error code
Suspend Time	VNx04	read/write	Number of milliseconds that the processor is suspended for.
Breakpoint	VNx05	read/write	Program enters halt mode when the program counter equals the value in this register.
Numeric Accumulator	VNx06	read/write	Numeric accumulator. Holds a 32 bit floating point value.
String Accumulator	VSx06	read/write	String accumulator. Holds a string containing 0 to 64 characters. String is null terminated.
Numeric Scratchpad	VNx07	read/write	Numeric scratchpad register. Holds a 32 bit floating point value.
String Scratchpad	VSx07	read/write	String scratchpad register. Holds a string containing 0 to 64 characters. String is null terminated.

13. Is there a summary list of all the AI-80 hardware properties?

Yes, the following table summarizes the hardware properties. For a description of each property, consult the AI-80 User Guide.

ID	ObjName	PropName	DataType	DataAccess	MinVal	MaxVal
1	System	UnitID	string	readonly		
2	System	SoftID	string	readonly		
3	System	HalID	string	readonly		
4	System	FfsID	string	readonly		
5	System	VtpID	string	readonly		
6	System	ParamID	floating			
7	System	ParamTYPE	floating	readonly		
8	System	ParamGetNum	floating	readonly		
9	System	ParamGetStr	string	readonly		
10	System	Reset	floating	writeonly		
11	System	Init	floating	writeonly		
12	System	HvSup	floating			
13	System	VTrim	floating		0	15
108	System	Options	floating	readonly		
166	System	HaltCmds	floating			
109	File	IDlow	floating			
110	File	IDhigh	floating			
111	File	Exist	floating	readonly		
14	Comm	Init	floating	writeonly		
15	Comm	Baud	floating	writeonly	0	4
16	Comm	RxCount	floating	readonly		
17	Comm	GetByte	floating	readonly		
18	Comm	SendByte	floating	writeonly	0	255
132	Comm	SendString	string	writeonly		
19	Comm	RxStatus	floating	readonly		
20	Comm	TxFree	floating	readonly		
103	Comm	CTS	floating	writeonly		
104	Comm	RTS	floating	readonly		
21	Display	SegmentA	floating	writeonly	0	255
22	Display	SegmentB	floating	writeonly	0	255
23	Display	SegmentC	floating	writeonly	0	255
24	Display	SegmentD	floating	writeonly	0	255
25	Display	Blink	floating	writeonly	0	5000
26	Display	Led	floating	writeonly	0	1023
27	Display	LedOn	floating	writeonly	1	3
28	Display	LedOff	floating	writeonly	1	3
29	Display	Num	floating	writeonly		
30	Display	DP	floating	writeonly	0	3
31	Key	Start	floating	readonly		
32	Key	Pause	floating	readonly		
33	Key	Stop	floating	readonly		

34	Key	Up	floating	readonly		
35	Key	Down	floating	readonly		
36	Timer	System	floating		0	10000
37	Timer	Slow	floating		0	10000
38	Timer	Fast	floating		0	100
112	Timer	OnHook	floating			
113	Timer	OffHook	floating			
39	TelInt	PortB	floating			
40	TelInt	Current	floating			
41	TelInt	Reverse	floating			
42	TelInt	LineImp	floating			
43	TelInt	OSI	floating			
44	TelInt	HookDetect	floating	readonly		
105	TelInt	Balance	floating	readonly		
106	TelInt	Length	floating			
107	TelInt	MeasPoint	floating			
45	CPE	HookSwitch	floating			
47	Ring	Freq	floating		10	100
48	Ring	Level	floating		0	80
49	Ring	Enable	floating			
50	ToneB	Freq	floating		20	10000
51	ToneB	Level	floating		0	4
52	ToneB	Enable	floating			
53	ToneB	Phase	floating		0	360
54	Noise	Level	floating		0	2
55	Noise	Enable	floating			
56	ToneA	Enable	floating			
57	ToneA	Freq	floating		20	10000
58	ToneA	FreqMark	floating		20	10000
59	ToneA	Level	floating		0	4
60	ToneA	LevelMark	floating		0	4
61	ToneA	BitTimeSpace	floating		0	1
62	ToneA	BitTimeMark	floating		0	1
63	ToneA	FskBitIndex	floating		0	4096
64	ToneA	FskNumBits	floating	readonly		
65	ToneA	FskContinuous	floating			
66	ToneA	FskHoldCarrier	floating			
67	ToneA	Modulation	floating		0	2
68	ToneA	AmDepth	floating		0	100
69	ToneA	FskActive	floating	readonly		
70	ToneA	Phase	floating		0	360
71	Data	Clear	floating	writeonly		
72	Data	Parity	floating		0	2
73	Data	StopBits	floating		1	100
74	Data	AddMark	floating	writeonly	0	4096
75	Data	AddSpace	floating	writeonly	0	4096

76	Data	AddAlternate	floating	writeonly	0	4096
77	Data	AddByte	floating	writeonly	0	255
78	Data	AddChar	floating	writeonly	0	255
79	Data	AddString	string	writeonly		
80	Data	AddXsum	floating	writeonly		
81	Data	XsumEnable	floating			
82	Data	XsumType	floating		0	1
83	Data	XsumValue	floating		0	65535
84	Measure	Source	floating			
85	Measure	Smoothing	floating			
86	Measure	Level	floating	readonly		
87	DTMF	Enable	floating			
88	DTMF	Source	floating			
89	DTMF	Digit	floating	readonly		
90	DTMF	FreqTol	floating		0	2
91	DTMF	FreqTime	floating		0	20
92	DTMF	MinLevel	floating			
93	DTMF	LowFreq	floating	readonly		
94	DTMF	LowLevel	floating	readonly		
95	DTMF	HighFreq	floating	readonly		
96	DTMF	HighLevel	floating	readonly		
46	Speaker	Volume	floating		1	4
97	Speaker	SignalGain	floating		0	10
98	Speaker	TelIntGain	floating		0	10
99	Speaker	CPEGain	floating		0	10
100	Speaker	BeepFreq	floating		100	5000
101	Speaker	BeepTime	floating		1	10000
102	Speaker	BeepEnable	floating			
114	MFGen	Index	floating			
115	MFGen	Value	floating			
116	MFGen	Level	floating		0	4
117	MFGen	FreqAdjust	floating		-20	20
118	MFGen	OnTime	floating			
119	MFGen	OffTime	floating			
120	MFGen	Symbol	floating	writeonly	0	20
121	MFGen	String	string			
122	MFGen	Active	floating			
123	IO	DeviceID	floating	readonly		
124	IO	Version	floating	readonly		
125	IO	Name	string	readonly		
126	IO	Serial	string	readonly		
127	IO	MemRegister	floating			
128	IO	MemWriteNum	floating	writeonly		
129	IO	MemWriteString	string	writeonly		
130	IO	MemReadNum	floating	readonly		

131	IO	MemReadString	string	readonly		
133	IO	AinChannel	floating		1	4
134	IO	AinLevel	floating	readonly		
135	IO	AinCompare	floating	readonly		
136	IO	AudioOut	floating		0	5
137	IO	AudioIn	floating			
138	IO	AudioMix	floating			
139	IO	DOut	floating		0	255
140	IO	DIn	floating	readonly		
141	IO	BitSet	floating	writeonly	1	15
142	IO	BitClear	floating	writeonly	1	15
143	IO	BitInput	floating		1	24
144	IO	GetBit	floating	readonly		
145	IO	DcLevel	floating	readonly		
146	IO	DcTrigger	floating			
147	IO	DcTime	floating		0	1
148	IO	DcCalibrate	floating			
149	IO	PMode	floating		0	4
150	IO	PTime	floating	readonly		
151	IO	PulseMode	floating		0	5
152	IO	PulseCount	floating		0	
153	IO	PulseGate	floating		0	
154	IO	PulseDuration	floating		0	
155	IO	PulseFreq	floating		0.05	100000
156	IO	PWM1	floating		0	1023
157	IO	PWM2	floating		0	1023
158	IO	CommBaud	floating		0	4
159	IO	CommParity	floating		0	2
160	IO	CommSendByte	floating	writeonly		
161	IO	CommSendString	string	writeonly		
162	IO	CommTxEmpty	floating	readonly		
163	IO	CommRxCount	floating	readonly		
164	IO	CommRxError	floating	readonly		
165	IO	CommGetByte	floating	readonly		
167	FSK	Active	floating			
168	FSK	Source	floating		0	1
169	FSK	LastByte	floating			
170	FSK	MarkTime	floating			
171	FSK	Count	floating		0	700
172	FSK	Index	floating		1	700
173	FSK	ByteValue	floating	readonly		
174	FSK	ByteTime	floating	readonly		
175	FSK	ByteStatus	floating	readonly		