ARBITRARY Precision

7-May-18

CFSoundIV - Digital Audio Repeater

General Description

The CFSound-IV is an extremely versatile digital audio player that plays Windows .WAV files recorded at multiple sample rates, 8 or 16-bit, mono or stereo off of industry standard Secure Digital Flash (SD/SDHC) cards. Sounds may be associated with contact events or played autonomously by utilizing a file naming convention. Extra sound playout functionality is provided via a text configuration file included on the CF card. A built-in Basic interpreter may be used to explicitly control the unit's operation.

Features

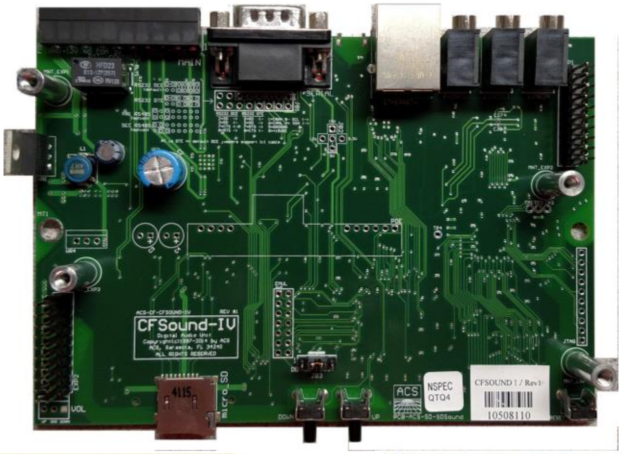
- Uses inexpensive, industry standard Secure Digital FLASH (SD/SDHC) Cards.
- Built-in 20 Watt Class D Stereo (2 x 10W) Amplifier.
- Runs on 12 – 15VDC with supplied 120 – 240VAC 50/60Hz wall transformer
- Built-in 35mW @ 32 ohms Headphone Amplifier.
- RS-232 Serial Port for controlling audio play out via an attached computer or PLC.
- Scriptable via built-in Basic
- USB port for connection to PC as a Flash Drive or Serial device.
- Ethernet connection with programmable configuration and multiple protocol support: DHCP client, FTP server, VNC server, HTTP, TCP/IP Raw, NTP client, SMTP client (via Basic) and Art-Net™.
- Diagnostic LED's to indicate operating status.
- Optional boards for contact inputs to activate sounds.
- Two built-in contact inputs to activate sounds.
- Optional boards for contact outputs activated with sounds for other control.
- Push-To-Talk (PTT) dry relay contact output that can optionally close whenever a sound is played.
- Optional Power Over Ethernet operation.
- Digital Up/Down volume control push buttons with remote connector.
- Optional RS-485 operation.

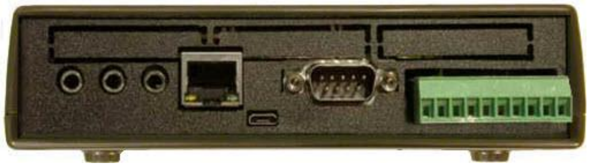
Typical Applications

- Museum Exhibit Control
- Message on Hold
- Amusement Equipment
- Paging and Alarm Systems
- Timed Identification
- Advertising Kiosks

Specifications

Enclosure Dimension:	6.1"(W) x 4.2"(D) x 1.8" (H)	PTT Output Contacts Rating:	1A @ 30VDC, 0.5A @ 120VAC
Module Dimension (board):	5.7"(W) x 3.95"(D) x 1.1" (H)	Contact Input Activation Current:	10mA sink @ 12VDC
Supply Voltage:	12 – 15VDC (wall transformer)	Line Level Outputs:	1.0Vrms (vol controlled) @ 47K ohm
Supply Current (Idle):	85mA @ 12VDC	Operating Temperature & Humidity:	32-113F (0 to +40C) 20-80%RH
Supply Current (medium vol)	250mA @ 12VDC		
Supply Current (full vol):	1500mA @ 12VDC	Supplied wall transformer:	120-240VAC 50/60Hz input

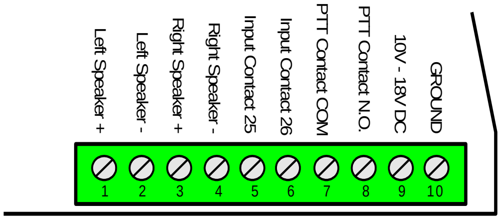




Connections

Speaker and Power Connector

MAIN



Pin #	Signal	Filename
1	Left Speaker +	
2	Left Speaker -	
3	Right Speaker +	
4	Right Speaker -	
5	Input Contact 25	19C.WAV / 19O.WAV
6	Input Contact 26	1AC.WAV / 1AO.WAV
7	PTT Contact COM	
8	PTT Contact N.O.	
9	10VDC – 18VDC	
10	Ground	

Mating Connector: 10-position removable Terminal Block (included)

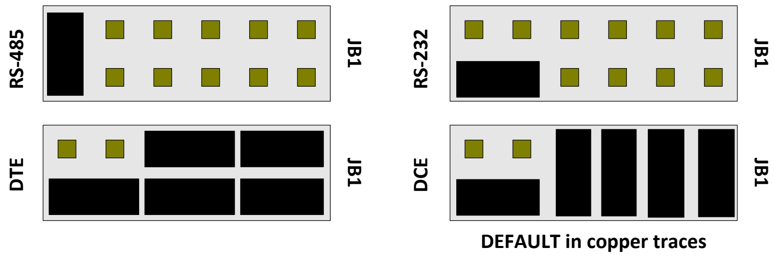
Serial Connector

RS232

PIN	DCE Signal	JB1 as DCE Direction	DTE Signal	JB1 as DTE Direction
1	RS-485 B-	I/O	RS-485 B-	I/O
2	RS-232 TxD	OUT	RS-232 RxD	IN
3	RS-232 RxD	IN	RS-232 TxD	OUT
4				
5	GND	PWR	GND	PWR
6	RS-485 A+	I/O	RS-485 A+	I/O
7	RS-232 CTS	IN	RS-232 RTS	OUT
8	RS-232 RTS	OUT	RS-232 CTS	IN
9	+12-15VDC	PWR	+12-15VDC	PWR

Mating connector: DB9 Female

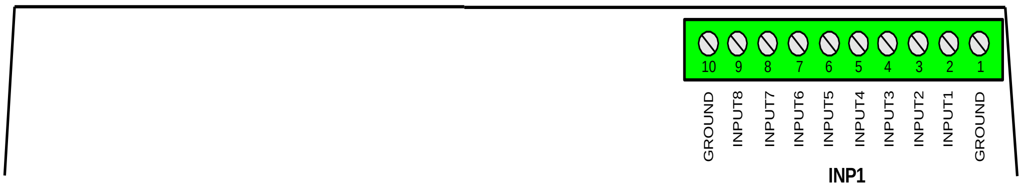
JB1 Serial Configuration Jumpers



The board is configured as RS-232 DCE to allow use of a 1 to 1 cable between the CFSound and a PC. This configuration is established by copper jumpers on the bottom of the board between the JB1 pins.

Optional
Contact Sense 8 Modules

INP1

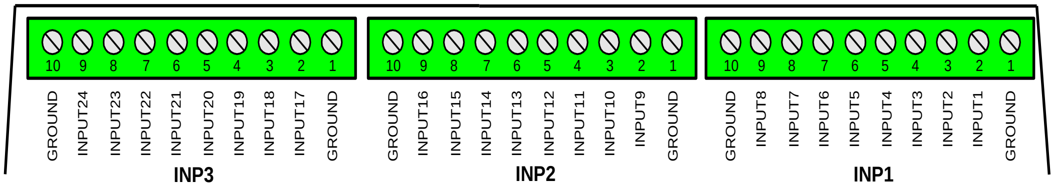


INP1 Pin #	Rear Signal	Front Signal
1	GROUND	GROUND
2	INPUT 1	INPUT 33
3	INPUT 2	INPUT 34
4	INPUT 3	INPUT 35
5	INPUT 4	INPUT 36
6	INPUT 5	INPUT 37
7	INPUT 6	INPUT 38
8	INPUT 7	INPUT 39
9	INPUT 8	INPUT 40
10	GROUND	GROUND

Mating Connector: 10-position removable Terminal Block (included)

Optional
Contact Sense 24
Modules

INP1
INP2
INP3

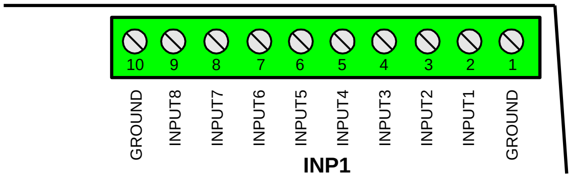


INP3 Pin #	Rear Signal	Front Signal	INP2 Pin #	Rear Signal	Front Signal	INP1 Pin #	Rear Signal	Front Signal
1	GROUND	GROUND	1	GROUND	GROUND	1	GROUND	GROUND
2	INPUT17	INPUT49	2	INPUT9	INPUT41	2	INPUT1	INPUT33
3	INPUT18	INPUT50	3	INPUT10	INPUT42	3	INPUT2	INPUT34
4	INPUT19	INPUT51	4	INPUT11	INPUT43	4	INPUT3	INPUT35
5	INPUT20	INPUT52	5	INPUT12	INPUT44	5	INPUT4	INPUT36
6	INPUT21	INPUT53	6	INPUT13	INPUT45	6	INPUT5	INPUT37
7	INPUT22	INPUT54	7	INPUT14	INPUT46	7	INPUT6	INPUT38
8	INPUT23	INPUT55	8	INPUT15	INPUT47	8	INPUT7	INPUT39
9	INPUT24	INPUT56	9	INPUT16	INPUT48	9	INPUT8	INPUT40
10	GROUND	GROUND	10	GROUND	GROUND	10	GROUND	GROUND

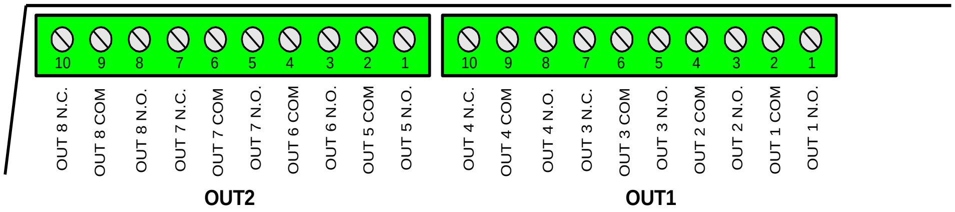
Mating Connectors: 10-position removable Terminal Block (included)

Optional
Contact I/O 8 Modules

INP1
OUT1
OUT2



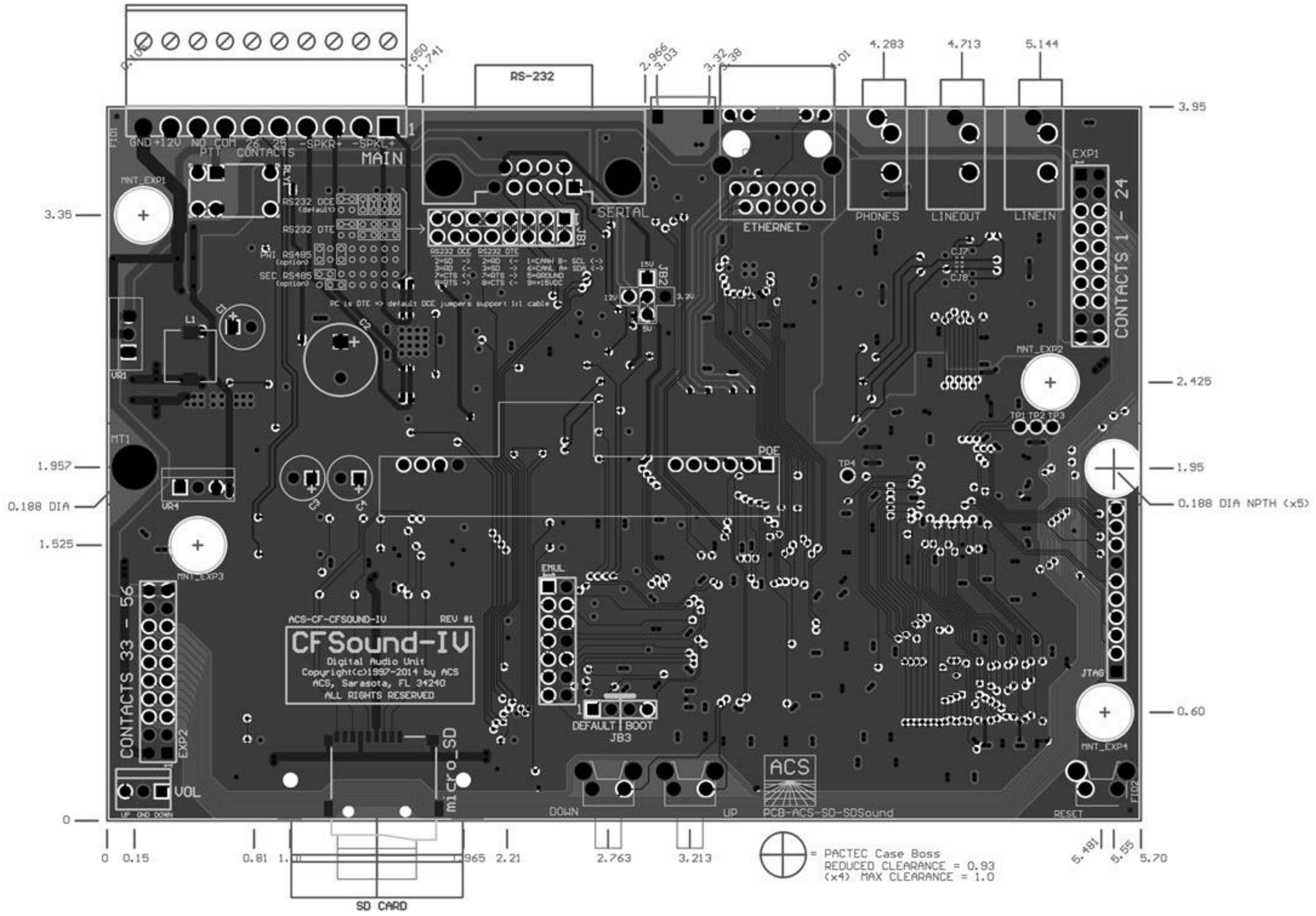
INP1 Pin #	Rear Signal	Front Signal
1	GROUND	GROUND
2	INPUT 1	INPUT 33
3	INPUT 2	INPUT 34
4	INPUT 3	INPUT 35
5	INPUT 4	INPUT 36
6	INPUT 5	INPUT 37
7	INPUT 6	INPUT 38
8	INPUT 7	INPUT 39
9	INPUT 8	INPUT 40
10	GROUND	GROUND



OUT2 Pin #	Rear Signal	Front Signal	OUT1 Pin #	Rear Signal	Front Signal
1	OUT 5 N.O.	OUT 37 N.O.	1	OUT 1 N.O.	OUT 33 N.O.
2	OUT 5 COM	OUT 37 COM	2	OUT 1 COM	OUT 33 COM
3	OUT 6 N.O.	OUT 38 N.O.	3	OUT 2 N.O.	OUT 34 N.O.
4	OUT 6 COM	OUT 38 COM	4	OUT 2 COM	OUT 34 COM
5	OUT 7 N.O.	OUT 39 N.O.	5	OUT 3 N.O.	OUT 35 N.O.
6	OUT 7 COM	OUT 39 COM	6	OUT 3 COM	OUT 35 COM
7	OUT 7 N.C.	OUT 39 N.C.	7	OUT 3 N.C.	OUT 35 N.C.
8	OUT 8 N.O.	OUT 40 N.O.	8	OUT 4 N.O.	OUT 36 N.O.
9	OUT 8 COM	OUT 40 COM	9	OUT 4 COM	OUT 36 COM
10	OUT 8 N.C.	OUT 40 N.C.	10	OUT 4 N.C.	OUT 36 N.C.

Mating Connectors: 10-position removable Terminal Block (included)

Mechanical



File Naming Format for CFSound style operation

(see CFSound-III User's Manual for more detailed information)

Filename format: XX[COPRSBNFD].WAV

Where:

XX	Two digit ASCII Hex identifier 01 - FE, may be associated contact number
C	File plays on Closure of contact XX, may not be used with O
O	File plays on Opening of contact XX, may not be used with C
P	File plays while contact XX is closed or open, may not be used with B
R	File repeats, may not be used with B
S	On board PTT relay and contact XX will activate while sound is playing
B	File plays as background when no other sound is playing, may not be used with C, O, P, R or N
N	File playing is non-interruptable, may not be used with R or B
F	Matching DMX channel number Fades up/down with sound start/stop
D	First 32 channels set to entries in associated DMX scene file with sound start/stop
.WAV	File extension identifies Windows PCM sound file format

CFSOUND.INI Configuration File

(see CFSound-III User's Manual for more detailed information)

[Section] / Parameter	**Description**
[Comm]	**Communications Port Section**
BaudRate=ddddd	Sets the serial port baudrate to the decimal value ddddd. Default=2400.
[DEBUG]	**Debug Section**
ShowStartStop=TRUE/FALSE	Enables RS-232 message display of sound start/stop events. Default=FALSE.
[Background]	**Background Section**
BackgroundDelay=ddddd	Sets the delay in seconds between background sound playouts to the decimal value ddddd. Default=0.
BackgroundRestart=TRUE/FALSE	Enables interrupted background sound to restart from the beginning instead of where it was interrupted. Default=FALSE.
[Quiz]	**Quiz Section**
QuizMode=TRUE/FALSE	Enables Quiz/Kiosk mode of operation. Default=FALSE.
QuestionContacts=dd	Sets the number of question contacts to the decimal value dd. Default=4.
AnswerContacts=dd	Sets the number of answer contacts to the decimal value dd. Default=4.
NoAnswerTimeout=dd	Sets the delay in seconds between the end of the question sound and the timeout answer sound to the decimal value dd. Default=5.
AwaitAnswerSound=xx	Sets the hexadecimal sound number xx to play after the question sound before the timeout answer sound. Default=0 (no sound).
AnswerWithoutQuestionSound=xx	Sets the hexadecimal sound number xx to play if an answer contact is activated before a question contact. Default=0 (no sound).
[Contacts]	**Contacts Section**
Force=TRUE/FALSE	Setting this value to TRUE restores the original CFSound contact behavior wherein the contact's active status is 'forced' upon reset, power-up or card-insertion. This will cause associated sound activation if the contact was active. Setting this value to FALSE (the default) causes the new behavior wherein the contact's current status is sampled upon reset, power-up or card-insertion. This will cause no associated sound activation until the contact is re-activated. Default=FALSE.
SequenceContactNumber=dd	Sets the number of the contact that will play sounds in sequence to the decimal value dd. Default=0 (no sequencing)
FirstSoundNumber=dd	Sets the first sound number that will be played in sequence to the decimal value dd. Default=1 (sound #1)
LastSoundNumber=dd	Sets the last sound number that will be played in sequence to the decimal value dd. Default=127 (sound #127)
SaveNIContacts=TRUE/FALSE	Setting this value to TRUE will remember any contact events that occur while a non-interruptible sound is playing. Note that this can cause a non-interruptible sound to play again if its contact is re-activated while it is playing. Default=FALSE
OutputContactModulus=dd	Setting this value to non-zero will cause the output contacts associated with sounds to repeat on the modulo value if **QuizMode=FALSE**. Example: OutputContactModulus=4 activates contact outputs 1 through 4 for sounds 1 through 4, contact outputs 1 through 4 for sounds 5 through 8, etc. Default=0
RandomSequence=TRUE/FALSE	Setting this value to TRUE will cause each activation of the SequenceContactNumber to play a random sound from the range FirstSoundNumber to LastSoundNumber. Default=FALSE
OffsetContactNumber=dd	Sets the number of the contact that will offset the sounds associated with the other contacts by ContactOffsetAmount to the decimal value dd. Does not affect Sequence or Quiz mode. Default=0 (no offset)
ContactOffsetAmount=dd	Sets the value that will be added the the input contact number when the OffsetContactNumber input is active, to offset the actual sound number that will play to the decimal value dd. Does not affect Sequence or Quiz mode. Default=0 (no offset amount)
AutoplayEntireSequence= TRUE/FALSE	Setting AutoplayEntireSequence to TRUE causes the entire sequence of sounds to be played once whenever the SequenceContactNumber activates one time. Default=FALSE (no autoplay)
LineInputEnableContactNumber=dd	Sets the number of the contact that will stop any sound currently playing and enable the Line level Input to the decimal value dd. Audio on the Line level Input is amplified to the current volume setting and appears on the speaker and Line level Output. Default=0 (no Line In control contact)
PttOutputWithLineInputEnableContact =TRUE/FALSE	Setting this value to TRUE will cause the PTT relay to follow the non-zero LineInputEnableContactNumber state, otherwise the PTT relay activation is controlled by sounds with the Relay attribute in their filename. Default=FALSE (PTT for sounds w/Relay attr)
[LineIn]	**LineIn Section**
LineInputAlwaysEnabled=TRUE/FALSE	Setting this value to TRUE enables the Line level Input always. when no sound is playing. When this is FALSE, the Line level Input is controlled by the LineInputEnableContactNumber. Default=FALSE (Line level Input disabled)

RS-232 Protocol

(see CFSound-III User’s Manual for more detailed information)

SOH / ETX Commands / Responses

Command	Serial Character Sequence
Start a Sound	<SOH> ”p” “+” {Sound number in two digit ASCII Hex, (01 – FE)} <ETX>
Stop a Sound	<SOH> ”p” “-” {Sound number in two digit ASCII Hex, “00” stops currently playing sound} <ETX>
Queue a Sound	<SOH> ”p” “&” {Sound number in two digit ASCII Hex, (01 – FE)} <ETX>
Flush queued Sounds	<SOH> ”p” “~” <ETX>
Stop playing Sound and flush queued Sounds	<SOH> ”p” “!” <ETX>
Set volume	<SOH> ”v” “=” {volume in two digit ASCII Hex, 00 – 3F} <ETX>
Increase volume	<SOH> ”v” “+” {volume increase in two digit ASCII Hex, 00 – 3F} <ETX>
Decrease volume	<SOH> ”v” “-” {volume increase in two digit ASCII Hex, 00 – 3F} <ETX>
Fade volume	<SOH> ”v” “<” {fade volume to zero in seconds expressed as two digit ASCII Hex, 00 – 3F} <ETX>
Mute amplifier	<SOH> ”a” “-” <ETX>
Un-mute amplifier	<SOH> ”a” “+” <ETX>

Basic Commands

(see CFSound-IV Basic Programming Manual for more detailed information)

Variables

- Basic has four types of variables:
    - **32-bit Integer Numeric**, **32-bit Integer Numeric Arrays, unsigned 8-bit character Strings** and **unsigned 8-bit character String Arrays**.
- Variable names *are* case sensitive. The may contain letters, numbers and underscore but they must start with a letter. They can be up to 32 characters long. String variables names must end with a '$'.
- Numeric variables can assume the integer values **(–2,147,483,648 ≤ variable ≤ +2,147,483,647)**.
- Character Strings are limited to **255 characters** in length.
- Variable arrays are indexed with up to three array subscripts separated by commas and enclosed in square brackets **[ ]** and must be **DIM**ensioned before they are used.
- The number of variables is limited only by the available memory.

System Variables

<table>
<tr><td><b>@TIMER[x]</b></td><td>(10) 16-bit timers that decrement at 50Hz (20mSEC) until zero.</td></tr>
<tr><td><b>@PORT[x[</b></td><td>(256) 8-bit expansion port access for rear I/O module</td></tr>
<tr><td><b>@PORT2[x[</b></td><td>(256) 8-bit expansion port access for front I/O module</td></tr>
<tr><td><b>@CONTACT[x[</b></td><td>(56) contact I/O access</td></tr>
<tr><td><b>@CLOSURE[x[</b></td><td>(56) contact closure event access</td></tr>
<tr><td><b>@OPENING[x[</b></td><td>(56) contact opening event access</td></tr>
<tr><td><b>@FEOF[#N[</b></td><td>End of File on file #N</td></tr>
<tr><td><b>@FILE.SIZE[#N]</b></td><td>Size in bytes of previously opened file #N</td></tr>
<tr><td><b>@FILE.POSITION[#N]</b></td><td>Ascertain or set the position of the next file read / write operation of a previously opened file #N</td></tr>
<tr><td><b>@SOCKET.EVENT[#N]</b></td><td>Determine the state of an opened streaming socket connection</td></tr>
<tr><td><b>@SOCKET.TIMEOUT[#N]</b></td><td>Control the timeout period of a socket connection send / receive data phases</td></tr>
<tr><td><b>@SECOND,@MINUTE,@HOUR,
@DAY,@DATE,@MONTH,@YEAR</b></td><td>Real Time Clock / Calendar:
<table>
<tr><td>@SECOND</td><td>00 <= seconds <= 59</td></tr>
<tr><td>@MINUTE</td><td>00 <= minutes <= 59</td></tr>
<tr><td>@HOUR</td><td>00 <= hour <= 23</td></tr>
<tr><td>@DOW</td><td>1 <= day of week <= 7</td></tr>
<tr><td>@DATE</td><td>1 <= date of month <= 31</td></tr>
<tr><td>@MONTH</td><td>1 <= month of year <= 12</td></tr>
<tr><td>@YEAR</td><td>00 <= year <= 99</td></tr>
</table></td></tr>
<tr><td><b>@SOUND$</b></td><td>Sound playing queue access</td></tr>
<tr><td><b>@VOL</b></td><td>Sound volume access</td></tr>
<tr><td><b>@NSVOL</b></td><td>Sound volume access without saving</td></tr>
<tr><td><b>@BAUD</b></td><td>Serial Port baud rate access</td></tr>
<tr><td><b>@MSG$</b></td><td>Serial Port delimited message access</td></tr>
<tr><td><b>@SOM</b></td><td>Delimited message Start Of Message character</td></tr>
<tr><td><b>@EOM</b></td><td>Delimited message End Of Message character</td></tr>
<tr><td><b>@MSGENABLE</b></td><td>Enable / disable MSG$ parsing of the serial data stream</td></tr>
<tr><td><b>@EOT</b></td><td>Returns 1 when any PRINT serial data has finished transmitting</td></tr>
<tr><td><b>@SMTP.EVENT</b></td><td>Returns the last Simple Mail Transfer Protocol event</td></tr>
<tr><td><b>@SMTP.MESSAGE$</b></td><td>Returns any text message associated with the @SMTP.EVENT</td></tr>
<tr><td><b>@PTT</b></td><td>Push-to-Talk relay control</td></tr>
<tr><td><b>@MUTE</b></td><td>Mute / Un-mute the speaker amplifier</td></tr>
<tr><td><b>@LINEIN</b></td><td>Line level input control</td></tr>
<tr><td><b>@DMX.CHANNELS</b></td><td>Sets the number of transmitted channels sent via ArtNet™</td></tr>
<tr><td><b>@DMX.DATA[x]</b></td><td>Gets or Sets the current value of the channel data x</td></tr>
<tr><td><b>@SOUNDFRAMEPRESCALER</b></td><td>Sets the number of ticks between @SOUNDFRAMESYNC events while sound is playing</td></tr>
<tr><td><b>@SOUNDFRAMESYNC</b></td><td>Gets the frame number of the currently playing sound</td></tr>
<tr><td><b>@CONFIG.ITEMS</b></td><td>Returns the total number of configuration items</td></tr>
<tr><td><b>@CONFIG.TYPE[n]</b></td><td>Returns the type of the configuration item n:
<table>
<tr><th>@CONFIG.TYPE[n]</th><th>Item Type</th><th>Fields</th></tr>
<tr><td>1</td><td>Byte</td><td>0</td></tr>
<tr><td>2</td><td>Boolean</td><td>0</td></tr>
<tr><td>3</td><td>Unsigned short</td><td>0</td></tr>
<tr><td>4</td><td>Baudrate selector</td><td>0</td></tr>
<tr><td>5</td><td>Parity selector</td><td>0</td></tr>
<tr><td>6</td><td>Data Bits selector</td><td>0</td></tr>
<tr><td>7</td><td>Stop Bits selector</td><td>0</td></tr>
<tr><td>8</td><td>Keybeep selector</td><td>0</td></tr>
<tr><td>9</td><td>Firmware Version</td><td>0</td></tr>
<tr><td>10</td><td>Keypad style</td><td>0</td></tr>
<tr><td>11</td><td>Keypad scheme</td><td>0</td></tr>
<tr><td>12</td><td>Protocol selector</td><td>0</td></tr>
<tr><td>13</td><td>MAC address</td><td>6</td></tr>
<tr><td>14</td><td>IP address (only display if static)</td><td>4</td></tr>
<tr><td>15</td><td>IP address</td><td>4</td></tr>
</table></td></tr>
</table>

16	Hex Byte	0
17	Hex Unsigned short	0
18	Hex Array	8
19	Short	0
20	RS485 Mode	0

@CONFIG.NAME$[n]	Returns the name of the configuration item n
@CONFIG.VALUE$[n {, f]}	Returns the human readable value of the configuration item n {optional field number f}
@CONFIG.MIN[n]	Returns the allowed minimum value of configuration item n
@CONFIG.MAX[n]	Returns the allowed maximum value of the configuration item n
@CONFIG.FIELDS[n]	Returns the number of fields for configuration item n
@CONFIG.FIELD$[n, f]	Returns the human readable value of the configuration item n field f
@CONFIG.SEPARATORS[n, f]	Returns the human readable value of the configuration item n field f field separator
@CONFIG.VALUE[n {, f}]	Gets or Sets the value of the configuration item n {optional field number f}
@CONFIG.DEFAULT[n {, f]}	Gets the default value of the configuration item n {optional field number f}
@CONFIG.WRITE[n {, f}]	Writes the current value of the configuration item n {optional field number f} to NVM
@CARD.MOUNT	Mount / Unmount the SD card

Statements

BREAK {line / `label}	Exit from within FOR / NEXT or WHILE / WEND loops {optionally going to a line / `label]
CHANGE string, replacement	Searches program for string then prompts for replacement
CLEAR	Erase variables
CLOSE #N	Close file #N(0 – 9) opened with OPEN statement
CONST var{$}=value {, var{$}=value …}	Defines one or more constant variables that can't be modified after they are created
CONTINUE	Continues the next iteration of FOR / NET or WHILE / WEND loops
DATA	Inline DATA statements for READ and ORDER statements
DEL path	Delete CF card files
DELAY value	Pause program execution for value * 20mSEC
DIM var{$}[size1{, size2{, size3}}]	Dimension numeric or string variable to hold up to size1 elements {optional up to 3 dimensions]
DIR {path}	Show files on the SD card with optional path / wildcards
EDIT line	Edit line on connected ANSI terminal
END	Terminate program with no message
ERROR value	Force a program error
FOR var=init TO limit [STEP increment]	Perform counted loop of statements until NEXT statement with optional BREAK / CONTINUE
FINPUT #N, var{$}, … , var{$}	Get the value for one or more variables from a single line from previously opened file #N
FPRINT #N, expr {, expr …}	Write the value of one or more expressions to a single line into previously opened file #N
FOPEN #N, recordlength, "path"	Open file #N for fixed length record I/O
FREAD #N, recordnumber, var[$], var[$], … , var[$]	Reads ASCII data from fixed length record file #N at recordnumber into variables
FWRITE #N, recordnumber, var[$], var[$], … , var[$]	Writes ASCII data to fixed length record file #N at recordnumber from variables
FINSERT #N, recordnumber, var[$], var[$], … , var[$]	Inserts ASCII data to fixed length record file #N at recordnumber from variables
FDELETE #N, recordnumber	Deletes recordnumber from fixed length record file #N
FUNCTION name{$}(parm1{$}, … parmN{$})	Define a user function name with zero or more integer or string parameters
ENDFUNCTION	Ends a user defined function
GOSUB line / `label	Call a subroutine starting at line / `label
GOTO line / `label	Jump to program line / `label
INCLUDE path	Include Basic statements from file path
IF test THEN line/statement [ELSE line/statement]	IF test evaluates non-zero jump to program line or execute statement, optional ELSE clause
INPUT ["prompt",]var	Get value of variable from serial port with optional prompt
INPUT #N, var	Get value of variable from file #N
{LET }var{$}=expr{$} *(default statement)*	Sets variable = expression, LET is optional
LIF test THEN statement{ : statement}	IF test evaluates non-zero execute statements to end of line
LIST {start {, end}}	LIST program lines to the serial port
LIST #N{ start {, end}}	LIST program lines to OPENed file #N
LOAD path	LOAD (or chain to) program from SD card
MD path	Makes a new Directory on SD card
MEMORY	Displays the currently available program, resource and SD card memory
NEW	Erase all program statements and clear variables
NEXT [var]	End of a counted loop of statements from FOR statement
ON expr, GOSUB line0,line1,line2,…,lineN	Case statement subroutine dispatch
ON expr, GOTO line0,line1,line2,…,lineN	Case statement execution dispatch
ONERROR GOTO line	One-shot error handling
ONEVENT @specialvar, GOSUB line	Semi-asynchronous event handling via subroutine

Special Variable	Event
@TIMER[x]	event occurs one time whenever the timer counts down to zero. Special variable **@TIMER(0)** is the highest priority, followed by **@TIMER(1)**, … then **@TIMER(9)**. 0 <= x <= 9
@CLOSURE[x]	event occurs whenever the associated CFSound-4 contact has closed. 0 <= x <= 55
@OPENING[x]	event occurs whenever the associated CFSound-4 contact has opened. 0 <= x <= 55
@FEOF[#N]	event occurs after FREAD #N reaches end of file #N
@SECOND	event occurs once per second.

@MINUTE	event occurs once per minute.
@HOUR	event occurs once per hour.
@DOW	event occurs once per day.
@DATE	event occurs once per day.
@MONTH	event occurs once per month.
@YEAR	event occurs once per year.
@MSG$	event occurs after receipt of a serial character stream delineated by the **@SOM** and **@EOM** characters.
@EOT	event occurs after complete transmission of serial data stream
@SOUND$	event occurs after the last queued **@SOUND$** sound has finished playing.

OPEN #N,"path","options"	OPEN filename path as file #N for access via DIR #, INPUT # or PRINT# statements
ORDER line	Position READ data pointer to statement line number
PLAY file	Play sound file and wait for completion
PRINT expr{$} {, expr{$} …}	PRINT one or more numeric or string expressions to the serial port
PRINT #N, expr{$} {, expr{$} …}	PRINT one or more numeric or string expressions to opened file #N
PRINT USING fmt$ expr{$} {, expr{$} …}	PRINT zero or more formatted numeric or string expressions to the serial port
PRINT #N, USING fmt$ expr{$} {, expr{$} …}	PRINT zero or more formatted numeric or string expressions to opened file #N
READ var{$} {, var{$} …}	READ data from DATA statements into numeric or string variables
RETURN	RETURN from subroutine invoked via GOSUB statement
REM	Comment, remainder of line is ignored
REN oldfile, newfile	REName oldfile to newfile on SD card
RESQ {start{-end}{, new}{, incr}}	Resequences program lines start through end and writes them to programname.RSQ
RUN {line} / RUN {path}	Execute program in memory or from path at lowest or line number
SAVE {path}	SAVE the current program to a SD card file
SEARCH string {filename}	Performs case insensitive search for string in memory or optional filename with wildcards
SIGNAL @specialvar	SIGNAL event associated with specialvar
SORT var{$}	Sorts an integer or string array variable in ascending order
SMTP.SERVER name, ipaddr{,port{,userb64,passb64}}	Prepares the SMTP network stack for subsequent SMTP.SEND operation
SMTP.SEND from, to, cc, subject, message	Sends a text message via the previously configured SMTP.SERVER
SMTP.SEND #N, from, to, cc, subject{,header}	Sends the contents of a previously opened file #N via the previously configured SMTP.SERVER
SOCKET.ASYNC.CONNECT #N, "ip:port", connect( ), send( ), recv( )	Initiates an outgoing asynchronous network socket connection as file #N on ip address / port number where execution is controlled by the connect( ), send( ) and recv( ) functions
SOCKET.ASYNC.LISTEN #N, ":port", connect( ), recv( ), send( )	Initiates an incoming asynchronous network socket reception as file #N on ip port number where execution is controlled by the connect( ), recv( ) and send( ) functions
STOP	Terminate program and display message
TYPE path	Display SD card file on serial port
VARS	Displays a table of the name, type and current value of variables currently defined or used
WAIT @systemvar	Pause execution until systemvar event occurs
WHILE test : statement{s} : WEND	Conditional execution code block loop with BREAK / CONTINUE

Operators

Operator	Description	Priority
`NOT`	Logical NOT	7
`-`	Unary minus (negate)	7
`~`	Bitwise NOT (1's complement)	7
`* , / , %`	Multiplication, division, modulus	6
`+`	Addition, string concatenation	5
`-`	Subtraction	5
`<< , >>`	Left Shift, Right Shift	4
`= , <>`	Assign / test equal, test NOT equal (numeric or string)	3
`< , <= , > , >=`	LT, LE, GT, GE (numeric only)	3
`& , | , ^`	AND, OR, Exclusive OR	2
`AND , OR`	Logical AND, OR	1

Functions

<table>
<tr><td><b>ASC(char)</b></td><td>Returns integer value of ASCII character argument</td></tr>
<tr><td><b>ABS(expr)</b></td><td>Returns absolute value of numeric expression argument</td></tr>
<tr><td><b>CHR$(expr)</b></td><td>Returns character equivalent of expression value argument</td></tr>
<tr><td><b>COS(expr)</b></td><td>Returns an integer scaled cosine value of the degree expression where -1024 ≤ COS( ) ≤ 1024</td></tr>
<tr><td><b>ERR( )</b></td><td>Returns last error number</td></tr>
<tr><td><b>ERR$( )</b></td><td>Returns string error message of last error number</td></tr>
<tr><td><b>FILE.EXISTS(path$)</b></td><td>Returns one of the file specified by “path” exists else returns zero</td></tr>
<tr><td><b>FIND(var$,searchstr$ {, startpos})</b></td><td>Returns zero based position of searchstr$ in string variable argument starting at zero (or optional startpos) or -1 if not found</td></tr>
<tr><td><b>FMT$(fmt$ {,expr{$}, … , expr{$}})</b></td><td>Returns formatted ASCII string of zero or more expressions using printf() style fmt$ argument:
<b>% {Flags}{Width}{.Precision}Type</b>
<table>
<tr><td><b>Type</b></td><td>Required character that determines whether the associated <i>argument</i> is interpreted as a character, a string, or a number:
<table>
<tr><td>c</td><td>character</td></tr>
<tr><td>d</td><td>signed decimal integer</td></tr>
<tr><td>i</td><td>signed decimal integer</td></tr>
<tr><td>u</td><td>unsigned decimal integer</td></tr>
<tr><td>s</td><td>string</td></tr>
<tr><td>o</td><td>unsigned octal integer</td></tr>
<tr><td>x</td><td>unsigned hexadecimal integer</td></tr>
<tr><td>X</td><td>unsigned HEXADECIMAL integer</td></tr>
</table></td></tr>
<tr><td><b>Flags</b></td><td>Optional character or characters that control justification of output and printing of signs, blanks, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.
<table>
<tr><td>-</td><td>left align the result in the given field width</td></tr>
<tr><td>+</td><td>prefix the output with a sign (+/-) if the type is signed</td></tr>
<tr><td>0</td><td>if Width is prefixed with 0, zeros are added until the minimum width is reached. If 0 and – appear, the 0 is ignored. If 0 is specified with an integer format, the 0 is ignored.</td></tr>
<tr><td><i>blank(‘ ’)</i></td><td>prefix the output with a blank if the result is signed and positive; the blank is ignored if both the blank and + flags appear</td></tr>
<tr><td>#</td><td>when used with o, x or X format, prefix any nonzero output value with 0, 0x or 0X respectively, otherwise ignored</td></tr>
</table></td></tr>
<tr><td><b>Width</b></td><td>Nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values — depending on whether the – flag (for left alignment) is specified — until the minimum width is reached. If <b>Width</b> is prefixed with 0, zeros are added until the minimum width is reached (not useful for left-aligned numbers). The <b>Width</b> specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if <b>Width</b> is not given, all characters of the value are printed (subject to the <b>Precision</b> specification).</td></tr>
<tr><td><b>Precision</b></td><td>Specifies a nonnegative decimal integer, preceded by a period (<b>.</b>), which specifies the number of characters to be printed, the number of decimal places, or the number of significant digits. Unlike the <b>Width</b> specification, the precision specification can cause truncation of the output value. If <b>Precision</b> is specified as 0 and the value to be converted is 0, the result is no characters output.
<table>
<tr><td>c</td><td><b>Precision</b> has no effect</td></tr>
<tr><td>d,i,u,o, x,X</td><td><b>Precision</b> specifies the minimum number of digits to be output. If the number of digits is less than <b>Precision</b>, the output is padded on the left with zeroes. The value is not truncated when the number of digits exceeds <b>Precision</b></td></tr>
<tr><td>s</td><td><b>Precision</b> specifies the maximum number of characters to be output. Characters in excess of <b>Precision</b> are not output</td></tr>
</table></td></tr>
</table></td></tr>
<tr><td><b>GETCH(expr)</b></td><td>Returns next available serial character or -1 if none available if expression is zero else waits for and returns next character</td></tr>
<tr><td><b>HEX.STR$(expr {,digits})</b></td><td>Returns a string hex representation of expression optionally constrained to digits length</td></tr>
<tr><td><b>HEX.VAL(expr$)</b></td><td>Returns the numeric value of the string hex expression</td></tr>
<tr><td><b>INSERT$(var$, start, var2$)</b></td><td>Returns string variable with string variable2 inserted at zero based start character position</td></tr>
<tr><td><b>LEFT$(var$,length)</b></td><td>Returns leftmost length characters of string variable argument</td></tr>
<tr><td><b>LEN(var$)</b></td><td>Returns length of string variable argument</td></tr>
<tr><td><b>MID$(var$,start,length)</b></td><td>Returns length number of characters of string variable from zero based start character position</td></tr>
<tr><td><b>MULDIV(number,multiplier,divisor)</b></td><td>Returns a 32 bit result of ((number * multiplier) / divisor) where number, multiplier and divisor are 64-bit internally</td></tr>
<tr><td><b>MULMOD(number,multiplier,divisor)</b></td><td>Returns a 32 bit result of ((number * multiplier) % divisor) where number, multiplier and divisor are 64-bit internally</td></tr>
<tr><td><b>RIGHT$(var$,length)</b></td><td>Returns rightmost length characters of string variable argument</td></tr>
<tr><td><b>REPLACE$(var$, start, var2$)</b></td><td>Returns string variable overwritten with string variable2 at zero based start character position</td></tr>
<tr><td><b>RND(expr)</b></td><td>Returns a pseudo random number from 0 to value of expression – 1</td></tr>
<tr><td><b>SIN(expr)</b></td><td>Returns an integer scaled sine value of the degree expression where -1024 ≤ SIN( ) ≤ 1024</td></tr>
<tr><td><b>STR$(expr)</b></td><td>Returns a string representation of numeric expression</td></tr>
</table>

SOCKET.SYNC.CONNECT(#N, “ip:port”, connect( ), send( ), recv( ))	Initiates an outgoing synchronous network socket connection as file #N on ip address / port number where execution is controlled by the connect( ), send( ) and recv( ) functions
SOCKET.SYNC.LISTEN(#N, “:port”, connect( ), recv( ), send( ))	Initiates an incoming synchronous network socket reception as file #N on ip port number where execution is controlled by the connect( ), recv( ) and send( ) functions
UBOUND(dimVar{[dimNumber]}	Returns the size of dimVar dimension zero as declared in the DIM statement optionally other dimensions.
VAL(expr$)	Returns numeric value of string expression representation of a number

Errors

Error #	Error Message	Causes
1	"Syntax error in line dd"	Incorrect statement format
2	"Illegal program command error in line dd"	Direct mode only statement in program mode
3	"Illegal direct command error in line dd"	Program mode only statement in direct mode
4	"Line number error in line dd"	Target line number not in program
5	"Wrong expression type error in line dd"	Numeric value when String expected or vice versa
6	"Divide by zero error in line dd"	Division by zero
7	"Nesting error in line dd "	NEXT without preceding FOR, RETURN without preceding GOSUB
8	"File not open error in line dd "	CLOSE#, LIST#, PRINT# or INPUT# without successful OPEN statement
9	"File already open error in line dd "	OPEN# on already open file
10	“File # Out of Range error in line dd “	#N argument not 0 <= #N <= 9
11	"Input error in line dd "	Numeric value expected in INPUT # statement
12	"Dimension error in line dd "	Subscript on non-dimensioned variable
13	“Index Out of Range error in line dd “	Subscript out of range
14	"Data error in line dd "	ORDER line # not DATA statement, READ past DATA statements
15	"Out of memory error in line dd "	Insufficient memory
16	"No File System error in line dd "	Basic running without CF card
17	“Unknown @var error in line dd “	Unknown special variable
18	"Timer # out of range error in line dd "	@TIMER(x) subscript out of range 0 - 9
19	"Port # out of range error in line dd "	@PORT(x) subscript out of range 0 - 255
20	"Contact # out of range error in line dd "	@CONTACT(x), @CLOSURE(x), @OPENING(x) subscript out of range
21	"Stack Overflow error in line dd "	Too many nested FOR and/or GOSUB and/or events
22	"No CF card error in line dd "	Statement requiring Compact Flash card with no card detected
23	"Invalid .WAV file error in line dd "	.WAV file format not 44.1KHz 16-bit mono or stereo
24	“LCDx arguments Out of Range”	One or more argument to a LCDx statement are out of range
25	“FWRITE record # Out of Range”	FWRITE record number out of range
26	“FWRITE exceeds record length error”	FWRITE record length exceeds FOPEN record length
27	“FINSERT record # Out of Range”	FINSERT record number out of range
28	“FINSERT exceeds record length error”	FINSERT record length exceeds FOPEN record length
29	“FDELETE past end of file error”	FDELETE record number past the current end of file
30	“Can’t delete file”	Error deleting file
31	“Can’t make directory”	Error creating directory
32	“Can’t rename file”	Error renaming file
33	“No DMX module error in line dd”	@DMX--- specialvar access attempted with no DMX I/O module present
34	“DMX Channel # Out of Range error in line dd”	@DMXDATA(x) access where x >= 511
35	“DMX Analog # Out of Range error in line dd”	@DMXANALOG(x) access where x >= 7
36	“DMX Analog # Read Only error in line dd”	Attempt to set @DMXANALOG(x)
37	“Unknown command”	Unknown command
38	“Can’t use @VAR in line dd”	Illegal use of specialvar in FOR, DIM, INPUT, READ, FREAD or FINPUT statement
39	“Mis-matched quotes in line dd”	Missing one of a pair of double quotes delimiting a string
40	“Resource already exists”	
41	“Font # out of range”	
42	“.fonts file invalid”	
43	“Scheme # out of range”	
44	“.schemes file invalid”	
45	“Obj # out of range”	
46	“Screen # out of range”	
47	“.screens file invalid”	
48	“Config # out of range”	
49	“Config Item < min or > max”	
50	“DRAW.POLYGON”	
51	“SD Card”	
52	“File System”	
53	“Read Only”	Attempt to write to a CONST variable
54	“Option # Out of Range”	
55	“Data # Out of Range”	
56 – 57	Internal Usage	
58	“SMTP Connection Failed”	
57 - 32767	“x error in line dd”	ERROR x statement

Please Read Carefully:

Information in this document is provided solely in conjunction with Arbitrary Precision products. Arbitrary Precision reserves the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time without notice.

All Arbitrary Precision products are sold pursuant to Arbitrary Precision's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the Arbitrary Precision products and services described herein, and Arbitrary Precision assumes no liability whatsoever relating to the choice, selection or use of the Arbitrary Precision products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license or grant by Arbitrary Precision for the use of such third party products and services, or any intellectual property contained therein or considered a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ARBITRARY PRECISION'S TERMS AND CONDITIONS OF SALE ARBITRARY PRECISION DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ARBITRARY PRECISION PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ARBITRARY PRECISION PRODUCTS ARE NOT AUTHORIZED FOR USE IN WEAPONS. NOR ARE ARBITRARY PRECISION PRODUCTS DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTICAL APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS. THE PURCHASER SHALL USE PRODUCTS AT PURCHASER'S SOLE RISK, EVEN IF ARBITRARY PRECISION HAS BEEN INFORMED IN WRITING OF SUCH USAGE.

Resale of Arbitrary Precision products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by Arbitrary Precision for the Arbitrary Precision product or service described herein and shall not create or extend in any manner whatsoever, any liability of Arbitrary Precision.

www.arbitrary-precision.com