UNIVERSAL INTERFACE

PROGRAMMING MANUAL

UNIVERSAL PROGRAMMING INTERFACE
MODEL: UCI MK1

CONCEPT

This product is designed to provide a method of attaching a paging transmitter to any device with a data output port. Usually this will be a serial port, but there are 14 digital input/output lines which can be used to implement a parallel port if required.

The product has 2, 9-way 'D' connectors, conforming to the standard RS232 pin configuration. One of these connectors is for the panel or driving device (Com2), the other for the transmitter or device to which the processed data is to be sent (Com1). Each port has its own commands for sending and receiving data, the active port will be switched as required.

A simple Basic language has been developed which will allow the unit to capture, process and retransmit the data according to the program content. This product will suit either simplex or duplex protocols as both ports are fully bi-directional. Baud rates can be set independently under software control. This will allow the system to communicate with the panel at one rate, and to increase the rate when sending data to the transmitter which may support much higher rates than most panels.

OPERATION

When first powered up, the system enters the programming menus, of which there are four with a variety of different commands for capturing, storing and analyzing the received data. Portions of the incoming text may be stripped or added to, prior to onward transmission of the formatted text. A table function allows the user to search for particular keywords or strings, which can then be replaced or used to determine which messages are acted upon, and those which are not.

The operator writes a list of commands which are carried out in sequence when the program is 'RUN'. This can be tested fully via the terminal before it is committed to memory. Once filed away, it is possible to force the system to 'Auto-boot' into the saved file when the system is powered up. A number of files may be stored, and the active program can load and run files at will.

To aid in the development of a program it is possible to steer all output to a single port. This allows the programmer to monitor the output data normally destined for the transmitter or driven device.

The program can be LISTed, and additional program lines may be POKEd into the listing. There is also a facility for uploading a short form listing to the unit via the serial port. Table data is also loaded in this manner, though there is a facility for appending single messages to an existing table.

NUMERIC VALUES

A lot of the commands included in this product offer text or numeric input types. The numeric values relating to the input port are the actual numeric value of the character in question. This method ensures that all valid 8-bit values can be used, since these values are saved as their printable equivalent. It also ensures that the programs can be transferred via terminal software without certain characters being trapped as control codes. Values may be entered as decimal or hexadecimal, but hex input must be preceded by the '$' sign to be identified as such. ($F8). Hexadecimal is most useful for the logical functions related to the parallel ports.

PLEASE REFER TO THE ASCII TABLE AND HEX/DECIMAL CONVERSION CHARTS INCLUDED AT THE BACK OF THIS MANUAL.

PANEL PORT INSTRUCTIONS

The COM2 port is for connection to the panel or 'driving' device, the output of which is to be processed by the program. To achieve this, it is necessary to provide a number of commands which collect data from the port until a defined event occurs. This event could be a specific number of characters collected, or recognition of a particular character or string. The data received up to this point can be saved or dumped, depending upon the chosen option. Alternatively, rather than executing a loop which continually receives data, it is possible to get a single character or value which can be analyzed immediately using the comparison functions.

GET:

This command is followed by a number of prompts which offer alternative ways of treating the data received, either DUMP or SAVE it. As well as this, the condition under which the process is stopped must be defined. These alternatives are offered when the GET command is typed in as a program step, and added to the saved command data.

The following list shows the complete set of GET options:

GET SAVE/DUMP (X)	GET (X) CHARACTERS
GET SAVE/DUMP TILL "STRING"	GET TILL "STRING" IS SEEN
GET SAVE/DUMP TILL (X)	GET TILL (X) IS SEEN

The (X) represents a numeric value between 0 and 255, whilst all ASCII characters lie within this range, not all of them are readily accessible from a keyboard or terminal program. The ability to define the absolute value provides the user with a guaranteed method of representing the whole range of numbers obtainable from a serial port.

GETA:

This command will get a single character from the serial port into the comparison register 'A'. At this point the various 'SKIP IF' commands are used to determine what to do next. This function offers a greater range of alternatives since multiple SKIP IF commands allow the program to branch depending upon the value of the received character. It is also possible to identify whether the character lies within a range of values, for instance is it a valid ASCII character which the pager can display. The following example illustrates this concept:

```
[START]
CLEAR
GETA
SKA<>'A'
JUMP [ITSA]
JUMP [START]
[ITSA]
```

This program will continue to receive characters until the (Capital) letter 'A' is seen, at which point it will branch to another part of the program. The most obvious use for such a scenario is a menu where the user is prompted for input, and a branch made dependent upon the response.

FIND:

Used to position the pointer at a particular value or word, this command prompts for the type of data to find, either a character or string of text. One use for this is to position the pointer at a word and then delete everything up to this word using DELBF, which will remove everything up to the pointer position.

RESPOND:

This command provides a means of sending data to the panel port as well as receiving from it. Some paging protocols require a response to certain output data, for instance as an acknowledgement. ESPA444 is such a protocol which 'polls' devices connected to the bus. RESPOND will prompt for the type of data with which to respond, be it a single numeric 8-bit value (0-255), or a string of characters.

BAUD:

This allows the program to set up the baud rate of the serial ports, the user is prompted for the number of bits, parity and stop bit count, as well as the baud rate to be used. It should be noted that speeds below 9600 make the programming interface very slow to use. It is recommended that the baud rate is temporarily set to the highest supportable value when using the programming menu's. (The listing of the current menu commands may be suppressed by turning on DIP switch 8.) It is possible to set the panel baud rate at the start of a program, and then to reset it to the higher rate when the program completes and prior to returning to the programming menu's.

RUN:

This command will run the program currently held in memory. It is used to test the program as it is written, and is a 'real-time' instruction in that it executes immediately and is not a saved command.
A program can be stopped by sending an escape (ASCII27) to the unit, this is providing it is not in one of the loop functions such as GET TILL. The escape character is trapped between instructions, so will not be seen until the program moves past the loop function to the next command.

CLEAR:

This command is used to clear out the text buffer and comparison register. It should be issued at the start of any attempt to capture data, at which point it will remove the old contents. Normally it will be the first instruction in the program.

NEW:

This command resets all pointers and clears out the program memory ready for a new program.

CLEAN:

This command will delete all files in the filing system memory, it should be used cautiously, and will prompt twice before performing the erasure. The filing system uses Flash memory and will take time to clean the whole device. Flash devices are very secure, the particular type used have a software data protection scheme built in which should prevent any possibility of accidental loss of data. When the CLEAN operation is complete, the command menu will reappear.

POKE:

This is the only way to insert additional lines into a program listing. The new command is entered and occupies the last line of the program. The command POKE is then issued, and will prompt for a line number into which to POKE the command. Because line numbers are generated at the time of listing, they are open to change as other lines are deleted or poked. This differs from normal Basic conventions where each command in the list is fixed to its line number, and reference to a command line is through its line number.

SEND:

This command is used to send data to the transmitter port, either individual characters, text strings or the text buffer can be sent, the user will be prompted for these options at the time it is invoked.

FILING SYSTEM COMMANDS AND FUNCTIONS

As with the line numbers assigned to a command listing, the index numbers which appear against filenames are not fixed to that file, it is generated at the time of listing the files and is used to index the file for loading, deleting or chaining. If a file is deleted, the numbers will regenerate and may change.

FILES:

This command provides a direct route to the file menu, repeated carriage returns cycle through the various menu's also.

LOAD:

When invoked, this command will list the contents of the filing system, showing the names, and their numeric position within the filing system. (Program length is also shown in brackets). A prompt is issued to select a file to load, this is entered as the file number, not the filename. The selected file will be loaded into memory, it will not be run until such an instruction is issued. If the command memory already has a program loaded, it will issue a prompt to confirm that you wish to over-write the contents.

SAVE:

This command prompts for a filename then saves the contents of the command memory under the chosen name. Filenames have no limits on length at this time, though it is prudent to keep them concise. Any character up to the carriage return is included within the name, this means that any character available from the keyboard is valid including punctuation and symbols.

DELF:

This command is used to remove files from the filing system memory. A list of files is printed to the screen with an index number, it is the number which should be entered to select the file to delete.

CHAIN:

This function is used to allow programs to call up other programs from within the filing system. For example, the !BOOT program if so selected by the relevant DIP switch, forces the system to 'CHAIN' the boot file at start-up. Programs can be chained by the currently running program which will then be cleared from memory and the new one loaded and run. This process has many uses, it allows a program to be chopped into smaller files which sequentially run and then load the next file, or this sequence could be made conditional on data input from the driving device.

LISTF:

This command produces a list of the files held in the filing system. This is true only when in the Files menu, there is another command called LIST which, when issued from any other menu will produce a program listing. Files are displayed sequentially, according to the order in which they were saved, a prompt will be issued prior to the listing being scrolled past the screen.

MENU:

This command exits the files menu and returns to the first command menu. This can also be achieved by entering a carriage return from within the files menu which will generate a command error and default back to menu 1. This is true of all menu's, hitting enter will cycle to the next menu in the sequence, whilst issuing the error beep.

TABLE FUNCTIONS

LOADT:

This command is used to input table data via the panel port. Tables can be constructed with a text editor and sent to the filing system using this function. Table entries must be separated with a carriage return, linefeeds, if present, will be stripped. A table must end with the binary value zero, this informs the system that the end of the file has been reached.

LISTAB:

This sends a list of the table data to the screen, assigning index numbers as it goes. This listing will also show any embedded commands which may be present in the table. (See 'embedded commands later in this section).

MATCH:

This is the primary way of using the data tables held in memory. This is a 'skip' function in that if a match is made, the following program command will be skipped. This provides a method of confirming the match, it also enables alternative program flow depending upon whether a match is successful or not.

EMBEDDED COMMANDS:

As well as the skip function, it is possible to force certain actions when a match is found. This is achieved by inserting a particular character after the string which is to be matched, followed by a second string on which this action will take place. There are 4 embedded commands as follows:

'>': Insert following string after matched string in buffer.
'<': Insert following string prior to matched string in buffer.
'=': Replace matched string in buffer with following string.
'-': Strip matched string from buffer.

Obviously there must be a second entry in the table for most of these commands, this follows immediately after the embedded command and is terminated with a carriage return. On listing such a table, the format will resemble the following example:

THIS IS THE STRING TO MATCH
>THIS WILL BE PLACED AFTER THE MATCHED STRING
ANOTHER STRING TO MATCH
=THIS WILL REPLACE THE FIRST ENTRY

Tables do not have to contain embedded commands, it can be made up of single entries in which case the skip function will be used to determine what action is taken. Single text entries and embedded entries can be mixed at will. From the above it will be seen that no message in the table should begin with one of these embedded characters, this will result in misinterpretation of that entry. The characters have been chosen because of their unlikely first use in a message entry.

All table data must be in printable ASCII characters, this rule ensures that firstly it is not trapped by terminal software as a control code, and also provides a means of identifying end of messages and files through the use of the forbidden characters. The only exception to this rule is the zero which defines the end of the table. In fact any value outside of the range 32-127 will result in termination of the loading process. So any non-ASCII character can be used to terminate the loading, but the search functions within the software will look for the zero to mark the end of the table.

DELBF:

This command will delete everything between the start of the input buffer and the current buffer pointer, moving the remaining data down to fill the gap. This would generally be used to clear out any unwanted text up to a recognized word or character.

LOOP:

For repetitive functions there are 5 loop registers available. When this command is used, the programmer is prompted for the loop number and the value to place in it. These registers are linked to the DECSZ command following and will provide counting facilities, or software loops.

DECSZ:

This command is inextricably tied to the loop registers described above. This is a skip function and the defined loop register is decremented, if the loop counter has reached zero, the next program command is skipped. This is similar to the FOR/NEXT functions in standard basic and can be used to either repeat a sequence of program steps a predefined number of times, or to generate software delays, the period of which depends upon the number and type of commands being repeated.

LABEL:

For navigating around the program it is necessary to provide a means of 'marking' certain places within the program. The label function achieves this, offering either a numeric or text option for the label. Text labels may contain any printable character, and may be any length though the shorter the better in general. A label within the program is enclosed within square brackets. JUMP's or CALL's are made to labels, the program will pass through labels if not directed to them, so they can be placed anywhere.

JUMP:

This command allows program flow to be directed to the labels as defined above. If the label is not found the command is ignored.

CALL:

Like a JUMP, this command will direct program flow to a label, but it will save its current position first. When it encounters the next RETURN instruction it will branch back to this saved position in the program. Subroutines are sections of program which may need to be carried out a number of times throughout the program, this allows them to be entered once, with the CALL and RETURN providing the gateway to them. This product does not support 'nested' subroutines, that is, you cannot CALL a subroutine from within a subroutine, as there is only one saved address.

RETURN:

This command returns the program flow to the last saved position when a CALL was issued. It is the last command in a subroutine. A return issued without a previous CALL will crash the program, since the address in the stack will be undefined.

STRIP:

Sometimes panel data is padded with spaces to format the text for the destination printer or screen, these spaces are not usually required for the pager, and this command gives the user the opportunity to strip consecutive occurrences of a character. The character to strip will be requested at the time the command is entered.

SWAP:

This function allows the user to swap one value for another in the text buffer. This may be a single character, or a string of characters. The user will be prompted firstly for the data to remove, and then the data to exchange it with. If the second value is not defined, or a carriage return is entered in response to the prompt, then the effect is to strip the first value, replacing it with nothing.

SKIP:

This command is used to 'jump over' or reposition the buffer pointer a specified number of characters ahead.

PULL:

This function offers the ability to receive data from the transmitter port, a single character at a time. The value is placed in the 'A' comparison register. This would be used to provide a duplex comms facility between the unit and the transmitter or device it is driving. Generally, responses from a transmitter are of the single-character kind, such as an acknowledge or negative acknowledge, in which case this function will suffice to provide a means of confirming that the data has been received okay. If the response is negative, then the data can be re-sent until the required positive response has been received.

START:

This command places the current buffer pointer at the beginning of the buffer plus the entered offset, so START(0) will set it to the beginning, and START(5) will place it 5 characters from the start. This is generally used to add a pager command prefix to the contents of the buffer prior to sending the data to the transmitter.

INSERT:

When entered, the user will be prompted for a numeric value or string of characters to insert. The insert will take place at the current buffer pointer position.

PUTAB:

This gives the programmer the ability to APPEND a message onto the end of the table in memory. This is the only way to add messages other than to load a new table using LOADT.

WAIT:

This command provides a means of introducing delays into the program, which will sit and do nothing for the specified number of 'ticks'. A value of one is equal to some 50mS, so the longest delay which can be generated with a single wait command is 1 second. (255*50mS).

STORA:

This stores the current value in the comparison register 'A' to the input buffer at the current buffer pointer position. It can be used to selectively store data when used with the comparison instructions. For instance, if numeric pagers are used on a system, it would be possible to extract and store only the numeric content of a message for onward transmission.

SETTING THE DIP SWITCHES

SWITCHES 1-4:

These switches are used to pre-select a page number in the filing system memory. Normally, only the first 3 are required (For a 29C020), unless the device fitted is the larger 29C040. The former provides 256K or 8 pages of memory, the latter has 512K or 16 pages. When the PCB is powered up, the micro looks at these switches to determine the page into which it should look for files and tables. This option allows a number of different program files to be loaded into separate pages, the DIP's then select which program to run at power-up.

The command PAGE is provided so that a program may select its own page number, perhaps to CHAIN another file held elsewhere in memory. The current Table data is also related to the page which the unit is currently in. If the value entered into the PAGE command is 255, then the command will look at the dips for the page number setting.

SWITCH 5:

There is a facility to 'boot' the system into a particular program at power-up, this file should be named '!BOOT', and it should reside in the page selected by the dip switches. With this switch 'ON' a search is made for this program and if found it will be loaded and run. If the switch is 'OFF', this process is bypassed and the programming menu will be displayed instead.

SWITCH 6:

For developing and writing programs via a terminal, it is useful to have the transmitter port data sent to the terminal instead of that port. This switch, if 'ON' will override the units attempts to write to the transmitter port, and send it to the panel port instead. This feature obviates the need to run two serial ports for programming. If the program under development is a duplex one where the panel is bi-directional, this may need to be switched off, otherwise the transmitter data will be sent to the panel.

SWITCH 7:

This switch sets the default baud rate of the system when powered up. 'OFF' puts the system into 1200,n0 parity, 8 data bits, and 1 stop bit. When 'ON' the higher rate of 9600 is chosen, a much more practical speed to run the programming interface at.

This protocol can be changed from within a program, or by entering a single line program and running it. This is useful if the panel and transmitter need to be run at separate rates, or to select the fastest possible rate whilst programming, and dropping into the panel rate when the program is run. If testing a program by running it, and the rate is too slow for programming, then make the last command reselect the faster rate before the program finishes.

SWITCH 8:

This switch disables the printing of menu's during the programming stage. It may speed up the process for those who are familiar with the command set. When 'ON' menus are suppressed, when 'OFF' they are not.
When the menu's are switched off, a beep and the prompt '>' is all that is printed to the screen. Note it is not possible to identify whether you are in the files menu or not when menus are suppressed, but the command FILES will take you there from the other menu's.

PORT INSTRUCTIONS

There are 14 input/output lines taken to the 25-way 'D' connector on the PCB. These can be used to construct a parallel port if required, or as general purpose switched lines. Each one is individually settable as an input or an output using the SMA/SMB functions described below. These lines are connected directly to the microprocessor and have limited drive capability. They should not be used to drive LED's or relays without some form of amplifier or driver in between. As a guide, no more than 5mA should be drawn form any of these lines, damage may occur if this level is exceeded.

The 14 lines are made up of 8 which represent PORT A, and 6 which are called PORT B. Port A can be used as an 8-bit data bus, and the 6 lines of port B provide ancillary control lines. Together these can be made to drive a bi-directional parallel port, though its speed will be relatively slow. A number of low-level logical commands are used to set the states of the lines, or to read them in.

SMA/SMB:

This command sets the direction of the lines on each port, where a '1' represents an input, and a '0' is an output. For port B, only the lower 6 bits of the 8-bit value are significant, the other two will be mapped out.

SPA/SPB:

This command writes the specified value directly to the port. Only the lines set as outputs will be affected by this instruction.

ORPA/ORPB:

This command is used to set individual lines on the ports high. Those bits in the value entered which are high, will result in the corresponding port line going high. (If set as output).

ANPA/ANPB:

This is the way to clear individual port lines, any bits in the value entered which are low, will set the corresponding port bits low. For instance, ANPA(0) will clear all lines on port A to zero, whilst ANPA(254) will clear only the lowest bit.

INPA/INPB:

This command loads the specified port into the A register.

ATPA:

This command provides a means of directing stored data to Port A. It is the contents of the 'A' comparison register which are written, and since this register can hold data from the serial input port, it is a way of echoing serial input to another device connected to the parallel port lines. Either a printer or an LCD screen could be driven in this way, using the port A lines as the data bus, and the port B lines as the control signals. To achieve this, the user must consult the technical data for the device they wish to use. The port configuration has been designed to offer most of the Centronics functions, though it should be noted that its relative speed, compared to a computer will be very slow. There is a 16-way connector strip on the PCB which is laid out to suit the Hitachi range of LCD Alphanumeric modules. (LM093 etc)

The software overhead required to drive these ports is considerable, each bit must be manipulated in sequence, and a program to drive an LCD screen will be relatively long. Printers have many control lines indicating paper out etc which must all be checked prior to sending data to the port. These functions have been offered because they are viable, whether they are practical depends upon the application and the user's experience at assembler level programming.

COMPARISON/SKIP FUNCTIONS

The commands revolve around the comparison register 'A'. All these commands are SKIP functions, that is, if the specified condition is met, the following command is skipped. This allows the program flow to be modified in response to the type of data captured, or to set up program loops which terminate on recognized characters or values.

SKA=:

This command will skip if the value in A is equal to the specified value. If this condition is not met, the next command will be executed normally.

SKA<>:

This function will skip a command if the A register is not equal to the specified value. This command can be used to perform jumps to particular places depending upon the input data. The clearest example would be a menu, where each choice leads to a specific action as follows:

```
[LABEL]
GETA
SKA<>(84)
JUMP [ITST]
SKA<>(83)
JUMP [ITSS]
JUMP [LABEL]
```

This sequence is looking for the characters 'S' and 'T', and will repeatedly input data until it sees either of them, at which time it will divert to another part of the program.

SKA</SKA>:

As above, these commands are self-evident, skipping on the condition of greater than or less than. The most common use for these commands, used together, is to determine whether a character lies within a certain range of values, for instance is it a numeric character between 0 and 9.

```
[LOOP]
GETA
SKA>(47)
 JUMP [LOOP]
SKA<(58)
JUMP [LOOP]
```

This sequence of commands will only complete when the value input from the port is a numeric character between 0 and 9.

The use of skip functions keeps the program complexity down and reduces the time taken to execute a given sequence of instructions. This is because, for the most part, these commands translate directly into a single processor instruction, rather than the usual multiple commands required to achieve such a calculation. For this reason it is recommended that the skip commands in their various forms should be mastered and used. An interpreter takes time to execute a particular command because though the command may be a single word, the steps required to carry out that function may be numerous at processor level. Using commands that are native to the processor means that these steps are minimized and the time taken to execute a given function is greatly reduced. Since the aim of the product is to capture and respond to data, it is important that it is away from the serial port for as short a time as possible.

LOADING A PROGRAM IN SHORT-FORM.

LOADP:

This command will attempt to load a pre-written program from the serial port in short-form. Whilst most commands are entered as full words, they are saved in a much smaller form to cut down program size. This command will allow the program to be constructed on a text editor, and then downloaded to the unit via the serial port. Each command must be terminated with a carriage return and the end of file will be marked with a zero, or any character of value less than 10 decimal. No line numbers should be included, these are generated internally when a file is listed.

SHORT-FORM COMMAND STRUCTURE:

ATPA	ATPA	PLACE CONTENTS OF A TO PORT A
ANPA	ANPA(X)	LOGICAL AND PORT A WITH (X)
ANPB	ANPB(X)	LOGICAL AND PORT B WITH (X)
BAUD	BAUD(9,N,8,1)	SET BAUD: SPEED,PARITY,BITS,STOP
CLEAR	C	CLEAR BUFFER
CALL	C[LABEL]	CALL SUBROUTINE AT LABEL
CHAIN	CHAIN(X)	LOAD AND RUN FILE NUMBER (X)
DELBF	DELBF	DELETE BUFFER UP TO POINTER
FIND	F(X) OR F"TEXT"	FIND AND POSITION POINTER AT
INPA	INPA	GET CONTENTS OF PORT A TO A
INPB	INPB	GET CONTENTS OF PORT B TO A
INSERT	I(X) OR I"STRING"	INSERT AT CURRENT POINTER
JUMP	J[LABEL]	JUMP LABEL NAME
SKIP	K(X)	SKIP OVER (X) CHARACTERS
SWAP	S(X),(Y) OR S"TEXT","TEXT"	SWAP ONE THING FOR ANOTHER
ORPA	ORPA(X)	LOGICAL OR PORT A WITH (X)
ORPB	ORPB(X)	LOGICAL OR PORT B WITH (X)
ONPA	ONPA(X)	SKIP IF SET BITS OF (X) ON PORT A ARE 'ON'
ONPB	ONPB(X)	SKIP IF SET BITS OF (X) ON PORT B ARE 'ON'
OFPA	OFPA(X)	SKIP IF SET BITS OF (X) ON PORT A ARE 'OFF'
OFPB	OFPB(X)	SKIP IF SET BITS OF (X) ON PORT B ARE 'OFF'
SPA	SPA(X)	PUT (X) TO PORT A
SPB	SPB(X)	PUT (X) TO PORT B
SMA	SMA(X)	SET DIRECTION OF PORT A
SMB	SMB(X)	SET DIRECTION OF PORT B
RETURN	U	RETURN FROM SUBROUTINE
MATCH	MATCH(X)	MATCH BUFFER WITH TABLE
RESPOND	R(X) OR R"TEXT"	SEND TO PANEL PORT (X) OR "TEXT"
SEND	EB OR E(X) OR E""	SEND (X) OR "TEXT" OR BUFFER
LABEL	[LABEL]	LABEL NAME
LOOP	O(X),(Y)	LOAD LOOP(X) WITH (Y)
DECSZ	DECSZ	DECREMENT LOOP(X) SKIP IF 0
START	X(OFFSET)	BUFFER POINTER TO BASE+(X)
GETA	GA	GET TO A FROM PANEL PORT
STORA	PA	PUT CONTENTS OF A TO BUFFER
STRIP	P(X)	REMOVE CONSECUTIVE (X)'S
SKA=	SKA=(X)	SKIP IF A=(X)
SKA<>	SKA<>(X)	SKIP IF A<>(X)
SKA>	SKA>(X)	SKIP IF A>(X)
SKA<	SKA<(X)	SKIP IF A<(X)
GET	GSTN(X)	GET AND SAVE TILL NUMERIC VALUE (X) IS SEEN
	GDTN(X)	GET AND DUMP TILL NUMERIC VALUE (X) IS SEEN
	GS(X)	GET AND SAVE (X) CHARACTERS

	GD(X)	GET AND DUMP (X) CHARACTERS
	GST"TEXT"	GET AND SAVE TILL "TEXT" IS SEEN
	GDT"TEXT"	GET AND DUMP TILL "TEXT" IS SEEN
PAGE	PAGE(X)	SET CURRENT PAGE NUMBER

EXPORT:

This command will send out the current loaded program to the serial port, but unlike LIST, it will not generate line numbers. The purpose of this is to make the file re-loadable with the above LOADP command. A program saved from a LISTing will have to be edited to remove line numbers prior to trying to load it back, EXPORT removes this necessity.

DECIMAL TO HEX CONVERSION

DECIMAL	HEX	ASCII	DECIMAL	HEX	ASCII
10	0A	LF	62	3E	>
11	0B	VT	63	3F	?
12	0C	FF	64	40	@
13	0D	CR	65	41	A
14	0E	SO	66	42	B
15	0F	SI	67	43	C
16	10	DLE	68	44	D
17	11	DC1	69	45	E
18	12	DC2	70	46	F
19	13	DC3	71	47	G
20	14	DC4	72	48	H
21	15	NAK	73	49	I
22	16	SYN	74	4A	J
23	17	ETB	75	4B	K
24	18	CAN	76	4C	L
25	19	EM	77	4D	M
26	1A	SUB	78	4E	N
27	1B	ESC	79	4F	O
28	1C	FS	80	50	P
29	1D	GS	81	51	Q
30	1E	RS	82	52	R
31	1F	US	83	53	S
32	20	SPACE	84	54	T
33	21	!	85	55	U
34	22	“	86	56	V
35	23	#	87	57	W
36	24	$	88	58	X
37	25	%	89	59	Y
38	26	&	90	5A	Z
39	27	‘	91	5B	[
40	28	(	92	5C	\
41	29	)	93	5D	]
42	2A	*	94	5E	^
43	2B	+	95	5F	_
44	2C	‘	96	60	‘
45	2D	-	97	61	a
46	2E	.	98	62	b
47	2F	/	99	63	c
48	30	0	100	64	d
49	31	1	101	65	e
50	32	2	102	66	f
51	33	3	103	67	g
52	34	4	104	68	h
53	35	5	105	69	i
54	36	6	106	6A	j
55	37	7	107	6B	k
56	38	8	108	6C	l
57	39	9	109	6D	m
58	3A	0	110	6E	n
59	3B	:	111	6F	o
60	3C	;	112	70	p
61	3D	<	113	71	q

DECIMAL	HEX	ASCII
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	|
127	7F	DEL

The values between zero and 10 have been omitted, these are deemed control codes as are some of the above. Values above 127 are usually graphics characters and will not normally appear on panel output.

LCD CONNECTOR PINOUT (where fitted)

1	0V
2	5V
3	VO
4	RS
5	RW
6	EN
7	D0
8	D1
9	D2
10	D3
11	D4
12	D5
13	D6
14	D7
15	LED+
16	LED-

LAYOUT OF SERIAL PORT CONNECTORS:

COM2 (PANEL PORT)		COM1 (TRANSMITTER PORT)	
1	NC	1	NC
2	RX	2	RX
3	TX	3	TX
4	DTR	4	DTR
5	0V	5	0V
6	DSR	6	DSR
7	RTS	7	RTS
8	CTS	8	CTS
9	NC	9	VIN FROM EXTERNAL SUPPLY

THE VIN (PIN 9) ON COM1 IS DESIGNED TO BE CONNECTED TO A SCOPE TRANSMITTER SUPPLY. IT SHOULD NOT BE CONNECTED WITHOUT REFERENCE TO SCOPE.

LPT1 PARALLEL PORT PIN OUT

1	NC	13	PB2	25	GND
2	PA0	14	NC		
3	PA1	15	PB3		
4	PA2	16	PB4		
5	PA3	17	PB5		
6	PA4	18	GND		
7	PA5	19	GND		
8	PA6	20	GND		
9	PA7	21	GND		
10	PB0	22	GND		
11	PB1	23	GND		
12	NC	24	GND		

Scope's policy is one of continuous improvement and specifications are subject to change without notice.

****