6th May 1974

MINSYS

Preliminary Reference Manual

Contents

1. Introduction

    1.1 New Features in MINSYS

    1.2 The MINSYS Stack

    1.3 Some Notes for BASYS Programmers

    1.4 MINSYS Environment

2. Expressions in MINSYS

    2.1 Numerical Expressions

    2.2 String Expressions

    2.3 Numeric Variables and Functions

    2.4 Arrays

    2.5 Access to System Information

3. Synopsis of MINSYS Commands

    3.1 Storage

    3.2 Editing and Housekeeping

    3.3 Assignment

    3.4 Conditionals

    3.5 Transfer of Control

    3.6 Procedures and Iteration

    3.7 String Operations

    3.8 Peripheral Transfers

MINSYS Preliminary Documentation

1. INTRODUCTION

MINSYS is a recent member of the line of Essex extended BASICs (variously known as BASYS, QUASIC, QUASAC and AIMS). This family of languages takes Dartmouth BASIC as a model for syntax but extends and modifies it to provide a complete range of facilities for system and major application programming.

The major objectives are:

(a) To allow tasks normally requiring assembly language on a minicomputer to be carried out in a high-level language even on small configurations (BASYS interpreters are normally 4k words or less).

(2) To provide interactive program development facilities enabling very rapid system development (BASYS programs may be freely stopped, interrogated, edited, and continued).

(3) To support interactive, conversational systems by providing extensive string-handling and contextual editing within the language.

(4) To support large complex file structures by providing efficient and direct access to backing stores.

Dartmouth BASIC was taken as a model because it has been so successful in use by non-programmers. This appears to arise partly from the line numbering giving natural interactive editing, and from the syntax where a single, meaningful command establishes the main action to be taken and the residual syntax for numeric or string expressions is simple. These two features have been retained in MINSYS and some BASIC command names have been carried over - however, in detail BASIC and MINSYS are very different.

A BASYS or MINSYS program appears similar to extended BASIC and consists of a set of numbered lines each containing one or more commands

separated by colons, e.g.

```
100 INPUT $5 :UNLESS $5 = 'YES' :PRINT $5 '?' :GOTO 200+k
```

One major feature is that character strings may also be held in the same numbered "program" lines. This simplifies the run-time system and enables string constants and dynamic strings to be simply manipulated. Other conveniences are that on program entry only the minimum unambiguous command string need be entered (e.g. P5 instead of PRINT 5) and all commands are available as directly executed or as a stored program. Debugging is simple since programs may be stopped by a keyboard command, modified, direct commands executed, and program restarted from where interrupted.

Other BASYS/MINSYS extensions include:

Improved string-handling modelled on SNOBOL - handles arbitrary-length, dynamically changing strings with full ASCII 7-bit character set - string relational operators allow for anchored and embedded searches, and tests for string equality, inequality or telephone-directory comparisons;

Numerical expressions are accepted wherever a number could occur, and these may contain logical and relational operators. Hence all transfers of control may be computed;

Several commands may follow on one line and any statement may follow a conditional. Many commands have implied conditionals such that execution along the line continues only if the command is successfully executed.

1.1 New Features in MINSYS

For those users who have met BASYS in one of its forms the following notes review the main changes in MINSYS. The main objectives of the new features have been:

(a) To improve the management of large system developments - the one-level name system of BASYS with all names having global scope through all subroutines, overlays, etc. is a source of problems when large systems are being developed and even more so when they are being modified after periods of use. Use of BASYS is intended to minimize documentation requirements and having to list name usage in different packages to avoid conflicts is highly repugnant. MINSYS provides a block-structured name-space in which names may be given scope only during procedure calls. It is intended that these calls be used freely to control the names in use.

(b) To reduce the use of the GOTO command to major transfers of control only. Currently BASYS often requires the use of GOTO to skip over parts of a conditional expression. This impedes both editing and readability. The new commands ELSE and AND are conditionals dependent on the results of previous conditionals. Their use gets rid of many GOTO's and makes the conditional structure far clearer.

(c) To improve the syntax and power of the string-handling facilities. The BASYS PUT command provided new language capabilities for string-handling. It's syntax, however, goes against the principles of meaningful simplicity discussed earlier and it compresses many different facilities in one command. MINSYS provides a more extensive range of facilities including those of the BASYS PUT, but in a range of commands. In particular any string may be used as source or destination and the source string, the pointer to it, and the destination string may be freely charged.

Finally, please note that the development of BASYS is an experiment

in providing rapid, simple interactive programming. The new features of MINSYS are intended as improvements along these lines. Critical comments and further suggestions are welcome, particularly in the near future when MINSYS is still under development.

1.2 The MINSYS Stack

MINSYS makes far more explicit use of its run-time stack than do previous versions of BASYS and an outline of its mechanism will be an aid to understanding several new facilities. The data-structures of MINSYS are a linked list of character strings (the program and $-lines - note not arrays in MINSYS) and a stack of named simple variables and arrays.

New arrays and variables are put onto the stack as they are created and the stack is always searched from the latest item. This ensures that if multiple items with the same name are on the stack then the one accessed is that most recently created.

Simple variable names are of the form a letter, or a letter followed by letter or a number. Array names are a letter only - the following opening bracket serves to indicate an array. Note that there is no relation between the array X ( ) and the simple variable X.

The simple variable Q also acts as a stack marker establishing the current local context which consists essentially of all simple variables and arrays created since Q. A simple variable or array element within this local context is said to be local to the program section executing at the time.

The variable Q is not normally created by an assignment but by the command: BEGIN - the MINSYS start of block command. Three further commands may be used to clear the stack back to, and including, the last Q marker: BACK - the MINSYS procedure return; NEXT - the MINSYS iteration return; and END - the MINSYS end of block command.

Ordinary simple variables are mainly created by one of two commands:
LET - the normal assignment;
LOCAL - the local assignment which behaves as LET but only searches the

local environment for a previous copy. Thus LOCAL is guaranteed to assign to a local variable, setting one up if necessary. LET may assign to an existing non-local variable - however, note that if it sets up a new variable it will be local.

Arrays are set up by the ARRAY command which always creates a new array, and hence a local one.

1.3 Some Notes for BASYS Programmers

MINSYS differs from BASYS in many important aspects. The following notes are an aide memoire to possible confusions in transferring from BASYS to MINSYS.

First similarities:

(1) Program lines numbered 0 to 32k.

(2) String lines as program lines referred to as $NE.

(3) Simple variables letter, or letter followed by letter or digit.

(4) Numeric expressions and string expressions very similar.

(5) LET, IF, UNLESS, GOTO, RUN, DO, STOP, PRINT, INPUT, LIST, LOOP, GARB, commands very similar.

(6) Input/output to numbered channels referred to as #NE.

Now the major differences:

(a) The ARRAY command is executed in MINSYS. Arrays themselves are not program lines but are set up by the ARRAY command in the stack.

(b) The GOSUB command has become DO a line commencing with BEGIN and RETURN has become BACK. This emphasizes some changes:

  (i) The calling address is held in the simple variable stack as the variable Q. It can be changed by assigning to Q.

  (ii) Arrays and simple variables created by the procedure are cleared on return.

  (iii) Parameters may be passed to the procedure by value or by reference. These are assigned to local variables through the BEGIN command. If parameters are not passed to some of these variables they take the default value zero.

  (iv) BACK is conditional if followed by a numeric expression. BACK NE only causes a return and stack clear if NE is non-zero, e.g. BACK X = 5 only returns if X = 5.

An example of a procedure call with parameters passes is:

```
 50  [CALLING SEQUENCE
 60  LET  X = 1  Y = 2  Z = 3
 70  PRINT  @4  X  Y  Z
 80  DO  100  X  %X  Y
 90  PRINT 'BACK'  :DO  70
 99  STOP
100  BEGIN  Q0  Q1  Q2  Q3  : [PICK UP PARAMETERS
110  PRINT 'BEGIN
120  PRINT  Q  Q0  Q1  Q2  Q3
130  LET  Q0 = 0  :DO  70  : [LOCAL PARAMETER VALUE
140  LET  Q1 = 0  :DO  70  : [LOCAL PARAMETER REFERENCE
150  LET  Y = 10  :DO  70  : [NON-LOCAL
160  LOCAL  Z = 20  :DO  70  : [NEW LOCAL
170  BACK
RUN
1  2  3
BEGIN
 80  1  1  2  0
(note 80 is calling line, 0 is default when none passed)
1  2  3  (no change)
0  2  3  (reference accesses X)
0 10  3  (Y may be accessed directly)
0 10 20  (locally Z is 20)
BACK
0 10  3  (but restored on return)
```

Array elements may be passed to simple variables:

```
DO  100  %A(7,3)
```

makes Q0 a reference to the 3-byte variable commencing at element 7 of array A. This is a clean and fast way of accessing items in a record.

(c) The PUT command has been replaced by a new series of commands, capable of similar effects but generally more powerful and readable:

(i) PUT SE now creates a string $QS (where QS is a system variable) containing SE. It also zeros the system variable QP which is a pointer to a character within this string.

(ii) QS/QP define a source string and a position within it and may be used freely in arithmetic expressions and assignments.

(iii) AS $NE creates a null string $NE and sets the system variable QD to NE. This is the destination string to which output will be appended. Again QD may be freely used in arithmetic and assignments.

(iv) WITH SE appends SE to $QD.

(v) TO, FROM, and SEEK are macro template search commands specifying a template to be looked for in the source string from the QP'th character. If the pattern is found then QP is updated to point to the next character beyond it. Also, if specified, the characters before, and between parts of, the template may be appended to the destination string $QD. If the template does not fit then neither QP or $QD is affected (note the difference from the BASYS PUT).

For example:

```
50  PUT  $6 = 'AB' > $7 'KJ'
```

is now:

```
50  PUT  $6 : FROM 'AB' : AS $7 : TO 'KJ'
```

or, on several lines:

1.4 MINSYS Environment

MINSYS runs under the MINIC device-independent, multi-user operating system MUC (Multi-User Companion). It fully supports all operating system facilities - on disc-based systems MINSYS itself is used for all the operating system utilities, configuring, log in, log out, directory listings, etc. MUC is documented in "Multi-User Companion Programmer's Guide" (Micro-Computer Systems January 1974).

2. EXPRESSIONS IN MINSYS

2.1 Numerical expressions

A numerical expression (ne) is something that can be evaluated to produce a number. The following operators are allowed:

PRECEDENCE	OPERATOR	MEANING Arithmetic Operators
1	0	Ten to the power (alphabetic 0)
1	←	a ← b: open bit shift of a by b places left if b > 0, right if b < 0
1	↑	Exponentiation
2	-	Unary minus
3	/	Division
3	*	Multiplication
4	-	Subtraction
4	+	Addition

Relational operators

5	<	Less than	)
5	=	Equal	) and combinations of these
5	>	Greater than	)

String operators

6	>	Alphabetically greater than	)
6	=	Identical	) and combinations of these
6	<	Alphabetically less than	)
6	↑	a ↑ b: true if string-a begins with string-b	
6	←	a ← b: true if string-a contains string-b	
7	'or"	Quotes enclosing literal string	
7	$	Right-associative operator meaning string-name	

Logical operators

8	&	AND
8	/	Exclusive OR
9	!	Inclusive OR

2.2 String Expressions

A string expressions (se) is anything that can be evaluated to yield a string. String expressions are built up from one or more of the fields listed below. The value of the expression is formed by concatenating the values of the constituent fields.

FIELD	CORRESPONDING STRING
$(ne)	The string in dollar-line (ne)
'(string)'	(string) which may not contain '
"(string)"	(string) which may not contain "
(ne)	The value of (ne) converted to a string under control of the current output format and radix specification.
;	Carriage-return and line-feed
,	Has no value. May be used as a separator to resolve ambiguity.
%S(ne1) (ne2)	The (ne1)'th substring of line $(ne2).
%C(ne)	The ASCII character formed by taking the value of (ne) modulo 128.
%P(ne)	The program line (ne) - null if there is not one.
%R(ne)	Change output radix to (ne)&15. Right pack field with spaces normally, zeros if 32 added. Initial value of radix corresponds to value of +10.
@(ne)	If (ne)/100 = D, remainder W, then width of number formats is W characters with D after decimal point. Initial value of this parameter is picked up from system variable QF.

2.3 Numeric Variables and Functions

A simple variable in MINSYS is a letter (other than Q or O) or a letter followed by a letter or digit. The letter O is used as a unary operator equivalent to 10↑, i.e. O4 = 10,000. Names commencing with Q are mainly used for system variables and functions. Q alone is a simple variable but is used by MINSYS to hold return addresses and indicate the start points of local environments in the stack. Q0 through Q9 are ordinary simple variables - it is recommended that they be used as user system variables.

Some of QA through Q2 are functions taking an argument whereas others are system variables. Assignment is possible to the system variables but not to the functions. The following table indicates the role of the Q variables.

System Variables and Functions

QA	Var:	Modulus of remainder after division - valid only within same command as division.
QB	Var:	Break character after IO.
QC		
QD	Var:	Destination string for WITH, TO etc.
QE	Var:	Error code.
QF	Var:	Output format default.
QG	Var:	Garbage collection threshold
QH ne	Func:	ne = 0 how much space left.
QI	Var:	String IF counter.
QJ		
QK		
QL ne	Func:	Length of $-line ne.
QM		

QN ne Func: Next line after ne.

QO

QP Var: Pointer to source string.

QQ Var: DO QQ executed after an error.

QR ne Func: Random number if ne = 0, otherwise ne and reset random generator from ne.

QS Var: Source string.

QT ne Func: Tally of number of bits that are 1 in ne.

QU

QV ne Function: System variable pointer:

ne = 0 pointer to system JSB

ne = 1 pointer to current job JSB

ne = 256*J + 1 pointer to JSB of job J.

ne = 256*J + 16*C + 2 pointer to CSB of job J channel C.

QW Var: Where we are - line number of current line.

QX ne Func: X-position of "carriage" on channel ne.

QY ne Func: Y-position of "carriage" on channel ne.

QZ

2.4 ARRAYS

A MINSYS array consists of a block of bytes in the stack referenced by a name consisting of a single letter other than Q (note: the restriction to 25 arrays is no hardship since array names are distinct from simple variable names). Arrays are created dynamically by an array declaration which sets up an array on the stack (regardless of whether one with the same name already exists - as usual only the most recently created array with a given name can be accessed). Arrays are destroyed by operations which remove environments from the stack. Hence arrays may be created within local environments and destroyed when the local activity is terminated.

Arrays in MINSYS are used both as vectors or matrices of multi-byte numbers, and also as record buffers for data transfers. To cater effectively for all modes of use three types of array are provided:

(1) 1-dimensional array of multi-byte elements e.g. A(I) is the I'th 3-byte element of the array A of 3-byte elements (24-bit integers).

(2) 2-dimensional array of multi-byte elements e.g. X(I,J) is the (I,J)'th 5-byte element of the array X of 5-byte elements (40-bit integers).

(3) 1-dimensional array of variable-length elements e.g. E(I,J) is a J-byte element commencing at byte I of the array E (note: this is the classical BASYS array type - it is that needed for record structures).

2.4.1 Array Declarations

Arrays are created by the ARRAY command:

```
ARRAY  NAME  NE1   NE2   NE3
```

where the NAME is a letter naming the array; NE1 is the length of the array in bytes; NE2 is the element size in bytes (default value 1); NE3 is the multiplier for two-dimensional arrays.

When the ARRAY command is executed a block of bytes of length NE1 + 6 is allocated at the top of stack, the name and parameters NE1 (Length L), NE2 (Element size E), and NE3 (Multiplier, M) are placed in its header block and the remaining elements are zeroed. NE1 and NE2 are evaluated modulo 256 and no bounds check is made of them. In practice the element-size can be between 1 and the current precision (maximum 8 bytes). The multiplier can be between 0 and 255 with zero being a coded value signifying a 1-dimensional array.

2.4.2 Array Access

An element within an array block of length L is specified by a pointer, P, in the range 0 to L-1, and a size, S, in the range 1 to 8. There are two forms of access to any array - single-subscript and double-subscript. The subscript calculation performed in the double-subscript case depends whether the array multiplier, M, is zero (indicating a 1-dimensional array) or not. The following table shows the calculation for an array declared as ARRAY A L E M:

Access	Multiplier M	Pointer P	Size S	Remarks
A(I)	-	I*E	E	1-dimensional, fixed-size elements
A(I,J)	non-zero	(I*M + J)*E	E	2-dimensional, fixed-size elements
A(I,J)	zero	I	J	1-dimensional, variable-size elements

2.4.3 The "Store-Array" Q

The array Q is a system array consisting of the core store of the computer. It may be regarded as an ordinary array of length 65k with element size 1 and multiplier 0. Hence Q(I) is the I'th byte of the core. Q(I,J) is J bytes of the core commencing at the I'th byte. Beware that these are treated as 2-s complement numbers and the "sign-bit" extended. Mask out the top bits if necessary. Assignment to the array Q is a privileged operation.

2.4.4 Array Notes

(a) Apart from the convenience of using A(I,J) instead of A((I*M + J)*E, E) to access a 2-dimensional array, the advantage of the automatic subscript calculation is speed in that, with E and J each restricted to 1-byte, the multiplications are much faster as part of the specific subscript calculation.

(b) The only bounds checking is on the final element specified - subscript J can legally exceed M. This means that a 2-dimensional array may also be accessed as a 1-dimensional array. In particular A(I) and A(0,I) are the same element.

(c) When the multiplier is zero the array may be accessed either as a set of fixed-length items or as a set of variable-length items. In particular if the element size is set to 1 (the default declaration ARRAY A NE1) A(I) accesses a 1-byte element starting at I and A(I,J) accesses a J-byte element starting at I. Note that in the two-subscript accesses with the multiplier zero the first subscript is not multiplied by the element-length. This allows access on arbitrary byte boundaries.

(d) The declaration of a 1-dimensional array of N E-byte elements is:

ARRAY N*E E.

(e) The declaration of a 2-dimensional array of N*M E-byte elements is:

ARRAY N*M*E M E.

(f) The restriction of the multiplier to the range 1 to 255 limits one dimension of a two-dimensional array to this range. However, by using either the matrix or its transpose this will never be a practical limitation.

(g) Setting up reference variables to the elements of arrays is the most rapid way of gaining access to them since the array hookup and subscript calculation need be done only once when the reference is set up.

This is particularly useful in the case of record structures having a number of fields of varying length in the same array.

2.5 Access to System Information

Most system information is accessed in MINSYS through the use of expressions involving the array Q and system variable function QV. Some of the expressions are cumbersome, but it is intended that they be used only once by the user to set up a global reference variable giving fast, simple access to those system variables that are relevant to his application. The initialisation routines for any program suite will set up these references.

The following list gives the main system variables. For a complete guide see the "Multi-User Companion Programmers Guide".

2.5.1 Job Number

1 + QV1/32 is the current job number in the range 1 to 7.

2.5.2 Console Number

Q(QV2) is the current job console number (device on channel 0).

2.5.3 Time of Day

Q(QV0 + 19)	Year -1900
Q(QV0 + 18)	Month -1 (0-11)
Q(QV0 + 17)	Day-of-month -1 (0-30)
Q(QV0 + 16)	Hours since midnight (0-23)
Q(QV0 + 15)	Minutes (0-59)
Q(QV0 + 14)	Seconds (0-59)

Note that this time information is arranged in ascending order in memory so that a multi-byte "time" can be extracted for records and for comparisons if required, e.g.

Q(QV0 + 17,3) is a "date".

2.5.4 Other Time Information

Q(QV0 + 20)	Day of week (0-6, Monday-Sunday)
Q(QV0 + 21,3)	Time since midnight in sec/100
Q(QV0 + 24,2)	Day-within-year -1

2.5.6 System Times

Q(QV0 + 26,2)	System uptime in minutes
Q(QV0 + 28,4)	System nulltime in sec/100 (CPU time not used by users)
Q(QV1 + 28,3)	CPU time for this job in sec/100
Q(QV(256*J + 1) + 28,4)	CPU time for job J in sec/100
Q(QV1 + 26,2)	Login time for this job in minutes
Q(QV(256*J + 1) + 26,2)	Login time for job J in minutes

2.5.7 Channel Status

3. SYNOPSIS OF MINSYS COMMANDS

3.1 Storage (null effect on execution)

$ (characters)

sets up a character string in the program line to be used as a string variable.

[(characters)

sets up a character string for comments (the square bracket replaces 'REM' in BASYS).

3.2 Editing and housekeeping

LIST (ne1) (ne2) (ne3)

gives formatted listing on channel ne3 (default TTY) of program lines ne1 through ne2.

ZERO (ne1) (ne2)

deletes program lines ne1 through ne2.

CODE (se)

reacts to the string se as if it had been typed in during the keyboard edit phase - enables program to compile additional lines at run-time.

GARB

collects garbage to maximize free space - normally done automatically when free space goes below limit specified in QG.

3.3 Assignment

LET (name1) = (ne1) (name2) = (ne2) etc.

assigns the value ne1 to the numerical variable name1, etc.

If the "= (ne1)" is omitted then name1 is given the value zero, etc.

LOCAL (name1) = (ne1) (name2) = (ne2) etc.

assigns the value ne1 to the local variable name1 etc.

If the "= (ne1)" is omitted then name1 is given the value zero, etc.

ARRAY (letter) (ne1) (ne2) (ne3)

sets up the array, letter, of length ne1 bytes, with elements of size ne2, multiplier ne2 (for two-dimensional access), and zeros its elements.

3.3 Assignment

LET (assignment1) (assignment2) etc

where an assignment is either by value:

(name1) = (ne1)

which gives the simple variable, or array element, name1, the value ne1.

Or the assignment of a reference:

(name1) % (name2)

where the simple variable, name1 (not an array element), is set up as a new local variable referring to the simple variable, or array element, name2.

Or a combination of these two:

(name1) % (name2) = (ne1)

which gives the value ne1 to name2 and makes name1 a reference to name2.

Finally, there is the default assignment:

(name1)

which assigns the value zero to name1.

LOCAL (assignment1) (assignment2) etc

is similar in its effect to LET but only searches the current local environment when setting up a simple variable for assignment. Hence any new simple variables created will be ilocal to the current BEGIN block (procedure or iteration block), and become non-existent on exit from it.

ARRAY (letter) (ne1) (ne2) (ne3)

sets up the array, letter, of length ne1 bytes, with elements of size ne2 bytes, and multiplier ne3 (for two-dimensional access); the array elements are zeroed. If ne2 is zero it is taken to be one. If ne3 is zero it signifies a 1-dimensional array.

3.4 Conditionals

IF (ne)

continues execution of the line if the value ne is non-zero.

UNLESS (ne)

continues execution of the line if the value ne is zero.

Many other commands have implied conditionals and continue execution of the line only if they have been performed satisfactorily.

ELSE

continues execution of the line if the previous conditional (except ELSE/AND) did not.

AND

continues execution of the line if the previous conditional (except ELSE/AND) did so.

3.5 Transfer of Control

DO (ne)

executes program line ne - control returns to line following DO commands. However, if line ne commences with a BEGIN command then this acts as procedure call passing control to block commencing with begin (see next section).

GOTO (ne)

transfers control to line ne of program.

LOOP

transfers control back to the beginning of the current line (a terminal colon is taken as :LOOP).

RUN (ne)

deletes simple variables and arrays, gives system variables default values and transfers control to next line with number greater than or equal to (ne).

STOP

stops execution and returns to keyboard edit mode.

EXIT

stops execution and CALLs standard system program.

BYE

stops execution and logs user off system.

3.6 Procedures and Iteration

BEGIN (name1) = (ne1) (name2) = (ne2) etc.

puts the local variable Q, the environment marker, on the stack and sets it equal to QW, the current line number. If then assigns the expression ne1 to the local variable name1, etc., unless parameters have been passed to them by a DO statement. If the "= (ne1)" field is omitted name1 is given the value zero unless a parameter is passed to it, etc.

DO (ne1) (par1) (par2) etc.

unless line ne1 commences with a BEGIN command this executes line ne1. However, if a BEGIN block is entered by a DO the environment marker Q is set to the line number of the calling line and the fields par1, par2, etc. are assigned to name1, name2, etc. in the BEGIN command is preference to any assignment in that command (which therefore act as default values if no parameters are passed - the default default-parameter when there is no assignment is zero).

Par1, par2, etc. are either numeric expressions or are a percent sign, %, followed by a simple variable or array element. The % sign indicates a reference to the element passed.

BACK (ne)

if ne is omitted or ne is non-zero (e.g. conditional expression true) this causes the local environment back to and including the marker Q to be cleared and control to be passed to the line at Q + 1 (or next greater if there is none). Otherwise the command has no effect. BACK is intended as a procedure return.

NEXT (ne)

if (ne) is omitted or ne is non-zero this causes control to be passed to the line at Q + 1 (or next greater if there is none). Otherwise it causes the environment to be cleared back to and including the variable Q. NEXT is intended as an iterative block controller.

END

clears the environment back to and including the variable Q (equivalent to NEXT 0). END is intended as an end-of-block terminator for non-procedure, non-iterative, blocks.

3.7 String Operations

PUT (se)

sets up a source string $QS containing se and sets QP to zero.

AS $(ne) (se)

assigns ne to QD and sets up a destination string $QD containing se.

WITH (se)

appends se to $QD.

FROM (template)

examines $QS from the QP'th character for the template - fails if string does not start with template - otherwise advances QP to point after part of string matching template.

SEEK (template)

runs through $QS from the QP'th character looking for template - fails if not found - otherwise advances QP to point after part of string matching template.

TO (template)

runs through $QS from the QP'th character looking for template - fails if not found - otherwise advances QP to point after string matching template and appends characters skipped over to $QD.

All three commands continue execution of the line if they succeed but transfer to the following line if they fail (AND and ELSE may be used to further test their success or failure). If they fail they have no effect on QP or $QS.

A template consists of a succession of fields indicating matches or changes to the command parameters. A template matches only if all the successive fields match. The possible fields are:

% G (ne)	Get ne characters.
% F (ne)	Set numeric input format - if ne is expressed as 100*D + R, input radix is R and number of decimal places is D. Default initial setting is 10.
(variable)	Any variable to which assignment is possible - matches a numeric field which is: [spaces] [+, -, null] [spaces] [digits] [decimal point] [digits] - assigns any number found to the specified variable using the current format.
$ (ne)	Match the string in $ne.
"(string)" or '(string)'	Match the literal string.

3.8 Peripheral Transfers

All MINSYS peripheral transfers are mediated by the MINIC operating system, Multi-User Companion (MUC - see MUC Guide). Input-output takes place through numbered channels to each of which may be allocated a physical device or a disc file. The channels are numbered from 0 to N (normally 7). Channel 0 is reserved for the program control and editing device, or job console. Channel 1 is used by default for various purposes and is intended as a temporary device channel. The other channels are freely available. MUC is designed such that an input-output console need be allocated to only one channel.

3.8.1 Device Allocation

ALLOCATE # (nel) (ne2) (ne3)

assigns to channel nel (default is channel 1) the device whose number is ne2. The final field ne3 is an optional code which may be used to change line characteristics, flush buffers, or release devices.

Note that a keyboard device allocated to a channel also automatically makes its printer available as an echo/output device on the same channel. Hence the keyboard only of a teleprinter or typewriter need be allocated to a channel.

3.8.2 Device Numbers

0 - No device - IO causes error return

1 - No device - IO causes monitor return

2 - No device - IO causes normal return (this is a throw-away channel)

64 - Paper tape reader

80 - Paper tape punch

96 - Keyboard

112 - Caroussel projector and lamps

32 - Console teleprinter keyboard

48 - Console teleprinter printer

33 - IBM typewriter keyboard

49 - IBM typewriter printer

34 - Modem keyboard

50 - Modem printer

3.8.3 Allocate codes

The interpretation of the final field, ne3, in the allocate instruction is:

Value	Meaning
0 (or missing)	Just perform allocation
16	Release device from this job
16 + J	Release device from job J
256	Flush buffer of device
512 + L	Set line characteristics of device to L
768 + L	Set line characteristics of device to L and flush its buffer
1024*J + 768 + L	as above but for job J

The line characteristics are a number in the range 0-255 made up of sub-fields:

Sub-field	Meaning
128	ECHO characters input on associated printer - otherwise no echo.
64	ASCII - skip nulls, line-feeds and 200 octal, recognize rubout and flush characters - otherwise image mode in which all characters are input
32	CR-ECHO: Input devices - echo CR as CRLF - otherwise do not echo CR at all; Output devices - translate 200 octal into CRLF

Sub-field	Meaning
16	RESTART - recognize restart characters from keyboard.
8	CONTROL-ECHO - echo characters below 40 octal as ↑ the character + 100
4	AVAILABLE to other jobs
2	7-BIT - set top bit of characters zero in block transfers
1	PARITY - generate even parity on output - check parity on input

3.8.4 Transfers

PRINT #(nel) (sel)

causes the string expression sel to be output to channel nel - default is channel 0.

INPUT #(nel) ? (ne2) $ (ne3)

causes the string in $ ne2 to be output on channel nel (default string is a colour, default channel is 0) and then a string to be input up to a delimiter (character between 1 and 40 octal except TAB, and special system characters). The value of the delimiter, or break character, is placed in the system variable QB. The string input is put into $ ne3 or, if this field is absent, then it is put into $ QS and QP is set to zero (i.e. either input to any $-line or input sets up string as if a PUT statement had been used).

WRITE #(nel) (letter)

block transfers the bytes in the array named letter out on channel nel (default is channel 2).

READ #(nel) (letter)

inputs a block of bytes from channel nel (default is channel 3) into the array named letter.

For both READ and WRITE commands the size of the block transferred is that of the array.

The channel specification may be omitted normally when the standard default channels are used. Additional input/output and supervisory commands will exist for most systems since it has been the policy to interface most operating system facilities directly to BASYS (with suitable protection where necessary).

```
105 PRINT 'FILE ERROR' : STOP
110
```

may be replaced by

```
100 INPUT $
105 ELSE : PRINT 'FILE ERROR' : STOP
110
```

where the flow of control is easier to see and the program less liable to errors in editing.

```
50  PUT $6
52  FROM 'AB'
54  AS $7
56  TO 'KJ'
```

or, avoiding copying $6:

```
50  LET QS = 6  QP = 0 : AS $7
52  FROM 'AB' : TO 'KJ'
```

or as several other variants - the new command structure is more flexible and allows both simple and complex decoding to be written simply and naturally.

(d) System functions having arguments are treated as unary operators not requiring parentheses, e.g. QT is the tally function, QT(X) is the number of bits which are 1 in X - it may be written QTX.

(e) The format and radix specifications in string expressions apply to the remaining expression not to the previous item, e.g. @6X not X@6.

(f) The comment "COMMAND" is left square bracket, [, not REM.

(g) The CLEAR command is called ZERO (this is only to maintain rule that any command may be uniquely specified by first two letters).

(h) IF and UNLESS set a flag if they succeed (and continue execution) and unset it if they fail (and go on to next line). This flag may be tested by ELSE which succeeds if, and only if, the last IF/UNLESS failed, and AND which succeeds if, and only if, the last IF/UNLESS also succeeded. All other conditionals, such as the macro tests, TO, FROM, SEEK, and input/output commands, INPUT/PRINT, READ/WRITE, set the flag if they succeed and unset it if they fail. ELSE and AND may be freely used to avoid trailing GOTO's

e.g.

```
100 INPUT $ : GOTO 110
```

MINSYS USER BULLETIN 1

4'TH JULY 1974

(1) DISTRIBUTION LIST

PETER FACEY
JOHN GEDYE
TIM KENNEDY
LADISLAV KOHOUT
MALCOLM LEE
PETER WILLIAMS
KEITH WILLIAMSON

(2) STATUS OF MINSYS

THE FIRST ISSUE OF MINSYS IS VERSION 1C WHICH RUNS UNDER MULTI-USER COMPANION (MUC) USING THE FULLY BUFFERED IO FACILITIES. THIS IS A SINGLE-USER VERSION OF MINSYS AND IS NOW SUBSTANTIALLY COMPLETE BUT BY NO MEANS FULLY TESTED.

INITIAL USERS WILL BE GUINEA PIGS - PLEASE KEEP TTY OUTPUT WHEN BUGS OCCUR, MAKE NOTE IN LOG-BOOK AND COMPLAIN TO ME.

THIS BULLETIN WILL BE USED TO DOCUMENT UPDATES, KNOWN BUGS, AND BUGS REMOVED.

MUC ITSELF IS ALSO AT PRELIMINARY ISSUE STAGE BUT BOTH PACKAGES ARE NOW WORKING TO A LEVEL WHERE SERIOUS USE IS POSSIBLE.

(3) LOADING MINSYS

(A) FIRST LOAD THE APPROPRIATE MUC TAPE:-

UCH-MUC.BOT	V1C	FOR THE UCH SYSTEM
BP-MUC.BOT	V1C	FOR THE BP SYSTEM

(THE TAPES ARE IN TOP DRAW OF CABINET BY UCH SYSTEM)

THESE HAVE TO BE BOOTSTRAPPED IN - PRESS 'IDLE' - KEY-IN BOOTSTRAP - PRESS 'CLEAR-ALL' - PRESS 'RUN' WITH MUC TAPE IN FAST READER.

TAPE WILL LOAD AND PUT OUT MONITOR DOT ON TTY.

(B) NOW LOAD THE CURRENT MINSYS TAPE:-

MINSYS V1C.SBN

BY PLACING IN FAST READER AND TYPING 'L' TO MUC.

TAPE WILL LOAD AND PUT OUT MONITOR DOT ON TTY.

(C) NOW TYPE 'R' TO MUC TO START UP MINSYS.

MINSYS WILL RESPOND WITH GO-AHEAD (>) ON TTY.

YOU ARE NOW IN MINSYS EDIT MODE AND CAN PROCEED WITH MINSYS PROGRAMS.

(4) NORMAL STATE OF UCH SYSTEM

FOR CONVENIENCE, WE SHOULD TRY TO MAKE THE NORMAL STATE OF THE UCH SYSTEM THAT THE LATEST VERSION OF MINSYS IS LOADED AND RUNNING. THIS WILL BE INDICATED IN LOGBOOK AS 'MINSYS V1C UP'.

PLEASE ALWAYS FILL IN LOGBOOK AND NOTE ANY CRASHES.

(6) SPECIAL CHARACTERS

THE TTY IS CURRENTLY THE ONLY PROGRAMMING TERMINAL. IT IS OPEN ON CHANNEL ZERO AND ITS CHARACTERISTICS ARE SET UP SO THAT:-

CONTROL-C CAUSES RETURN TO MONITOR (FROM WHICH 'R' RUNS MINSYS OR 'C' CONTINUES MINSYS PROGRAM INTERUPTED).

CONTROL-R RESTARTS MINSYS (NORMAL WAY OF GETTING OUT OF MINSYS PROGRAM LOOPS).

CONTROL-F TURNS OFF OUTPUT WHEN ON, OR ON WHEN OFF.

RUBOUT ERAZES LAST CHARACTER INPUT.

THERE IS NO ERAZE LINE FACILITY YET.

(6) SAVING AND LOADING MINSYS PROGRAMS

THE INTERIM FACILITIES FOR SAVING AND LOADING MINSYS PROGRAMS ARE AS FOLLOWS:

THE COMMAND 'TAPE' ALLOCATES THE FAST READER TO CHANNEL 4 AND THE FAST PUNCH TO CHANNEL 5, INITIALIZING BOTH DEVICES FOR ASCII TRANSFERS & CLEARING THEIR BUFFERS. IT MUST ALWAYS BE GIVEN BEFORE ANY USE OF PUNCH OR READER IS ATTEMPTED. MUC INITIALIZES THESE DEVICES IN IMAGE MODE AND STRANGE RESULTS WILL ENSUE CURRENTLY IF THE COMMAND 'TAPE' IS NOT USED !

THE COMMAND 'LIST #5' ('LI#5') WILL PUNCH OUT THE MINSYS PROGRAM CURRENTLY IN CORE WITH APPROPRIATE LEADER AND TRAILER.
(LIST #5 2C 725 WILL PUNCH OUT LINES 2C THROUGH 725).

THE COMMAND 'CALL #4' ('CA#4') WILL LOAD A MINSYS PROGRAM FROM A TAPE CREATED BY A LIST #5 COMMAND.

(7) FACILITIES AVAILABLE

THE MINSYS 'PRELIMARY REFERENCE MANUAL' DATED 6TH MAY 1974 IS SUBSTANTIALLY CORRECT AND COMPLETE EXCEPT FOR A DESCRIPTION OF IO FACILITIES WHICH WILL BE ISSUED SHORTLY. THE FOLLOWING NOTES LIST UNIMPLEMENTED FACILITIES AND KNOWN PROBLEMS:

(7.1) ARITHMETIC EXPRESSIONS

THREE OPERATORS ARE NOT IMPLEMENTED IN V1C AND ACT AS NULL OPERATIONS:

- _ THE SHIFT OPERATOR
- 0 THE POWER OF 10 OPERATOR

^ THE EXPONENIATION OPERATOR

IT IS NOT INTENDED TO IMPLEMENT THIS LAST OPERATION SINCE IT IS OF LIMITED UTILITY IN FIXED-POINT ARITHMETIC. 0 GIVES POWERS OF 10 AND _ GIVES POWERS OF 2.

ALL OTHER ASPECTS OF ARITHMETIC EXPRESSIONS ARE IMPLEMENTED. INCLUDING 2-D ARRAYS, REFERENCE VARIABLES, PARAMETER-PASSING, ETC.

(7.2) STRING EXPRESSIONS

%S, THE 'SYMBOL TABLE' LOOKUP, IS NOT IMPLEMENTED IN V1C, OTHERWISE AS IN MANUAL.

(7.3) TAB COMMAND, QX & QY

THE TAB COMMAND & CARRIAGE POSITION POINTERS, QX & QY, ARE NOT IMPLEMENTED IN V1C.

(7.4) OPERAND LENGTH

THE ARITHMETIC OPERAND LENGTH IS SET AT 5 BYTES, 40 BITS. IN V1C. V1D WILL HAVE FACILITIES TO VARY THIS.

(7.4) FILE COMMANDS

THE COMMAND OPEN IS NOT YET FULLY IMPLEMENTED - IT OPERATES ONLY WITH NON-FILE DEVICES (GIVING LEADER ON PUNCH).
FACILITIES TO DELETE AND RENAME FILES ARE ALSO NOT YET IMPLEMENTED.

(8) KNOWN BUGS

(8.1) THE PROGRAM CODING ROUTINE DOES NOT REMOVE SPACES BEFORE THE COLON PRECEEDING A COMMAND. WHEN PROGRAMS ARE DUMPED AND RESTORED FROM PAPER TAPE AN EXTRA SPACE GETS INSERTED, CAUSES NO ERRORS BUT UGLY LISTINGS.

(8.2) %G IN 'TO' COMMAND NOT WORKING.

(8.3) UNDER SOME CONDITIONS ILLEGAL DATA STRUCTURES CAN CAUSE GARBAGE COLLECTOR TO HANG UP. RELOAD MINSYS & RESTART. THIS IS ONLY MAJOR PROBLEM & I HOPE TO PIN IT DOWN SHORTLY.

(8.4) 'BEGIN' WITH NO PARAMETERS CAUSES AN ERROR. GIVE IT A DUMMY PARAMETER IF YOU ARE PASSING NONE (EG BEGIN X).