Burroughs Corporation
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

2222 3044

INTERACTIVE BASIC COMPILER

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
C	2/3/82		Changes for MARK 11.0 Release 1-1 Removed "Number 4" of the components of the system. Removed "IBASIC/RUNNER" file name. 1-2 Changed "10.0 release" to "11.0 release". Deleted "sec. 4.1.2", "sec. 6.3", and "sec. 9.3.2". 3-2 Added "INITIAL PROGRAM PARAMETERS". 3-5 Updated "USERCODE CONSIDERATIONS". Updated "WORKFILE CONSIDERATIONS". 3-6 Changed "SESSION TIME LIMIT" to "RUN TIME LIMIT" and updated definition. Updated "SWITCH VALUES" by deleting all except SW=W. Updated description of "ODT OPERATION". Updated description of "EXECUTION UNDER SMCS". 3-7 Added "However, if the sign OFF ..." paragraph. Updated "UNDER SMCS USING SIGN ON SYNTAX" description. 3-9 Updated "EXECUTION WITH NO MCS" description. 4-4 Updated definition of "BYE" command. 4-15 Replaced "SW=WW option" with "AX DEBUG WW or SW=W option". 4-22 Updated "The IBASIC system ..." paragraph under Recovery. 5-1 Updated "RESERVED EXTENSIONS TO THE ANSI LANGUAGE" description paragraph.

TABLE OF CONTENTS

SCOPE OF DOCUMENT

This document describes the B1000 Interactive BASIC system. The components of the system are:

1. A compiler/interpreter/mcs/editor program written in SDL.
2. A set of microinstructions written in MIL which constitutes the Interactive BASIC interpreter.
3. A file containing code for the external intrinsic functions and subprograms.

The names of the files are:

IBASIC	(compiler)
IBASIC/INTERP	(interpreter)
IBASIC/INTRINSICS	(intrinsics)

The Interactive BASIC system requires the following minimum hardware and software:

B1700, B1800 or B1900 processor (excluding B1710 and B1830).
128K bytes. (dependent on number of users).
1MB disk per user.
TD820, TD830, TC4000, TTY type terminals.
MCPII (9.0 or later)
NDL (9.0 or later)

The syntax of the BASIC language implemented is not documented here. The proper source of this information is the Corporate Language Standard. However, since that document is currently very much neglected, the more accurate description of the language implemented is to be found in the following working document of the ANSI X3J2 BASIC standards committee:

X3J2/79-51 Preliminary draft ANS BASIC.

The following list of language features are not implemented in the 11.0 release of the Interactive BASIC system:

sec. 4.2.2.1	subprogram-part, loop.
sec. 4.2.2.8	array-linput-statement, call-statement, ginput-statement, gprint-statement, plot-statement, real-time-statement, set-statement, subexit-statement, transform-statement.
sec. 4.2.2.15	subprogram-part.
sec. 4.2.2.17	end-picture-line, paract-line, parend-line, picture-line, process-line, sub-line,subend-line.
sec. 4.2.2.18	subprogram, picture-definition.
sec. 5.6	all.
sec. 7.3	string-array substring and concatenation
sec.10.2	all.
sec.11.5	array-linput-statement.
sec.11.6.2	array-linput-statement,
sec.12	all.
sec.13	all.
sec.14	all.
sec.16	the value of variables assigned in the statement traced is not reported.

This document is organized in 5 sections:

1. Fundamental concepts
2. Operational considerations
3. Interaction syntax
4. Extensions to the ANSI language
5. Implementation defined values.

FUNDAMENTAL CONCEPTS

BASIC ENVIRONMENT

A BASIC environment consists of two parts, a BASIC code environment and a BASIC data environment.

A BASIC code environment is a set of BASIC statements each preceded by a line number.

A BASIC data environment is the set of data scalars and arrays which are accessed by the simple pre-defined names A, A0....A9,....Z, Z0....Z9 (for real numeric type values) and A$, A0$....A9$,....Z$, Z0$....Z9$ (for string type values).

INTERACTION MODES

Interaction with a BASIC environment occurs in one of two modes, command or entry.

The distinction between the two is dependent solely on the presence of a line number preceding the string of characters presented to the BASIC environment. If a line number exists then the interaction is termed an entry and implies the entry of a BASIC language statement. If a line number does not exist then the interaction is termed a command and may be either a BASIC language statement or a system command which is to be compiled and executed immediately.

ENVIRONMENT STATES

The BASIC code environment may be either in a running or a stopped state.

In a running state BASIC statements are executed as the program logic dictates until either a fatal run time condition is met (e.g. an invalid subscript) or until a BREAK, STOP or END statement is executed. The transition from a stopped state to a running state is made if a RUN, WALK, CONTINUE or STEP command is executed successfully.

SCOPE OF ACCESS

A particular BASIC environment may access, for input only, any disk file maintained by MCPII which may be accessed under the currently logged on usercode. The BASIC environment may only create or change a file which has a family name of the currently logged on usercode.

Thus the rules for forming a file name which is for input only, are more flexible than those for files which are to be created.

IBASIC SYSTEM

The IBASIC system is the running copy of the SDL2 coded IBASIC program and its associated micro interpreter.

PRIVILEGED STATUS

The IBASIC system has privileged status if it is executed under a privileged MCP type of usercode or a non MCP usercode. Non-privileged status can only happen if it is executed under a non-privileged MCP type of usercode.

OPERATIONAL CONSIDERATIONS

NDL CONSIDERATIONS

There are some constraints on the set up of the network controller.

The interactive BASIC system assumes the validity of the TYPE in the TERMINAL section. Currently only the following types are handled:

0	- B9350 (TTY)	(tested using a TI 700)
26	- TC4000	(untested)
41	- TD801	(untested)
42	- TD802	(untested)
43	- TD821	
44	- TD822	
45	- TD831	
46	- TD832	
54	- MT686	
61	- VIRTUAL	

Scrolling of input and output lines is supported only for TD820 and TD830 type terminals and, to enable scrolling, the standard CANDE request and control sets must be used.

For correct operation of TTY type terminals, the CANDE IOTTY request set should be used.

The following considerations are only relevant if non standard request and control sets are used. A knowledge of NDL coding and the use of station TOG and TALLY values is assumed. The special meanings for the following values are assumed.

TOG[1] set on an output message means do not scroll this message.

A true value in station TOG[2] in an input message means that the input message was scrolled or that the output message is to be scrolled.

Station TOG[3] will be set for all output messages to a teletype class of terminal and associated TOG[7] may be set to indicate inhibition of transmission of trailing CR and LF characters for this message.

Station TOG[3] will also be set for all messages that are to be scrolled. This method is used to enable multiple line output scrolling which is in the CANDE request sets.

A true value in station TOG[5] means that BREAK was detected in the last output attempt. This must only be true in a 'GOOD RESULTS REPLY' type message.

Station TALLY[0], with a value of 2, requests that a screen type terminal be forced to local (even in scroll mode) after this output message.

INTERACTIVE BASIC SYSTEM CONSIDERATIONS

The SDL program component (hereafter called IBASIC) may or may not require modifications to suit the particular user. These modifications may be applied through the MOdify MCP syntax or as parameters with the EXecute control syntax.

There will be one copy of the IBASIC program per user of the system. Thus the following considerations may be applied differently to each user or globally to all users (as required).

DYNAMIC MEMORY

IBASIC relies heavily on SDL paged array structures. Both s-code and data are maintained in paged arrays, hence if the average BASIC program has many statements or uses large amounts of data then IBASIC may require a larger dynamic memory size. Dynamic memory may only be changed at BOJ time of IBASIC so either an ME control parameter must be included in the EXecute command or a particular installation must MOdify the default value to suit its particular median requirements.

PRIORITY

For quick servicing of a request from a user station, IBASIC should be run at a PRiority (both memory and processor) higher than any batch jobs (as for most remote applications).

INITIAL PROGRAM PARAMETERS

The AX mechanism is used to suply values to a set of parameters which help determine the IBASIC environment. An AX message may be supplied with the EXecute statement. All of the parameter values desired must be supplied in one AX message. Following are the program parameters along with the allowable values and default setting for each.

PARAMETER	VALID ARGUMENTS	DEFAULT SETTING	EXPLANATION
[NO] AUTO_LOG		false	see UNDER SMCS using sign on syntax
BACKSPACE	<character>	"\"	
[NO] CONTINUOUS		false	causes page pauses on terminal ouput.
DEBUG	"DEBUG"	none	sets a debug option within the compilter.
	"LOG"	none	initiates a printer log of input and output operations with the remote user.
	"WW"	none	same effect as SW=W.
	"QC"	none	causes all output messages to the terminal to be routed to the remote file which approved the open for IBASIC. The message type will be set to "50" to indicate that the message is from IBASIC not the remote station.
[NO] EOJ_ON_LOGOUT		true	see EXECUTION UNDER SMCS
[NO] LOWER_CASE		true	specifies alphabetic case for output.
[NO] ODT		false	see 'ODT OPERATION'
RUN_LIMIT	<integer>	0	CPU runtime limit; 0 means no limit.
[NO] RUNNER		true	enables calls to the microinterpreter
[NO] SMCS_AUTO_LOG		false	see 'EXECUTION UNDER SMCS'
[NO] USERCOES		true	indicate whether operating under MCP usercode/password security.

WORKFILE_AREAS	<integer>	5	size of source workfile

The settings of SMCS_AUTO_LOG and AUTO_LOG are mutually exclusive, as are SMCS_AUTO_LOG and ODT_MODE. NO EOJ_ON_LOGOUT set requires SMCS_AUTO_LOG to be set.

USERCODE CONSIDERATIONS

IBASIC may or may not be executed under the MCP usercode system.

If IBASIC is not executed under the MCP usercode system then the scope of access of the BASIC environment is limited to public files. Note that for this case, the AX NO USERCODES option must be set (see INITIAL PROGRAM PARAMETERS). In this mode the concept of a password does not exist and a logged on usercode is simply a default family name of a file accessed for input only and a mandatory family name for a file to be created or changed.

If IBASIC is executed under a non-privileged MCP usercode then IBASIC automatically 'logs on' that usercode. Log on of another usercode is disallowed. IBASIC must be 'logged off' and re-executed under a different usercode to effect a change of user.

If IBASIC is executed under a privileged MCP usercode then, dependent on the value of the AX AUTO_LOG parameter, automatic log on to the execution usercode may or may not occur. If automatic log on is not requested then IBASIC will request an explicit log on from the user before continuing. The usercode and password used must, of course, be defined in the MCP's usercode file. Regardless of the AX AUTO_LOG setting another user may log on after the current one has logged off without IBASIC going to EOJ.

WORKFILE CONSIDERATIONS

By default, the workfile will allow up to 1979 BASIC statements in a program. If it is required to increase (or decrease) this limit (and/or change the disk space required per user), the AX WORKFILE_AREAS option must be used (see INITAL PROGRAM PARAMETERS). By default the value of this parameter is 5. Each area can hold 400 statements and thus, to increase (or decrease) the maximum program size handled, this WORKFILE_AREAS parameter must be incremented (or decremented). Note that this parameter only has effect on creation of a new workfile. If a recovery workfile is found, the maximum size is that which the existing workfile can contain. e.g. EX BASIC FI WORKFILE; AX WORKFILE_AREAS 10;

RUN TIME LIMIT

There is a means to specify a limit of CPU time that a user program may consume without interruption. If no limit is specified (the default situation) then a user program may continue indefinitely. If a limit is specified, a program will be temporarily suspended if that limit is exceeded during the session. Program execution may, however, be continued in the normal fashion after this interruption until the limit is again consumed. The intent of this feature is to stop a program from consuming vast amounts of CPU if it is in a loop and is running unattended.

The AX RUN_LIMIT parameter of the WORKFILE is used to specify this limit in units of a minute (see INITIAL PROGRAM PARAMETERS). Hence, if IBASIC is executed with AX RUN_LIMIT 2, then every time 2 minutes of CPU time is used up in program execution, the user session is interrupted and the user must do something positive (e.g. enter a CONTINUE command) to continue the session. If the RUN_LIMIT parameter has the value zero then the user session is unlimited.

SWITCH VALUES

For debugging purposes, the following value is defined:

SW = W - enables certain extensions to the ANSI language and forces auto log on.

ODT OPERATION

If no remote operation is required, i.e. no data communication system exists or is needed, then IBASIC may be operated from the system console. To achieve this end, the AX message (see INITIAL PROGRAM PARAMETERS) must include 'ODT'. Further communication with IBASIC is achieved through the normal MCP accept (unsolicited) and display interface. The ensuing log on process is identical to the normal remote log on process according to the usercode (if any) under which the IBASIC program is logged on. There is no implied limit to the number of users who may use the ODT.

EXECUTION UNDER SMCS

IBASIC may be initiated using either the EXecute or the sign ON syntax with one of two possible entries in the SMCS/JOBS file (one which causes a BOJ each time, one which causes a BOJ only the first time). If the EXecute syntax is chosen then no entry in the SMCS jobs file is required. If the sign ON syntax is used then an entry in the SMCS jobs file is required.

In either case, the signal character for IBASIC must not be either '#' or '.' or SUB (USASCII67 1/10).

Termination of the current IBASIC session may be caused in either case by using the sign OFF syntax (i.e. enter '<signal> OFF'). However, if the sign OFF syntax is used during an IBASIC session initiated with the AX parameter setting of 'NO EOJ_ON_LOGOUT', then IBASIC will not go to EOJ. If the '<signal> OFF CLEAR' syntax is used, the IBASIC job will be available for any station to sign ON to without causing BOJ. Otherwise, if the '<signal> OFF' syntax is used, SMCS will only allow the same station that signed OFF to sign ON again, firing up a new copy of IBASIC for any other station's sign ON.

Termination of an IBASIC job will occur if a message is sent to IBASIC via the SMCS PASS mechanism.

UNDER SMCS USING EXECUTE SYNTAX

IBASIC may be initiated by entering 'EX IBASIC'. (An AX message may be supplied, see INITIAL PROGRAM PARAMETERS.) If the terminal has been logged on under an MCP usercode (using the USER command) then IBASIC will be executed under that usercode. The normal usercode considerations apply. If automatic log on is required and the logged on usercode is privileged then supply in the initial AX message the setting 'AUTO_LOG'.

UNDER SMCS USING SIGN ON SYNTAX

IBASIC may be initiated by entering 'ON IBASIC'. The SMCS jobs file must have an entry of the following format:

```
     col 1        $
     col 2        IBASIC LOG-ON NO-SESSION ;
                  EX IBASIC
                  ME <user site default, see DYnamic MEmory)
                  PR <user site default, see PRiority)
                  AX SMCS_AUTO_LOG <plus other settings as desired,
                     see INITIAL PROGRAM PARAMETERS>
```

The above entry will cause an auto log-on, and will go to EOJ when the user signs off.

The following entry will cause an auto log-on, and will not go to EOJ when the user signs off. Instead, the user is logged off and his station is detached from IBASIC. IBASIC will wait in the job mix for another user to sign ON. (If a user is not logged on, IBASIC will go to EOJ upon receiving a BYE or sign OFF command.)

```
col 1        $
col 2        IBASIC NO-EOF <wait limit-see note below>
                 LOG-ON EXCEPT NO-SESSION
             EX IBASIC
             ME <user site default>
             PR <user site default>
             AX SMCS_AUTO_LOG NO EOJ_ON_LOGOUT <plus
                other AX settings>
```

Note: if a limit is desired on the time a job with NO EOJ_ON_LOGOUT set sits in the mix with no user, then place in the <wait limit> spot above:

```
"(" [[<hours> ":"] <minutes> ":"] <seconds> ")"
```

where <hours>, <minutes> and <seconds> are integers. e.g. (2:30) would be a limit of 150 seconds.

EXECUTION UNDER CANDE

Use the execute command to cause execution under the usercode which is logged on to CANDE. If automatic log on to this usercode is required then the AX message should include AUTO_LOG. e.g. EX *IBASIC <optional ME parameter>; AX AUTO-LOG.

N.B. If IBASIC does a display to the ODT (for example as a result of a '.SS' command) then the display messages will be repeated at the remote terminal and will only be displayed at the ODT if the RMSG system option is set.

EXECUTION WITH NO MCS

The IBASIC program must be executed from the ODT with ODT_MODE set in the initial AX message (see INITIAL PROGRAM PARAMETERS). If automatic log on to the usercode under which IBASIC is executed is desired, then the AX message should also include AUTO_LOG. If other than the default MEmory is required, then a MEmory clause should be added to the control string. e.g. US ME/MINE EX IBASIC; ME 400000; AX ODT_MODE AUTO_LOG;

INTERACTION SYNTAX

Interaction with the invoked BASIC environment may begin any time after the initial identifying message. Interaction is either in command or entry mode. Commands may be classified into two types, System commands and BASIC commands. System commands are only valid in command mode. Interaction mode and the state of the BASIC environment are not necessarily dependent, excepting that entry mode is disallowed while the BASIC environment is in a running state.

Spaces may not appear within any 'keyword'. For example, 'S A V E' will not be accepted for 'SAVE'. (Currently spaces around keywords are not required, but may be in the future.)

Lower case characters are mapped directly on their upper-case equivalent except within quoted strings. Thus SAVE, Save, sAve and saVE are equivalent and "SAVE", "Save", "sAve" and "saVE" are not.

GENERAL SYNTAX DEFINITIONS

LINE NUMBER

```
>-------------------- <digit> ----------------------------------#
                     |         |
                     |<- / 4 \ --|
```

At least one digit must be non zero.

valid line numbers	invalid line numbers
1	0000
0001	1 2 3
02340	123456
100	000100

LINE NUMBER RANGE

```
>----------------------------------<line number> --------------#
             |             | |                            |
             |-- FIRST ----| |-- "-" ----- <line number> -|
             |               |        | |                 |
             |               |-- TO --| |---- END --------|
             |                          |                 |
             |                          |---- LAST -------|
             |                                            |
             |--------------- LAST -----------------------|
             |                                            |
             |--------------- ALL ------------------------|
```

The second line number specified must be larger than the first specified. Also the pseudo line number FIRST (which refers to the first line in the program) must refer to a line number smaller than or equal to the second specified line number. Likewise the pseudo line number LAST (or END) must refer to a line number greater than or equal to the first specified line number. The special cases of FIRST TO LAST or ALL refer to the whole program.

valid line number ranges	invalid line number ranges
100-200	200-100
FIRST TO 3000	3000 TO FIRST
2000 - LAST	LAST TO 2000
FIRST	END - 100
1-END	FIRST TO 200 (where the first line > 200)
ALL	
FIRST TO 200 (first line <= 200)	

BASIC FILE NAME

```
>---------------------- <alphanumeric and "." and "_"> --------#
                    | |                                 | |
                    | |<-------------/ 9 \ -------------| |
                    |                                     |
                    |--- " --- <any character> ---- " -----|
                            |                 |
                            |<---- / 9 \ -----|
```

A BASIC file name inside quotes may contain a quote by using a double quote. A BASIC file name not using quotes will be translated to upper case and must begin with an alphanumeric character.

A BASIC file name may not begin with a "*", " ", "=" but it may begin with a "#". The first and last characters of a BASIC file name may not be "(" and ")" respectively.

valid BASIC file names	invalid BASIC file names
ABCD	ABCDEFGHIJK
1234	|"#$
P.Q	.ABC
"|""#$"	"qwe
Q123ABC	*ABC
abcd (equivalent ABCD)	(ME)
"abcd"	"(ME)"
c___d	----

PACK NAME

```
>-------------------------- <BASIC file name> ------------------#
                       |                                      |
                       |-- " -- <fewer than 11 blanks> -- " --|
```

A pack name may contain up to 10 characters according to the same formation rules as a BASIC file name. A special value of spaces indicates system pack.

valid pack names	invalid pack names
USER.243	(ME)
"1&2"	= or "="
""	`"           "`

MCP FILE NAME

```
>------------------------------- <BASIC file name> ----------->>
                        |                                  |
                        |-- * -- <BASIC file name> --------|
                        |                                  |
                        |-- ( -- <BASIC file name> -- ) ---|
                        |                                  |
                        |-*( -- <BASIC file name> -- ) ----|

    >------------------------------------------------------------#
                          |                            |
                          |-- / -- <BASIC file name> --|
```

The use of quotes needs a little explanation. If the first BASIC file name uses quotes then the quotes must be around any "*" and/or "(" and ")" used.

valid MCP file names	invalid MCP names
(IBASIC)/PQR	"(IBASIC)/PQR"
XYZ	*"QWER"
XYZ/P.Q.R	X$/Y$
*QWER	X/=
*(QWER)/LKJ	" ASD"
"*AS$F"	"(qwe"
"*(qwer)"/"zxcv"	=
X_Y/L.K	=/=

SYSTEM COMMANDS

These commands may be entered at any time, although their effect may be dependent upon the state of the BASIC environment. Unless stated to the contrary, the effect of the command will be immediate. These commands may not be preceded by a line number nor may they be imbedded within a BASIC statement entered without a line number.

BYE

```
>-- BYE -------------------------------------------------------#
```

The BYE command causes the BASIC environment to be cleared and the IBASIC program to go to EOJ, unless the IBASIC program was initiated under SMCS with an AX setting of NO EOJ_ON_LOGOUT, in which case the program will wait for another user to sign ON (see INITIAL PROGRAM PARAMETERS, EXECUTION UNDER SMCS). Checks on the state of the BASIC environment are made and the BYE function will not happen if

1. The source file is not saved.
2. The BASIC environment is running.

CONTINUE

```
>-- CON -------------------------------------------------------#
          |         |   |                   |
          |- TINUE -|   |-/ 1 \- TRACE -----|
                        |                   |
                        |-/ 1 \- PRINT -----|
                        |                   |
                        |<------------------|
```

The CONTINUE command causes the BASIC environment to continue execution from wherever it last stopped. The BASIC data environment is not cleared, i.e. data variables retain the values they had at the time the BASIC environment last stopped. A RUN or WALK command must have preceded the use of this command. Certain editing functions (e.g. deletion of the next statement to be executed, changing the dimensions of an array, changing some FOR-NEXT statements) will cause CONTINUE to be disallowed until after a RUN or WALK command. A suitable message will be emitted in these cases.

The TRACE option causes the statements to be displayed on the remote device as they are executed.

The PRINT option causes the output from BASIC PRINT statements and the input from BASIC INPUT and LINPUT statements to be written to the file LINE as well as the remote device. The file LINE is closed on execution of an END statement in the BASIC program.

If both the PRINT and TRACE options are specified then the trace of statements executed is directed to the file LINE interspersed with the BASIC PRINT, INPUT and LINPUT output. In this instance the traced statements are not displayed on the remote device.

The PRINT and TRACE options have effect until the program is stopped (either by a STOP or END statement or a fatal error).

This command will only have effect when the BASIC environment is in a stopped state.

Examples:

```
     CON
     CON TRACE          (continue and commence tracing)
     CONTINUE
     CONTINUE TRACE     (ditto)
     CON PRINT          (continue, and commence or continue
                         writing to file LINE)
```

DELETE

```
>-- DEL ----------- <line number range> -------------------------#
  |      |       |                          |
  |      |- ETE -|                          |
  |                                         |
  |------------------ <line number> --------|
```

The DELETE command deletes the specified line numbers from the current source file. If the line which would be the next one to be executed is deleted then program execution may only be initiated using the RUN and WALK commands. The line numbers requested must be present in the workfile. No deletion will occur if any requested line number is not present in the workfile.

If the BASIC environment is running then the command is ignored.

Examples:

```
     DELETE 123              (delete line 123 only)
     DELETE 2000-3000
     123                     (delete line 123 only)
     DELETE 4000 TO LAST
     DEL FIRST
     DEL FIRST TO 100
     DEL FIRST TO LAST       (delete all program statements)
```

EDIT

```
>-- EDIT ---------------------------------------------------------#
           |             |  |                                 |
           |--- ERRORS ---|  |------------ <line number> ---
                             |        |
                             |- FROM -|
```

The EDIT command is used to initiate edit mode. In edit mode, each line of the program is presented (at the top of the screen) for review and possible change. Editing of a program may begin either at the beginning of the program or at a particular point (line number) within the program. Edit mode is terminated when either the end of the program is reached, by explicitly using the pseudo break feature, or by entering any command (instead of entering a statement). If the line number is changed in the process of editing a statement, the next line after the new line will be the next one presented for editing.

A variant of edit mode is the edit-errors mode in which the next line which contains a syntax error is presented for change.

The EDIT command is only valid from a screen type terminal.

Examples:

```
     EDIT
     EDIT ERRORS
     EDIT FROM 23450
```

FILE

```
>-- FILE ------- <MCP file name> ---------------------------------#
           |                    |    |                     |
           |--- <file name>/= -----|    |- ON <pack name> -|
           |                    |
           |-------- = ------------|
```

The FILE command inquires of the disk directories available to the current usercode (cf. 1.4 Scope of access) whether the file exists and if so returns certain information about it. If no <pack name> is specified then the default pack associated with the current usercode is queried.

If the <file name>/= option is used, a sorted list of files with a family name of <file name> is returned. <file name> may be constructed according to the rules for the family name portion of <MCP file name>.

If the "=" option is used, a sorted list of the files in the current user's directory is returned.

If the 'ON <pack name>' clause is not used, then the file search will only occur on the current user's default pack. Thus, if the current user has a default pack other than system disk, to inquire of files on the system disk, an explicit 'ON ""' must be used.

Examples:

```
FILE IBASIC
FILE *XY
FILE X/Y ON P
FILE (PQR)/ZXY ON USER
FILE BLACKJACK ON ""    (look for BLACKJACK on system pack)
FILE =
FILE *IBASIC/= ON ""
```

FIND

```
>- FIND -- <variable name> --------------------------------------#
        |                          | |                           |
        |- <line number -----------| |- <line number range> -|
        |                          |
        |- <line number range> ----|
        |                          |
        |- <user FN name> ---------|
        |                          |
        |- <delim> <string> <delim> -|
```

The FIND command displays the BASIC statements that reference the requested item. If a line number range is specified after a requested item then only that part of the BASIC program is searched. A request for a variable name (A-Z9, A$-Z9$) looks for both scalar and array references using the given name. A request for a line number returns only the statements which reference that line number. A request for a line number range returns references to within that range inclusively. A request for a user FN name (FNA-FNZ9, FNA$-FNZ9$) returns the statements which call, define or assign that name. A request for a delimited string does a search through the source statement(s) for occurrences of that string as a strictly literal string. The delimiter must be a non-alphanumeric character.

Examples:

FIND A$	(lists those program lines which reference A$)
FIND 100	(lists those program lines which reference line 100)
FIND 100-200 2000-3000	(lists those program lines, between lines 2000 and 3000 which reference lines 100-200)
FIND FNX 123-654	(lists those program lines between 123 and 654 which reference FNX)
FIND "X"	(lists those program lines which have the character X occurring in them)

FIX

```
>-- FIX ------------------------ <delim> - <text> - <delim> ->>
     |                        |
     |- <line number range> -|

    >------------------- <new text> ---------------------------#
                                         |            |
                                         |-- <delim> --|
```

The FIX command searches the specified line number range for the occurrence of the required text, replaces each occurrence of it within each line scanned by the new text and displays the new line. If <text> and <new text> are not the same length then the rest of the line will be shifted appropriately.

If the <line number range> is omitted then the whole program is scanned. <text> may be any string of characters excluding <delim> and may contain no characters, in which case the <new text> will be inserted after the line number of the line scanned and the rest of the line shifted right accordingly.

<new text> may be any string of characters excluding <delim>. If the trailing <delim> is omitted then any trailing blanks are not included in <new text>. <new text> may contain no characters, implying that occurrences of <text> are deleted from the scanned line shifting the rest of the line left accordingly.

<delim> may be any character other than a-z, A-Z, 0-9 or space.

If any change occurred then the resulting statement is checked for syntax and included in the current file, overwriting the previous line with that line number. If syntax error(s) result then the program may not be run until the syntax error(s) are resolved.

If the resulting statement exceeds 256 characters then the excess is simply truncated.

Examples:

```
FIX /3.14159/PI
FIX 100 .X/Y.Y/X
FIX LAST /END/   END  (move the end statement three chars to the
                       right)
FIX 2230 TO 2500 ::!: (make lines 2230 through 2500 comment
                       lines)
```

GET

```
>-- GET -------- <MCP file name> ------------------------------#
                                      |                  |
                                      |- ON <pack name> -|
```

The GET command searches the directory for <MCP file name> and proceeds to load the file into the BASIC environment. If any syntax errors are found in the file then suitable error messages are emitted and RUN's of the file are inhibited until these errors are fixed. If those syntactically incorrect statements are LISTed then they are highlighted (reverse video on TD820 and TD830 terminal types, preceded by an "*" otherwise).

Only those files within the current usercode scope of access are available. The default pack for the current usercode may be overridden by using the ON option.

Various parameters of the file are checked to ensure that it is a BASIC source file (e.g. file type is data, record size <= 256 bytes, number of records less than maximum allowed (c. 2000 records)). Any record in this file which does not start with a valid line number is not included in the loaded file.

Examples:

```
GET BLACKJACK            (load BLACKJACK from default pack)
GET PQR ON ""            (load PQR from system pack)
GET *MYPROG              (load MYPROG from system pack,
                          bypassing usercode defaults)
GET (HIS)/FILE           (access another user's file)
GET (MY)/FILE ON OTHER   (load a file from a specific pack)
```

HELLO

```
>-- HELLO -------------------------------------------------------#
            |                                          |
            |-- <usercode> ----------------------------|
                            |       |  |               |
                            |-- / --|  |- <password> --|
```

The HELLO command causes the log on or log off of a particular usercode. If just 'HELLO' is entered then a log off function is being requested and this will only happen if the current BASIC environment permits (see conditions for BYE command). If the function is allowed then the current BASIC environment is cleared and only HELLO, BYE and TEACH commands are allowed thereafter.

If <usercode> (and optionally <password>) is included then an implicit log off of the current usercode is performed, followed by a log on of the requested usercode/password pair.

Formation of <usercode> follows the same rule as for <BASIC file name> except that it may only be a maximum of 7 characters long.

Formation of <password> follows the same rules as for <BASIC file name>.

The <usercode> and <password> pair must be in the SYSTEM/USERCODE file if IBASIC was executed under a privileged usercode. If IBASIC is not running under an MCP usercode then <password> has no meaning and is not allowed (refer to 2.2.3 USercode considerations).

This command is ignored if the IBASIC system is not privileged.

Examples:

```
     HELLO ME/SECRET          (log ME on)
     HELLO                    (log off, if allowed)
     HELLO MYOWN              (log on, not running under
                               the MCP usercode system)
```

LIST

```
>-- LIST -----------------------------------------------------#
           |                          |  |              |
           |--- <line number range> ---|  |--- PRINT ---|
```

The LIST command causes the requested line(s) or all of the current program to be listed at the remote terminal. Any syntactically incorrect lines will be highlighted by means of reverse video (for TD820 and TD830 terminals) or a preceding "*" (USASCII67 2/10 character). If the line which would be executed if a CONTINUE or STEP command were to be entered is displayed, then it will be highlighted by means of bright video (for TD830 terminals) or a preceding ">" (USASCII67 3/14) (for all other terminal types).

If the PRINT option is requested then the list will be written to the file LINE (which is defined as a printer file) and not to the remote device.

Examples:

```
     LIST                     (list the whole program)
     LIST FIRST TO 100        (list up to line 100)
     LIST 1234                (list line 1234 only)
     LIST PRINT               (list the current file on
                               the host printer)
```

MAKE

```
>-- MAKE ---------- <BASIC file name> --------------------------#
```

The MAKE command clears the BASIC environment (under the same restrictions as the BYE command) and prepares for entry of a file to be called <BASIC file name>. If <BASIC file name> already exists then the MAKE is ignored.

Examples:

```
     MAKE NEWPROG
     MAKE "ODDNAME?"
```

MERGE

```
>-- MERGE -----------------------------------------------------
               |                                      |
               |-- <line number range> ---------------|
                                         |          |
                                         |-- FROM --|

    >-- <MCP file name> ------------------------------------------#
                               |                    |
                               |-- ON <pack name> --|
```

The MERGE command searches the directory for <MCP file name> and proceeds to load the requested portion of it (or the whole file if <line no range> is omitted) into the current BASIC environment. If a line that already exists is MERGEd, then the new MERGEd line overwrites the existing line.

If a syntax error is found in a MERGEd line then a suitable error message will be emitted and RUN's of the file are inhibited until these errors are fixed. If those syntactically incorrect statements are LISTed then they are highlighted (reverse video on TD820 and TD830 terminal types, preceded by an "*" otherwise).

Only those files within the current usercode scope of access are available. The default pack for the current usercode may be overridden by using the ON option.

Various parameters of the file are checked to ensure that it is a BASIC source file (e.g. file type is data, record size <= 256 bytes, number of records less than maximum allowed (c. 2000 records)).

Any record in the MERGEd file which does not start with a valid line number is ignored.

Examples:

```
     MERGE OTHER/FILE         (merge the whole of
                               OTHER/FILE into the current
                               environment)
     MERGE 1000 TO 2000 FROM OTHER/FILE
     MERGE LAST OTHER/FILE ON OTHERPACK
```

PASSWORD

```
>-- PASSWORD --- <old password> --- <new password> ------------>>

          >--------------- <new password> ------------------------#
```

The PASSWORD command changes tne password for the current usercode in the MCP usercode file and thus this command is only valid if the IBASIC system runs under an MCP usercode. The new password must be entered twice (identically) to ensure correct and intentional entry.

Example:

```
PASSWORD SECRET TOPSECRET TOPSECRET
```

RENAME

```
>-- RENAME --- <BASIC file name> ----------------------------->>
                                   |                    |
                                   |-- ON <pack name> --|

          >-------------------------- <BASIC file name>---------#
                |        |
                |-- AS --|
```

The RENAME command checks for the presence and availability of the first file on the requested pack, ensures the absence of the second file and then causes the MCP to change the name of the first file to the second file name. Only files within the current user's scope of access for change may be changed.

Examples:

```
RENAME ORANGES AS LEMONS    (on default pack)
RENAME PROG1 ON "" AS PROG  (force change on system pack)
RENAME X Y
```

RENUMBER COMMAND

```
>- RENUMBER ----------------------------------------------------#
            |                      ||                           |
            |- <line number range>-||-/ 1 \- STEP <increment>  -|
                                    |                           |
                                    |-/ 1 \- AT <line no> ------|
                                    |                           |
                                    |<--------------------------|
```

The RENUMBER command may be used to 'tidy' up the line numbers of (and references to) a part or whole of the currently loaded source program. The STEP parameter defines the amount to increment each new line number. The AT parameter defnes the starting value for the new numbers. If the STEP parameter is omitted then a value of 10 is assumed and if the AT parameter is omitted then a value of 100 is assumed. Before the actual renumbering is done, checks are performed to ensure that:

1. No overlap of existing statements would occur.
2. The order of execution of statements is not changed.
3. A previously unresolved line number reference would not become implicitly resolved by the renumber process.
4. The last line number in the program will not exceed 99999.

Examples:

```
RENUMBER                                  (renumber the whole program
                                           with implicit step 10 and
                                           starting with a value of 100)
RENUMBER 100 TO 1000                      (renumber the specified part
                                           of the program and all
                                           references to that part
                                           with the default parameters)
RENUMBER 5000 TO LAST AT 5000 STEP 100
                                          (renumber the last part of
                                           the orogram incrementing
                                           the line numbers by 100)
```

RUN

```
>-- RUN -----------------------------------------------------------#
       |                  |          |                 |
       |- <line number> -|          |-/ 1 \- TRACE ---|
                                     |                 |
                                     |-/ 1 \- PRINT ---|
                                     |                 |
                                     |<----------------|
```

The RUN command initiates the continuous execution of the statements in the current program file starting optionally from the specified line number. The BASIC data environment is cleared so that all numeric data items are zero and all string items have the value of the null string (""). The line number (if specified) must be a valid line number, as if a GOTO that line number were the first statement of the program.

The TRACE option causes the statements to be displayed on the remote device as they are executed.

The PRINT option causes the output from BASIC PRINT statements and the input from BASIC INPUT and LINPUT statements to be written to the file LINE as well as the remote device. The file LINE is closed on execution of an END statement in the BASIC program.

If both the PRINT and TRACE options are specified then the trace of statements executed is directed to the file LINE interspersed with the BASIC PRINT, INPUT and LINPUT output. In this instance the traced statements are not displayed on the remote device.

The PRINT and TRACE options have effect until the program is stopped (either by a STOP or END statement or a fatal error).

Examples:

RUN	(run the program from its first executable statement)
RUN TRACE	(run the program as above and trace execution)
RUN 1234	(run from line 1234 and clear the program's data variables)
RUN PRINT	(print output to be written to file LINE)

SAVE

```
>-- SAVE ------------------------------------------------------>>
          |                          | |                    |
          |-- AS <BASIC file name> --| |-- ON <pack name> --|

     >-----------------------------------------------------------#
          |                |
          |-- FOR CANDE --|
```

The SAVE command writes the current source program to a file on the current usercode default pack (unless overridden by the ON option) with the name of the current file (unless overridden by the AS option).

If the FOR CANDE option is used, an attempt is made to make the saved file compatible with CANDE BASIC files. This means that the line numbers will have leading zeros appended, if necessary, to make them 5 characters long and the resulting line is checked to be no longer than 80 characters. If a line exceeds 80 characters then the save is not done and a message is emitted to identify the line (or lines) which cannot be made compatible.

If the current workfile does not have a name associated with it then either the AS option must be used or the TITLE command must be used to associate a name with the workfile.

If the AX DEBUG WW or SW=W option is used then the current file will take the name of the AS file (if present), as if a TITLE command had been used.

Examples:

SAVE	(save the current workfile)
SAVE AS PQR ON P	(save the current workfile on pack P with name PQR)
SAVE FOR CANDE	(save the current workfile and attempt to make the file compatible with CANDE)

SCRatch

```
>-- SCR ---------------------------------------------------------#
        |        | |                                           |
        |- ATCH -| |- <BASIC file name> -----------------------|
                                          |                    |
                                          |-- ON <pack name> --|
```

The SCRatch command is used to clear the current BASIC code and data environment or to remove a file in the current usercode scope of access from the disk directory. If no file name follows the SCRatch then the clearing of the current file in the BASIC environment is assumed. Only if a file name is specified is anything removed from the disk (usercode default pack or explicit pack if the ON option is used). The SCRatched file is irrecoverably removed.

Examples:

```
     SCR                (clear the current workfile)
     SCRATCH OLDFILE    (remove OLDFILE from the user's directory)
     SCR BADFILE ON P   (remove BADFILE from user's directory on pack P)
```

SEQuence

```
>-- SEQ ---------------------------------------------------------#
         |         |  |                                       |
         |- UENCE -|  |--- /1\ - STEP - <increment> ----------|
                      |                                       |
                      |--- /1\ -------- <line number> --------|
                      |        |      |                       |
                      |        |- AT -|                       |
                      |                                       |
                      |<--------------------------------------|
```

The SEQuence command is used to initiate sequence mode on a screen type terminal. In sequence mode, line numbers are generated in sequence (starting at <line number> and incrementing by <increment>) at the top of the screen ready for statement entry. If the generated line number corresponds to a line that already exists, then the source line is presented in a fashion similar to edit mode. If, after entry of a line, a line is found between the last entered line and the next line in sequence, that intermediate line is presented as in edit mode.

Sequence mode is terminated if the generated line number exceeds 99999, if the pseudo break feature is used or if any command is entered in the normal fashion.

If <line number> is not specified, a value of 100 is assumed. If 'STEP <increment>' is omitted, a value of 10 is assumed.

Examples:

```
SEQ
SEQ STEP 100
SEQUENCE AT 1000
SEQ 1000 STEP 100
```

STEP

```
>-- STEP --------------------------------------------------------#
```

The STEP command may be used when the BASIC environment is stopped after a RUN or WALK command has been used. It causes the next statement to be executed to be displayed and then executed. An implicit STOP statement is executed after this statement. Thus the STEP command enables the execution of a BASIC program statement by statement at the user's discretion. The same conditions and effects that applied to the CONTINUE command, concerning the state of the BASIC environment, apply to the STEP command.

TEACH

```
>-- TE ----------------------------------------------------------#
        |       |     |                |
        |- ACH -|     |-- <command> --|
```

The TEACH command causes a syntax diagram of the requested command to be displayed on the remote terminal. If no particular command is requested then a list of available commands is returned.

Examples:

```
TEACH
TE XREF
TEACH CONT
```

TITLE

```
>-- TITLE ------------ <BASIC file name> ----------------------#
```

The TITLE command changes the name of the BASIC environment and this name will be used as the file name for a SAVE (unless explicitly overridden by the SAVE AS syntax).

Example:

```
     TITLE THIS     (the current workfile now has the name THIS)
```

USER

```
>-- US ----------- <usercode> -----------------------------------#
        |      |                |       |  |              |
        |- ER -|                |-- / --|  |- <password> -|
```

The USER command is identical in function to the HELLO command except that it may not be used just to log off. A usercode/password is required after the key word 'US' or 'USER'. This command was intended to enable the SMCS auto log on feature, but its use is general.

Example:

```
     USER ME/SECRET
```

WALK

```
>-- WALK --------------------------------------------------------#
             |                    |
             |-- <line number> --|
```

The WALK command initiates the execution of the BASIC environment, clearing the BASIC data environment and causing execution to be halted after the first statement has been executed. The statement executed is displayed on the remote terminal. The presence of the optional line number implies a GOTO that line number before execution begins. If the line number is invalid a suitable message is returned. The intent of this command is to enable single stepping through a BASIC program from the beginning. The STEP or CONTINUE command may be used to continue program execution after this command.

Examples:

```
     WALK          (run the first executable statement of the
                    current workfile)
     WALK 1234     (run line 1234 as the first executable statement
                    of the current workfile)
```

WHAT

```
>-- WHAT --------------------------------------------------------#
```

The WHAT command returns a statement of the status of the current BASIC environment.

WHERE

```
>-- WHERE ------------------------------------------------------#
          |              |
          |--- FROM ---|
          |              |
          |-- CALLED --|
```

The WHERE command returns a statement of where program execution would continue if a CONTINUE or STEP command were to be given.

If the FROM option is used then the last few (no more than 20) statements (and commands) executed will be displayed.

If the CALLED option is used then the last few (no more then 20) GOSUB or user FN<x> calls are displayed.

Examples:

```
WHERE
WHERE FROM
WHERE CALLED
```

XREF

This command is synonymous with the FIND command described above.

```
>- XREF -- <variable name> ---------------------------------------#
        |                           ||                          |
        |- <line number> ------------||- <line number range> -|
        |                           |
        |- <line number range> ------|
        |                           |
        |- <user FN name> -----------|
        |                           |
        |- <delim> <string> <delim> -|
```

The XREF command displays the BASIC statements that reference the requested item. If a line number range is specified after a requested item then only that part of the BASIC program is searched. A request for a variable name (A-Z9,A$-Z9$) looks for both scalar and array references using the given name. A request for a line number returns only the statements which reference that line number. A A request for a line number range returns references to within that range inclusively. A request for a user FN name (FNA-FNZ9,FNA$-FNZ9$) returns the statements which call, define or assign that name. A request for a delimited string does a search through the source statement(s) for occurrences of that string as a strictly literal string. The delimiter must be a non alphanumeric character.

Examples:

XREF A$	(lists those program lines which reference A$)
XREF 100	(lists those program lines which reference line 100)
XREF 100-200 2000-3000	(lists those program lines, between lines 2000 and 3000, which reference lines 100-200)
XREF FNX 123-654	(lists those program lines between 123 and 654 which reference FNX)
XREF "X"	(lists those program lines which have the character X occurring in them)

BASIC COMMANDS (Calculator Mode)

The general form of a BASIC command is as follows:

```
>-- <one of a subset of defined BASIC statements> -------------#
```

A BASIC command is similar to a BASIC statement in function except that it is executed immediately and is required to be re-entered completely if it is desired to be executed again.

By definition, some BASIC statements have no meaning unless properly accompanied by another statement or statements. Therefore the following BASIC statements are disallowed as commands:

CALL	(not yet implemented)
DATA	
DEF	
DIM	
FNEND	
FOR	
GOSUB	(may allow this at a later date)
IMAGE or:	
NEXT	
ON <expr> GOSUB	(may allow this at a later date)
OPTION	
SUB	(not yet implemented)
SUBEND	(not yet implemented)
SUBEXIT	(not yet implemented)
User FN calls	(may be allowed at a later date)

BASIC commands which reference line numbers (eg. GOTO, ON<expr>GOTO, IF<rel expr>THEN) simply change the next statement pointer. Hence if the BASIC environment is stopped then CONTINUE continues execution from where the next statement pointer points (not necessarily where the environment stopped). If the BASIC environment is running then execution continues as if that BASIC command were the next statement and hence the flow of execution of the running program may be changed on the fly.

'One BASIC statement' includes constructs like:

```
IF <boolean expr> THEN <statement> ELSE <statement>
```

Thus if the 'THEN' or 'ELSE' statements are disallowed in command mode then the whole 'IF' statement is disallowed.

STOP and END COMMANDS

The STOP and END statements, when executed as commands, may take on a special meaning. If a program is running or is stopped but able to be continued, the STOP or END commands have the same effect as a STOP or END statement, (i.e. they terminate the current program's execution such that continuation is prohibited). If a program is stopped, but not able to be continued, (i.e. a RUN or WALK statement must be used to cause a program to execute) then a STOP or END command will terminate the current 'session' by closing the printer backup file created by the PRINT option on the RUN, CONtinue and LIST commands. Thus a 'session' begins at the first LIST PRINT, RUN PRINT or CONtinue PRINT and ends either when a STOP or END terminates it or when the user logs off.

The printer backup file created will have the user's usercode as family name and the file name will contain the date and time of the session in the form YYMMDDHHMM.

BASIC STATEMENTS

The general form of a BASIC statement is:

```
>-- <line number> ------- <any defined BASIC statement> --------#
```

BASIC statements may be entered in any line number order. If a statement with a particular line number is entered more than once then the last entered line is retained, and all previously entered lines with that number are lost.

Syntax of the statement is checked at the time of entry and the line is retained even if there is a syntax error. Some BASIC statements rely on other BASIC statements for complete correctness of syntax and function (eg. FOR and NEXT statements, GOTO another line number). Any errors relating to this type of statement will be detected and suitable error messages will be emitted when any attempt is made to execute the program (by means of the RUN, CONTINUE, WALK or STEP commands).

Thus there are two types of syntax error message. One is emitted at the time the statement is entered, with corresponding error message and pointer to the possible position of the error in the line, and the other is emitted when an attempt to run the program is made. These types of error message are not mutually exclusive, hence they may appear interspersed as a result of an attempt to run a program.

A BASIC statement may not exceed 256 characters in length.

The syntax of the implemented BASIC language statements are not documented here. The correct place to find them is in the BASIC Corporate Standard (please refer to page 1 of this document).

RECOVERY

The IBASIC system is able to recover the current user's BASIC source program if the central system or the IBASIC system fails. At user log on (HELLO time or at BOJ of IBASIC if auto log on is requested or at sign ON time if initiated with AX SMCS_AUTO_LOG) IBASIC checks the system for the presence of a 'workfile' left over from a previous session. If this file is present then IBASIC automatically reloads itself with the contents of this workfile. Only BASIC source is maintained in this file, so the value of data variables from the previous session is lost.

If the recovery is not wanted then either the workfile must be removed before log on or the recovered file is SCRatched after log on. In order to remove the workfile it is necessary to determine what its name would be.

If IBASIC was using the MCP usercode system then the workfile name will be '<default pack name>/(<usercode>)/WORKFILE<xx>' where <xx> is a unique pair of characters generated from the usercode index in the MCP usercode file (cf CANDE recovery ps.2212 5561 page 4-10).

If IBASIC was not using the MCP usercode system then the workfile name is called '<usercode>/(WORKFILE)'.

PSEUDO BREAK FEATURE

If a long function has been requested of the BASIC environment and it would be opportune to terminate the function (prematurely) then the pseudo BREAK feature may be used. A special message of the precise form '.BRK' (no leading or trailing spaces) may be entered to terminate the function. This is analogous to depressing the BREAK key to terminate a string of output messages to a TTY type device. (The BREAK key is supported for TTY type devices and will have the same effect as '.BRK'. However the BREAK key is only effective on these devices during an output message.)

This pseudo BREAK feature may be used (from any supported terminal type) to terminate a LIST command, a RUN, WALK, CONTINUE or STEP command (when there are many syntax errors being reported), a GET command, a SAVE command, a FIX command, a RENUMBER command and a CHAIN statement or command.

In addition '.BRK' may be used to terminate an unwanted automatic recovery of a previous session. If it is used in this context then that previous session is thereafter irrecoverable.

'.BRK' may also be used to interrupt the execution of an INPUT or LINPUT statement which is currently soliciting a response from the remote terminal. In the case of screen terminals '.BRK' must be typed over the input prompt character in the top left portion of the screen. Interruption of an INPUT or LINPUT statement also implies changing the state of the BASIC environment from 'running' to 'stopped'.

USE OF SPCFY KEY (TD820 and TD830 Terminals Only)

Depending on the state of the BASIC environment, the SPCFY key may be used as a 'shorthand' way of typing a particular function.

Primarily, the SPCFY key may be used in all the instances where the pseudo BREAK feature may be used. In these instances it performs the identical function as '.BRK'.

Secondarily, the SPCFY key may be used as a shorthand for the STEP command or the BREAK BASIC command, according to the state of the BASIC environment. Thus if the BASIC environment is in a state ready to continue execution from where it left off, then a depression of the SPCFY key has the same effect as typing in 'STEP' (i.e. causing the program to be single stepped). If the BASIC environment is running then the effect is as if 'BREAK' were typed (i.e. the program is halted and may be continued from that point).

It is suggested that the terminal be put to local, explicitly by depressing the LOCAL key, before depressing the SPCFY key to ensure that the SPCFY key does what was requested of it. (A particular problem with the TD830, while it is being sent messages, necessitates this course of action.)

"DOT" COMMANDS

```
>-- "." ---- <one or more dot commands> ------------------------#
```

Dot commands are special commands which are primarily intended for debugging purposes. They do not go through the normal process of compilation and execution and their syntaces are simple and words are delimited by spaces.

HELLO	returns the opening message.
TIME	returns the time this user has been logged on and the amount of cpu time so far accumulated this session.
XPRINT <8 digit hex number>	prints the value of the 8 digit hex number assuming it to be a floating point number. Infinite precision is assumed and up to 40 fractional digits are returned. (debug version)
OVERLAY	returns the number of data and code overlays IBASIC has performed since BOJ.
LOCAL	sets or resets a toggle which forces the remote terminal to be put to local after every command response.
DEBUG	sets or resets a debug toggle which enables various compiler tracing and dumping.
PRINT <n>	prints the value of real scalar word n (debug version).
VALSTR <string>	converts <string> to internal form then prints that value in internal hex and as if it was printed by BASIC print statement.(debug version)
SPLIT PRINT	displays the setting of the micro interpreter splitter mask.
SPLIT CLEAR	resets all the micro interpreter splitter mask bits.
SPLIT INITIAL	calls the micro interpreter to initialise the splitter mask bits.
SPLIT <n>	sets (or resets) the n'th bit in the micro interpreter splitter mask according to whether it was reset (or set) at the time.

DUMP	causes an MCP DUMPFILE to be created.
LOG	open (or close) print file of all input and output messages. This command sets or resets a toggle accordingly.
HINTS <string>	prints the contents of a memory area called HINTS in a reasonable format. If <string> is present then it is included in the heading of the print.
OL	returns the datacomm status of the remote station.
RY	changes STATION(READY) to true.
ST	changes STATION(READY) to false. This is only allowed from the ODT.
CONTINUOUS	switches the setting of continuous or wait mode for consecutive output messages. Be warned that CANDE will not know of the change but will assume the mode to have been left untouched.
SS <string>	sends (or displays) <string> on either the remote terminal or the system console (the opposite of where this message originated). WARNING the RMSG system option should be set for system console messages to be seen......
STATUSLINE	switches off (or on) the maintenance of the TD830 'status' line. (pre level 2 firmware in the TD830 does not implement the status line feature correctly)
FREEZE	freezes (or thaws) the IBASIC program in memory, i.e. freeze inhibits the MCP rollout process.
CASE	enables or disables the use of lower case letters in system responses. By default, lower case is enabled for TD820 and TD830 terminal types and disabled for TTY and TD800 terminal types.
PROMPT	switches the form of the prompt for user input from either a single "#" (at the top left of a screen for screen terminals) or the word "ready" as a prompt message.
BACKSPACE	sets the character which will be used as a

<new backspace char>	backspace character for TTY type terminals only. By default this character is "\" (USASCII67 5/12).
TRACE	initiates (or stops) a trace of s_ops on the printer. (Debug version only)
PRECISION <n>	sets the number of significant digits that the compiler looks at when converting a number from external to internal form. Normally set to 8. The intrinsics should be loaded or entered with a value of 8.
CHECK <n>	sets the time interval for the micro interpreter to run without returning control to the compile for the purpose of checking for interrupts. This value is set to 15 (equivalent to 1.5 seconds) by default.
TIMING	This command sets or resets a toggle for the special timing interpreter and will cause, or cease, timing of S-ops.
CANCEL	IBASIC/RUNNER will be cancelled as a result of this command and the splitter mask will be cleared.

AUTOMATIC LOG OFF

If the remote terminal is prematurely disconnected from IBASIC before a proper log off (BYE or HELLO command) has occurred then an automatic log off will occur regardless of the state of the BASIC environment. The remote user must re-establish connection and log on again to IBASIC in the normal manner. The source file that was loaded at the time of the log off will be recovered, but any data values will have been lost.

The mechanism that implements this feature is the TERMINATE ERROR mechanism in the network controller. Thus if IBASIC receives a TERMINATE ERROR message from the remote terminal with one or more of the relevant error conditions true then a log off procedure will be initiated. The following datacomm errors are relevant:

TIMEOUT
LOSS OF DSR
LOSS OF CARRIER
ADDRESS ERROR
TRANSLATE ERROR
FORMAT ERROR
READ NOT READY

The same procedure will be invoked if the SMCS sign OFF command is used before proper log off procedures have occurred.

Thus, it is strongly recommended that the remote user log off IBASIC in the proper manner if conditions allow.

EXTENSIONS TO THE ANSI LANGUAGE

MONITOR STATEMENT

```
>--------------------- MONITOR ------ COUNT --------------->>
   |                  |            |         |
   |-- <line number> --|            |-- START --|
                                   |         |
                                   |-- STOP ---|

>----- <numeric expression> ---- , ---- <numeric expression> ---#
```

This statement is implemented for the support of tne SPI monitor. The MONITOR statement may be used to COUNT, START or STOP a particular family/member event. The two numeric expressions that follow the respective keyword identify the family and member within that family. The first numeric expression refers to the family and must be in the range 1 through 15 after integerization. The second numeric expession refers to the member within the family and must be in the range 0 through 7 after integerization. If either expression value is out of range, a run time fatal error will occur.

RESERVED EXTENSIONS TO THE ANSI LANGUAGE

The following extensions to the ANSI language are recognised <u>only</u> when IBASIC is executed with SW = W or by suppling AX DEBUG WW in the initial AX message (see INITIAL PROGRAM PARAMETERS). The intent of these extensions is to enable the writing of external intrinsic functions and subprograms in BASIC. A side effect of this extension is that while IBASIC is being executed with SW = W or AX DEBUG WW, the intrinsic file it is using is inaccessible to any other copy of IBASIC. Hence it is desirable to have your own copy of the intrinsic file (not the system's) when you want to run in this mode (by executing with FI INTRINSICS NAM <your own name>). By default IBASIC will use the interpreter named *IBASIC/INTERP. If it is desired to use some other interpreter (say, for experimental reasons), then IBASIC should be invoked with SW = W or AX DEBUG WW and FI WORKFILE NAM <your own interpreter name>.

INTRINSIC STATEMENT

```
>-- <line number> -- INTRINSIC ---- <intrinsic name> --------->>

    >------------------------------------------------------------#
              |                   | |                      |
              |- ( <parameters> ) -| |- <local variable list> -|
```

The INTRINSIC statement must be the first statement of an intrinsic function. The <intrinsic name> must match an entry in the intrinsic function table in the compiler and the <parameters> must match in type and number to this entry. The local variable list may be any valid BASIC scalar name (up to 7 of them). The statements that follow may only reference those names that are either in the parameter list or in the local variable list. (Enforced at COMPILE or RUN time). If the intrinsic being defined is RUN then it supercedes the intrinsic in the intrinsic file. The external intrinsic file is only changed after a COMPILE INTRINSICS command is executed. While the 'local' intrinsic is being run, the parameters and the locals are actually global. The compilation process and ensuing intrinsic call process 'localise' the references.

The parameters may be arrays which will be passed by reference. Array parameters are only allowed if the intrinsic requires them and are denoted by appending a '(' to the array name which will be passed. Within the intrinsic the array may be referenced with either one or two subscripts (with obviously bad results if a two dimensional array is accessed with one subscript and vice versa). The special function NDIM should be used to ensure correct usage.

Within the intrinsic definition the following statements are illegal:

Array references (unless they are passed as parameters)
CHAIN
CLOSE
DATA
DEF (and any FN references)
DIM
END
FNEND
INPUT
LINPUT
MARGIN
MAT
OPTION
READ
RESTORE
SCRATCH
STOP
SUB
SUBEND
SUBEXIT

INTEND STATEMENT

```
>-- <line number> -- INTEND ----- <numeric expression> ---------#
                               |                         |
                               |-- <array name> ( -------|
```

The INTEND statement must be the last statement in the intrinsic. Its function is to return the value of the intrinsic or to return an array name to the calling code. There must be a corresponding INTRINSIC statement. Normal BASIC statements or another INTRINSIC statement may follow.

ERROR STATEMENT

```
>------------------- ERROR( --- FATAL ---- , -- <integer> ) ----#
 |                 |        |            |
 |- <line number> -|        |- NONFATAL -|
```

The ERROR statement will cause the printing of an error message which is the <integer>'th one in the procedure SPELL_ERROR in the compiler. If NONFATAL is specified then execution will continue. If FATAL is specified then the intrinsic and thus the program (or command) execution is terminated immediately.

The following errors are currently used:

14 - numeric overflow
15 - numeric underflow
102 - zero argument
103 - negative argument
104 - negative number to nonintegral power
105 - argument too big, inaccurate result
106 - argument out of range
108 - zero to negative power
133 - square matrix required
134 - array dimension mismatch

HMON STATEMENT

```
>-- <line number> --- HMON <numeric expression> ---------------#
```

The HMON instruction causes the least significant 8 bits of the 'integerised' value of the numeric expression to be displayed on the hardware backplane. This feature is intended for use by an external timing device. The hardware monitor micro instruction is invoked. Negative values of the numeric expression will have the effect of using the value 255 (@FF@ in hex). Values of the expression in excess of 255 will have the effect of <expression> mod 256.

ALLOCATE

```
>- <line number> - ALLOCATE - <array name> - (-<numeric expression>-)--#
                            |                 |                    |  |
                            |                 |<---- , --- /1\ ----|  |
                            |                                         |
                            |<------------- , ---------- /6\ ---------|
```

The ALLOCATE statement is intended for use in some intrinsics that need local array storage. The statement allocates space for the array(s) on the top of scalar space and does <u>not</u> change the top pointer. Hence, care must be exercised using this statement to avoid unintentional overlapping allocation of arrays. For correct usage, there must be no ALLOCATE statements executed, no intrinsics (which use ALLOCATE statements) called and no implicit allocation of arrays caused between the first ALLOCATE statement and the corresponding return from the intrinsic.

SPECIAL FUNCTIONS

ADDR(A) returns the starting address of the array A. If the array has not been allocated space yet, this value will be zero. This function may also be used to determine if two arrays have been allocated the same space.

HOST$	returns a string value which is the setting of the host system's HOSTNAME attribute. If no HOSTNAME has been set for the system, a null string is returned.
NDIM(X)	returns the number of dimensions of array X.
MXI	returns the maximum integer that the current numeric representation may hold precisely.
RDUC(X)	returns the value of X such that .5 <= X < 1 as if X had been divided or multiplied by 2 repetitively until X is in that range.
XPND(X,Y)	returns the value of X as if multiplied by 2**Y.
XPON(X)	returns the power to which 2 must be raised such that X/2**XPON(X) results in .5 <= X < 1
XTIM	returns the current cpu time usage of IBASIC in 1/10'ths of a sec.

SPECIAL VARIABLE NAMES

DET	when used as a destination in an assignment statement assigns a value to the function DET (no parameters).
RAN1,RAN2,RAN3	may be used as sources or destinations. They are initialised to particular values at the beginning of any particular RUN of a program and are intended for use by the RND intrinsic function.

COMPILE COMMAND

```
>-- COMPILE ----- INTRINSICS ----------------------------------#
```

The COMPILE command is used to convert the 'locally' defined intrinsic function to a form which may be put into the external intrinsic file. This code is then written to the intrinsic file overwriting whatever code may have already been 'compiled' for the function or functions being defined locally. The local definition of the function is used until the local definition is deleted from the workfile. If a syntax error exists in the workfile then nothing is written out to the external file and command mode invocations of the intrinsic function will yield an 'invalid op code' run time error message.

EXTERNAL INTRINSICS

The following intrinsic functions are defined as external intrinsics and are maintained in a 'compiled' form in the intrinsics file:

SIN	
COS	
TAN	
SINH	
COSH	
TANH	
CSC	
SEC	
COT	
ASIN	
ACOS	
ATN	
ANGLE	
LOG	
LOG2	
LOG10	
EXP	
RND	
PWR	(invoked by the exponentiation operator e.g. X¬Y)
MDOT	(dot product of two vectors)
MDET	(calculate determinant of a matrix)
MADD	(matrix and vector addition)
MSUB	(matrix and vector subtraction)
MMUL	(matrix multiplication)
MSMUL	(scalar multiplication of a matrix)
MINV	(inversion of a matrix)
MIMUL	(in place matrix multiplication)

IMPLEMENTATION DEFINED VALUES

The ANSI standard leaves certain aspects of the language implementation undefined or not explicitly defined.

The following list should define any implementation parameter which is not explicitly defined in the standard (X3J2/79-51):

sec. 4.1.6	lowercase characters are mapped directly on their uppercase equivalent everywhere except within quoted strings.
sec. 4.2.2	a line number may contain up to 5 digits with a maximum value of 99999. end of line is defined to be either the end of the record or the first occurrence of a NUL, CR, LF or ETX character. a line may contain up to 256 characters.
sec. 5.1.4	numeric values are maintained with a precision of at least 6 decimal digits (6.32 to 7.22 digits depending on the value). numeric values may range from approximately 5.39761E-79 to 7.23701E+75.
sec. 5.1.6	in all cases throughout the implementation, numeric underflow is treated as a nonfatal exception.
sec. 5.2.4	at the initiation of execution of a BASIC program all numeric variables are defined to have the value zero.
sec. 5.4.5	a nonfatal exception occurs if arguments to the SIN, COS and TAN functions are so large that a meaningful result cannot be obtained and the argument is taken to be zero.
sec. 6.2 4	a string variable may contain up to 255 characters if its maximum length has not been explicitly defined. at the initiation of execution of a BASIC program all string variables are defined to have the value of the null string.
sec. 6.6.4	OPTION COLLATE NATIVE will cause the EBCDIC collating sequence to be used.
sec. 7.1	arrays may not be defined to have more than 4095 elements each. the value for the string length-fixer must be in the range 1 to 255.

sec. 8.1.4 a file name in an OPEN statement is of the form:

```
>-- <MCP file name> ------------------------#
                       |                  |
                       |-- ON <pack name> --|
```

if the record size for a file is not explicitly defined then the record size is assumed to be 180 characters or the record size of the file if it pre-exists. (For internal format files this size is 36 numeric items or the record size for the file.)

the maximum length of a record is 255 characters (51 internal format numeric items).

sec. 8.1.6 a maximum of 16 files may be open simultaneously.

sec. 10.1.2 a maximum of 7 parameters are allowed in a user defined function definition.

sec. 10.1.4 if the value of a user defined function has not been explicitly assigned then the value returned is either zero or the null string according to the type of the function.

sec. 10.3.4 the form of a program designator is identical with that of the file name in the OPEN statement.

sec. 11.2.2 end-of-line is defined identically to that in section 4.2.2.

sec. 11.2.6 the default input-prompt is a question-mark followed by a space.

sec. 11.3.4 the significance-width is 6 and the exrad width is 2.

end-of-line is either the end of the record or a NUL character.

the print zone width is 15 and there will be as many zones as fit in the current margin.

in the absence of an explicit MARGIN statement the margin for file 0 is the width of a line of the attached terminal. (80 characters for TD800 screens, 255 for TC4000, 72 for TTY).

if the value of a TAB expression is so large that significance is lost then a nonfatal exception occurs.

sec. 11.3.6 a trailing space is always generated when printing a numeric item regardless of the state of the current columnar position and the margin setting.

INDEX