SPRINT 1.1
USER SUPPLEMENT

CONTENTS

OASIS BASIC 1.1	page 1
Errors in the manual	page 1
Ammendments to the manual	page 1
Software support	page 1
Appendix A Compilation error codes	page 2
Appendix B Execution error codes	page 3
Appendix C OASIS BASIC description	page 4
Appendix D Summary of differences From DRAGON BASIC	page 11
Appendix F Some hints if you have difficulties	page 13

PLEASE READ APPENDIX F BEFORE USING THIS COMPILER

OASIS BASIC 1.1

OASIS BASIC 1.1 contains all the facilities of OASIS BASIC 1.0 plus commands and functions to enable cassettes and printers to be used.

The following changes have been made :-

* The EOF function has been added.
* The PRINT command has been changed to allow output to printers (#-2) and to cassettes (#-1).
* The INPUT and LINE INPUT commands now allow input from cassettes (#-1).
* The POS function has been extended to work with printers.
* OPEN and CLOSE commands to use cassettes have been included.
* PRINT TAB and PRINT USING have been included.
* AUDIO and MOTOR commands have been included.

Errors in the Manual

Page 6 REM$ A$ [255] , X$(10,10) [10]
should read REM$ A$ [254] , X$(10,10) [10]

Page 6 "....causes the compiler to leave 255 bytes"
should read "....causes the compiler to leave 254 bytes"

Page 18 4 PMODE 4,1 : PCLS : SCREEN,1
should read 4 PMODE 4,1 : PCLS : SCREEN 1,1

Ammendments to the manual

Version 1.1 is an upgrade of version 1.0 and therefore the appendices A to D have been rewritten and are included in this supplement. Please ignore the appendices in the main manual.

SOFTWARE SUPPORT

Sprint is fully supported with upgrades issued periodically. Upgrades when required are available for one pound seventy five pence.

If you experience any problems whatsoever please do not hesitate to contact us at:

Oasis Software,
Alexandra Parade,
Weston super Mare,
Avon.

Or telephone us on 0934 419921 and ask for John Gross. We will be only too pleased to help.

Appendix A Compilation error codes

Code		Error description
000	:	Integer constant expected.
001	:	String constant expected.
002	:	"(" expected.
003	:	")" expected.
004	:	"[" expected.
005	:	"]" expected.
006	:	"," expected.
007	:	";" expected.
008	:	":" expected.
009	:	"@" expected.
010	:	ELSE expected.
011	:	TO expected.
012	:	THEN expected.
013	:	STEP expected.
014	:	End of line expected.
015	:	Name reserved in DRAGON BASIC.
017	:	NOT expected.
018	:	AND or OR expected.
019	:	Relational operator expected.
020	:	Multiplication operator expected.
021	:	Adding operator expected.
036	:	Integer identifier expected.
037	:	String identifier expected.
200	:	Integer too large.
201	:	No closing quotes for string.
202	:	DRAGON BASIC feature not supported.
203	:	Program too large.
204	:	Unary + or - can only be applied to integers.
205	:	This operation may not be applied to strings.
206	:	Operands of different types.
207	:	Integer expression expected.
208	:	String expression expected.
209	:	Invalid start of command.
210	:	Line number not in sequence.
211	:	Invalid item in DATA statement.
212	:	Line number expected.
213	:	2nd PCLEAR or value out of range.
214	:	Line number out of range.
215	:	Action in PUT wrong.
216	:	Array in PUT or GET must have two dimensions.
217	:	G parameter expected.
218	:	PSET or PRESET expected.
219	:	B or BF expected.
220	:	"-" expected.
221	:	"=" expected.
222	:	GOTO or GOSUB expected.
223	:	String or string variable expected.
224	:	";","," or end of statement expected.
225	:	Identifier expected.
226	:	Expression and variable of different types.
227	:	Error in FACTOR.
228	:	FN or USR function not defined.
229	:	TIMER may only be set to zero.

230 : DATA string too long.
231 : "/" expected.
232 : "64" expected.
233 : Array already in use -- definition ignored.
234 : Invalid directive.
235 : String variable encountered before - definition ignored.
236 : FN or USR definition expected.
237 : Function already defined.
238 : A directive must be the last command on a line.
239 : String too large.
240 : INLINE integer must be between 0 and 255.
241 : Invalid INLINE item.
242 : String variable expected.
243 : "H" expected.
244 : "ELSE" or end of line expected.
245 : Hex digit expected.
246 : Too many dimensions to an array.
247 : Too few subscripts.
248 : Too many subscripts or ")" expected.
249 : Invalid PUT option.
250 : Invalid file number.
251 : ON or OFF expected.

Appendix B Execution error codes

Code		Error description
A	:	Integer overflow.
B	:	Runtime stack overflow.
C	:	Range violation (e.g. subscript of array out of range).
D	:	RETURN with no GOSUB.
E	:	String value too long.
F	:	Division by zero.
G	:	Value less than zero (e.g. ON-GOSUB or ON-GOTO value less than zero).
H	:	Illegal character (e.g. trying to use CHR$ on an integer which is too large).
I	:	Empty string value.
J	:	RETURN missing.
K	:	Zero step in FOR command.
L	:	NEXT without FOR.
N	:	Error in READ, no data left.
O	:	Attempt to read string as an integer.
P	:	Attempt to read an integer as a string.
Q	:	Argument to standard function or command out of range.
R	:	Error in PMODE; insufficient memory for graphics mode and/or graphics page used by program.
S	:	Cassette not OPEN in correct mode.
T	:	Invalid option; "I" or "O" expected.
U	:	INPUT from cassette failure; end of file or not open for INPUT.
V	:	Error reading from cassette.

NOTE: Standard DRAGON BASIC error messages may also be produced.

Appendix C OASIS BASIC description

This appendix summarises the syntax of OASIS BASIC using EBNF (extended Backus-Naur Form). EBNF is used to give a series of (production) rules for the various objects of the program. Thus section 1 defines how programs are constructed out of lines.

With this notation the following conventions are used.

Objects in double quotes represent themselves.

e.g. "READ" implies that READ will be part of the program.

Alternatives are separated by |.

e.g. compound-command = if-command |
for-command |
next.

implies that a compound-command can either be an if-command, a for-command or a next.

Definitions are terminated by a period.

Items enclosed in {} may be repeated zero or more times.

Items enclosed in [] may be repeated zero or once.

Items of the form (a|b|..|z) mean that one of a,b,..,z must be selected.

The following is a description of OASIS BASIC.

1. PROGRAMS

A program consists of a series of numbered lines 0,1,..,N. Each line consists of the line number followed by one or more commands separated by colons.

program = line {line}.

line = line-number command-sequence.

command-sequence = command {":" command}

line-number = integer

integer = digit {digit} | "&H" hex-digit {hex-digit}

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
"9".

hex-digit = digit | "A" | "B" | "C" | "D" | "E" | "F" .

NOTE

THE N+1 lines of the program must be numbered 0,1,2,..,N.
This can be achieved by using RENUM.

2. EXPRESSIONS

An expression can compute either an integer or a string value.

```
expression = and-expression {"OR" and-expression}.

and-expression = sub-expression {"AND" sub-expression}.

sub-expression = ["NOT"] sub-expression |
                 relational-expression.

relational-expression = simple-expression {relational-operator
.....                                      simple-expression}.

simple-expression = term {adding-operator term}.

term = factor {multiplying-operator factor}.

factor = unary-operator factor | "(" expression ")" |
         function-call | variable | integer | string.

relational-operator = "<" | ">" | "<=" | " >=" | "=" | "<>" |
                      "><" | "=>" | "=<".

adding-operator ="+" | "-".

multiplying-operator = "*" | "/" | "\".

unary-operator = "+" | "-".

variable = identifier [subscripts].

subscripts = "(" expression {"," expression }")".

string = '"' any-sequence-of-characters-except-" '"'.

function-call = identifier parameter-list.

parameter-list = "(" expression {"," expression } ")".

identifier = letter  {(letter | digit)}  ["$"].

letter = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
         "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
         "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
```

The available functions are:-

ASC	CHR$	INKEY$	STR$	VAL	ABS
FIX	INT	JOYSTK	PEEK	POINT	PPOINT
RND	SGN	VARPTR	LEFT$	RIGHT$	MID$
LEN	POS	HEX$	INSTR	STRING$	EOF
USR0 .. USR9		FNA .. FNZ			

The following standard variable exists :-

TIMER

Notes

(a) FNA..FNZ and USR0.. USR9 are user defined functions.

(b) Numeric expressions and variables use integer values in the range -32768..32767.

(c) "/" is integer division and "\" is integer remainder.

(d) INT and FIX are the identity function, i.e. INT (e) = FIX(e) = e. However they are included for compatibility, e.g. the following code can still be used to find the remainder of dividing X by Y.

```
X - INT(X/Y) * Y.
```

(e) As with DRAGON BASIC only "+" can be used with strings.

(f) RND may only take parameters >= 1.

(g) PEEK takes one integer parameter in the range -32768..32767. Negative values map onto values in the range 32768..65535,i.e. -1 = 65535 ,.., -32768 = 32768.

3. COMMANDS

```
command = BASIC-command | sound-command | graphics-command.
```

3.0 SOUND COMMANDS

Both PLAY and SOUND are supported.

(a) SOUND integer-expression, integer-expression.

(b) PLAY string-expression.

Note

All facilities of PLAY except execution of substrings are supported.

3.1 GRAPHICS COMMANDS

All graphics commands are supported.

(a) CIRCLE (x,y),r,c,h,s,e
where x,y,r and c are integer expressions, and h,s and e are of the form (integer-expression) / 64.

(b) COLOR integer-expression, integer-expression.

(c) DRAW string-expression.

(d) GET (x1,y1)-(x2,y2),array,G

PUT (x1,y1)-(x2,y2)array,action

where x1,x2,y1 and y2 are integer expressions and "array" is a two dimesional integer array variable.

(e) LINE (x1,y1) - (x2,y2),a,b

where x1,x2,y1 and y2 are integer expressions.

(f) PAINT (x,y),c,b.

where x, y, c and b are integer expressions.

(g) PCLS c

where c is an integer expression.

(h) PCOPY a TO b where a and b are an integer expression.

(i) PMODE m,p.

where m and p are an integer expression.

(j) PRESET (x,y)

PSET (x,y,c)

RESET (x,y)

SET (x,y,c)

where x, y and c are an integer expressions.

(k) SCREEN t,s

where t and s are an integer expression.

(l) PCLEAR integer.

Notes

(a) In DRAGON BASIC the last parameters of CIRCLE may be real values. However in this version of BASIC we require integer values. Using the above syntax maintains compatibility, i.e. (32)/64 represents 0.5 in DRAGON BASIC and 32 in this version of BASIC.

(b) All facilities of DRAW except the execution of substrings are supported.

(c) PCLEAR takes an integer parameter in the range 0 to 8.

3.2 BASIC COMMANDS

```
BASIC-command = definition | simple-command |
                compound-command | read-command |
                control-command | machine-comand |
                comment | inline-command.
```


(a) Definitions

```
definition = "DEFFN" letter "(" identifier ")""=" expression |
             "DEFUSR" digit "="( [("+" | "-")] integer | identifier)
             ("REM$"|"'$'") directive {"," directive }|
             "DIM" array-spec {"," array-spec}.
array-spec = identifier "(" integer { "," integer } ")".
directive = string-def | compiler-option.
string-def = (identifier | array-spec) "[" integer "]".
compiler-option = ("C" | "L" | "N")  ( "+" | "-").
```

Notes

(a) The dimensions of an array may not be defined by variables.

(b) DEFUSR may be defined with an integer value in the range -32768..32767.

(c) A string definition is used to specify the size of a string variable or the size of components of the string array. If a string variable is encountered before such a definition it is given a default size (i.e. 32.)

For example REM$ A$[10], B$(20)[30] defines a variable A$ which can hold strings up to 10 characters long and a string array B$ of 21 components each of which can be a string of up to 30 characters. To be effective REM$ definitions should contain the first "textual" occurance of the variables being defined.

A compiler option is used to control the compiler. "+" is used to turn on the given option and "-" is used to switch it off.

C - controls generation of runtime checks.

L - controls the production of a listing.

N - controls the generation of code to record line numbers.

A directive command must be the last command on a line.

(b) SIMPLE COMMANDS

```
simple-command = "CLS"[expression] | "END" | "STOP" |
                 "CLEAR" integer ["," integer] |
                 input-command | print-command | assignment
                 OPEN | CLOSE | AUDIO | MOTOR .
```

```
open  = "OPEN"  expression  "," "#-1",expression .

close = "CLOSE" "#-1" .

audio = "AUDIO" ("ON" | "OFF") .

motor = "MOTOR" ("ON" | "OFF") .

input-command = "INPUT"[input-option]variable{","  variable} .
                "LINE" "INPUT" [input-option] variable .

print-command = "PRINT"[print-option-1[ ","]]
                       [print-option-2]
                        print-list .

input-option = "#-1" "," | expression ";" .

print-option-1 = "#-1" | @ expression | "#-2" .

print-option-2 = "USING" expression ";" | "TAB" "("
                 expression ")"[("," | ";")] .

print-list = [expression]{("," | ";")[expression]} .

assignment =     ["LET"] variable "=" expression.
```

Notes

i) END and STOP both cause the program to terminate.

ii) CLEAR has no effect.

iii) The expression in an INPUT command is restricted to be a simple variable, a string or an arbitrary expression in parentheses.

(c) CONTROL COMMANDS

```
Control-command = "GOSUB" line-number | "GOTO" line-number |

                  "ON" expression "GOSUB" line-list |

                  "ON" expression "GOTO" line-list | "RETURN".

line-list = line-number { "," line-number }.
```

(d) COMPOUND COMMANDS

```
compound-command = if-command | for-command | next.

next = "NEXT" identifier { "," identifier }.

for-command = "FOR" identifier "=" expression "TO" expression
              ["STEP" expression].

if-command = "IF" expression "THEN action ["ELSE" action].

action = line-number | command-sequence.
```


(e) MACHINE COMMANDS

```
machine-command = "EXEC"(expression | "@" identifier) |

                  "POKE"  expression "," expression.
```

Note

All expressions in machine commands are integer expressions in the ranges -32768..32767 or 0..255. Negative values map onto integers in the range 32768..65535. •

(f) COMMENTS

```
comment = ("REM" | "'") any-sequence-of-characters.
```

(g) READ COMMANDS

```
Read-command = "READ" variable {"," variable} | "RESTORE" |

               "DATA" data-item {"," data-item}

data-item = [("+" | "-")] integer | string.
```

Note

String items must be enclosed in quotes.

(h) INLINE COMMANDS

```
inline-command = "INLINE" inline-item {"," inline-item}.

inline-item = integer | identifier ["("")"].
```

An inline command can be used to embed p-codes in a program. An inline-item which is an integer must be in the range 0 to 255. If an inline-item is an identifier then the address of the variable is generated as 2 bytes.

Appendix D Summary of differences from DRAGON BASIC

The main differences are as follows:-

(a) Integer values are used; not floating point.

(b) String variables have a fixed maximum size.
This size can be user defined or be the default size.

(c) Variables names cannot be used at runtime.

(d) Some new features (directives and the INLINE command) have been added.

(e) Some features of DRAGON BASIC have been omitted.

(f) Certain conventions on program layout need to be observed. For example reserved words and identifiers may not contain spaces.

Some detailed differences are as follows:-

Statement	comments
CLEAR n,h	No effect. n and h must be integers.
CLS c	Same.
DATA	String values must be in quotes.
DEF FN	same.
DEFUSRnn=address	Address must be an integer in the range-32768.. 32767 or an array variable.
DIM	same.
END	same.
EXEC address	Address must be present and be in the range -32768..32767.
FOR-NEXT	Same.
GOSUB	Same.
GOTO	Same.
IF THEN ELSE	Essentially the same.
INPUT	Prompt expression more general.
LET	Same.
LINE INPUT	Prompt expression more general.
ON GOSUB	Same.
ON GOTO	Same.
POKE location,value	Location must be an expression yielding a value in the range - 32768..32767.
PRINT	Same.
PRINT TAB	Same.
PRINT USING	Same.
PRINT @	Same.
READ	A numeric item may not be read as a string.
RESTORE	Same.
RETURN	Same.
STOP	Same as END. CONT cannot be used to continue the program.
PLAY string	The string may not contain the X command. There will also be a limit on the size of the string.

SOUND	Same.
AUDIO	Same.
CLOAD	Not included.
CLOADM	Not included.
CLOSE	Same.
CSAVE	Not included.
CSAVEM	Not included.
EOF	Same.
INPUT#-1	Same.
MOTOR	Same.
RENUM *	Not included.
NEW	Not included.
LLIST	Not included.
CONT	Not included.
DEL	Not included.
EDIT	Not included.
LIST	Not included.
RUN	Not included.
OPEN	Same.
PRINT#-1	Same.
SKIPF	Not included.
TRON	Not included.
TROFF	Not included.
PRINT#-2	Same.
CIRCLE(x,y),r,c,h,s,e	Special syntax for h,s and e.
COLOR	Same.
DRAW string	The string may not contain the X command. There will also be a limit on the size of the string.
GET	Same.
PUT	Same.
LINE	Same.
PAINT	Same.
PCLEAR n	n must be an integer in the range 0..8.
PCLS	Same.
PCOPY	Same.
PMODE	Same.
PRESET	Same.
PSET	Same.
RESET	Same.
SCREEN	Same.
SET	Same.
ASC	Same.
CHR$	Same.
HEX$	Same.
INKEY$	Same.
INSTR	Same.
LEFT$	Same.
LEN	Same.
MID$	Same.
RIGHT$	Same.
STRING$	Same.
STR$	Same.
VAL	Same.
ABS	Same.
ATN	Not included.

COS	Not included.
EXP	Not included.
FIX	Same.
INT	Same.
JOYSTK	Same.
LOG	Not included.
MEM	Not included.
PEEK(n)	n must be an expression in the range -32768..32767.
POINT	Same.
POS	Same.
PPOINT	Same.
RND(n)	n>=1.
SGN	Same.
SIN	Not included.
SQR	Not included.
TAN	Not included.
TIMER	Same.
USRnn	Same.
VAPTR	Same.
EOF	Same.

Appendix F Some Hints If You Have Difficulties

ASCII SAVING

The Dragon manual is not very clear at all on the subject of ASCII saving, and many users, quite understandably have been making BASIC saves of their programs only to find a screen full of garbage when trying to load the BASIC program to be compiled.

When saving your program be sure to put ',A' after the filename in quotes, for example:

CSAVE "TEST",A (see page 3 of our manual)
CSAVE "TEST" will not work.

ASCII LOADING

The routines the Dragon uses to load blocks of ASCII code can leave a little to be desired if the cassette player stops between blocks as it does when compiling from tape. There are four common symptoms.

1. The computer compiles the program but reports erratic errors which make no sense and probably vary between successive attempts at a compilation.

2. The compiler compiles the program, no errors are reported, the tape just keeps running and the keyboard does not respond. No more blocks are being loaded.

3. As in 2 but after a while "TOO MANY LINES" is reported.

4. Persistent "TAPE ERROR" reports.

What is happening is that the tape recorder is "running on" and not stopping quickly enough between blocks to prevent it running on to the next block. This does not present a problem if you are doing a CLOAD without the Compiler because there is no delay between blocks. This can be easily cured by extending the length of the header to each block. To do this all that is required is to use the POKE 144 described in appendix E. We have found that doing a POKE 144,2 before doing your CSAVE "FILENAME",A gives sufficient lattitude for most tape recorders. With the Audio on, you will hear that the "whistle" at the start of each block has been lengthened.

If you still experience problem 2 there is another solution. Add some extra lines to your BASIC PROGRAM, REMS will do, and when the compiler requests maximum line number, quote the line number of the last line you wish to compile and ignore the extra lines.

UNDEFINED LINE

If the last line of your program contains an "IF" statement, errors may be generated. To circumvent this just add an extra line with an "END" statement. Note that the maximum line number this time is the line number of the "END" statement.

RENUMBERING YOUR PROGRAM

Before attempting to compile your BASIC program don't forget to renumber it in steps of 1 with the first line taking the number 0.

STRING TOO LARGE

Page 6 is incorrect. The maximum length of a string variable is 254 and not 255. Our apologies.

GET AND PUT

When Dimensioning Arrays to store data for "PUTS" and "GETS" remember that an integer variable uses only 2 bytes and not 5 as in the case of floating point variables.

LINKER ERROR

If "LINKER ERROR" is experienced during linking it means that the program is too large to be linked.

MID$

All three parameters are required. If the last parameter is omitted then error 006 will be generated.