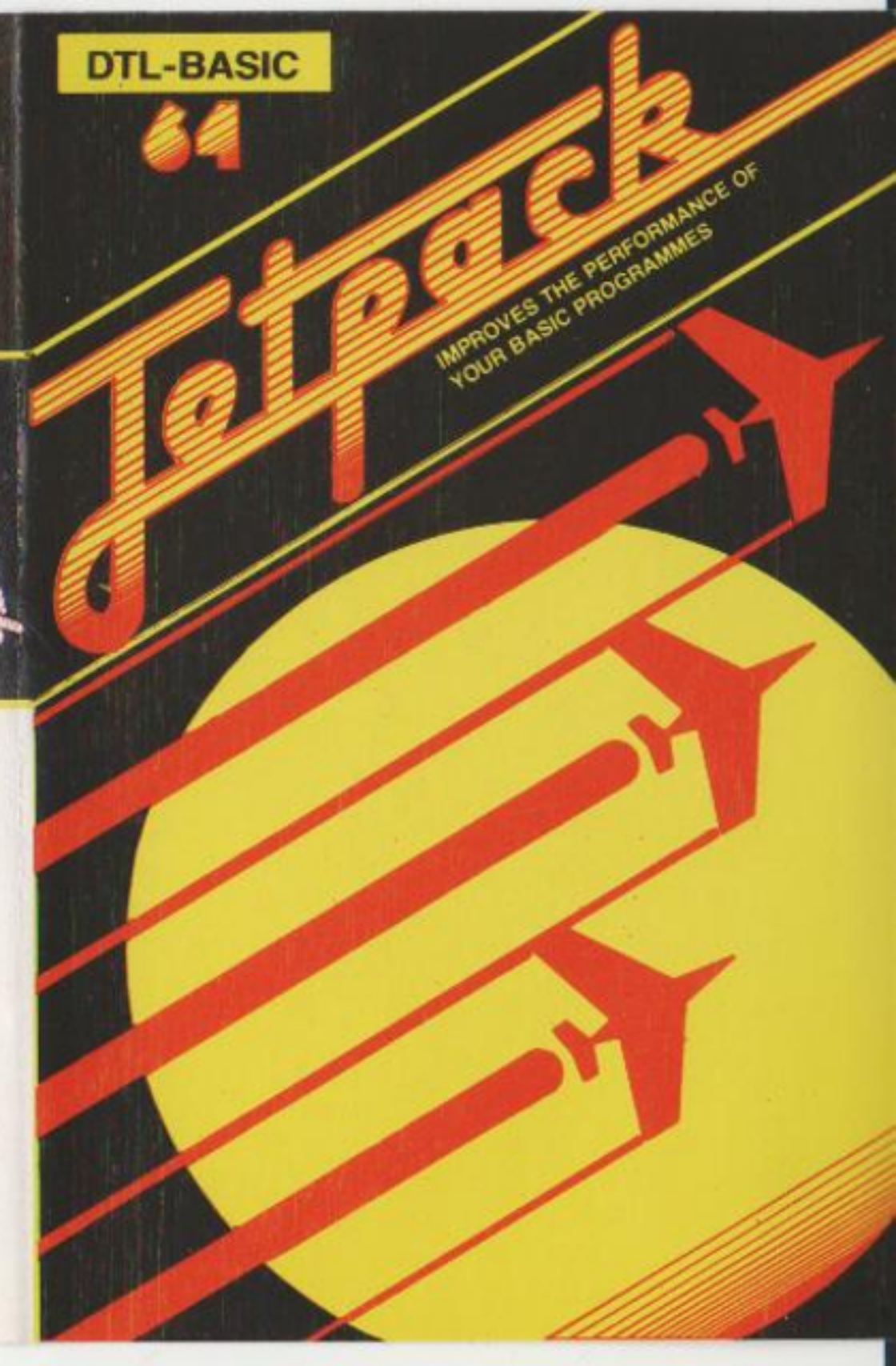
DTL-BASIC

64 Jetpack

IMPROVES THE PERFORMANCE OF YOUR BASIC PROGRAMMES



DTL-BASIC 64

Jetpack

By David Hughes

Published by: DATAVIEW WORDCRAFT LTD.
Radix House, East Street,
Colchester, Essex, CO1 2XB

Laadinstructies:

Cassette:

Stop de cassette in de recorder, typ "LOAD" en druk op RETURN. Als het laden klaar is typ dan "RUN".

Diskette:

Op de diskette staan twee compilers te weten DTL-BASIC en DTL-BASIC-E. De eerste is bedoeld voor de 1540/1541 disk-drive en de tweede voor de dubbele 4040 disk-drive. Deze zijn niet door elkaar te gebruiken!
Verder moet bij elk gebruik van de compiler de „dongle" in de cassette-poort zitten (met dongle wordt het blauwe blokje bedoeld dat bij de diskette geleverd werd).

198400487

DTL BASIC 64

INSTRUCTION MANUAL

Winner of the

Million Dollar Award
1983

Published by
DATAVIEW WORDCRAFT LTD
Radix House, East Street,
Colchester, Essex. CO1 2XB

CONTENTS

INTRODUCTION — DTL-BASIC 64 is a basic compiler for the CBM 64 that takes an existing program (known as the Source file) and produces the compiled equivalent of the program (known as the Object file). The compiled program will run very much faster than the original program and will also be much smaller.

Compiled programs may run up to 25 times faster in the best case but the speed-up factor does depend upon the program; typical improvements will be in the range 5 to 15 times faster. A fast garbage collection routine is included that avoids the very long delays that can occur in programs with a lot of strings. Compiled programs normally occupy between 50 and 80% of the space of the original program.

DTL-BASIC 64 is 100% compatible to CBM Basic and programs of up to 12k can be compiled. Also, compiled programs can use existing machine code subroutines without any alteration. The compiler can also compile programs containing extensions to Basic.

OPERATION OF DTL-BASIC 64 — It is usual before compiling a program to ensure that it is fully debugged, ie. that it works correctly.

Insert the DTL-BASIC 64 tape in the tape unit, LOAD and RUN the first program on the tape (DTL-BASIC). Do not press STOP as the program first loads two files of machine code. When prompted give the names of the Source and Object files and any printing requirements.

Start the compilation and follow the instructions to remove the compiler tape and replace it with the tape containing the program to be compiled.

Once the program has been loaded by the compiler the compilation process will start. The compiler makes two passes through the program (constantly displaying the number of the line being compiled).

When the second pass is complete (provided that no errors have been detected) the user is prompted to insert the tape that is to hold the compiled program in the tape unit.

The compiled program is written to tape and the user is asked if the Run-Time Library file (RTL-64) is to be written to tape after the compiled program. The user is then asked if the Line Number file is also to be written to tape. Normally RTL-64 will be required but the LN file will not be (see later for a description of these files). The compilation of the program is now complete and the user may choose to compile other programs.

OPERATION OF COMPILED PROGRAMS —
Operation of compiled programs is identical to that of uncompiled programs, ie. compiled programs are simply LOADED and RUN just like uncompiled programs.

The first time a compiled program is run after the machine has been turned on there will be a delay whilst it loads the file RTL-64. Each subsequent time that a compiled program is run there will not be a delay as it will detect that RTL-64 is already in memory. RTL-64 is a library of assembler routines used by compiled programs.

A copy of RTL-64 need not follow every compiled program on the tape, all that is necessary is to ensure that when the first compiled program is RUN after the machine has been turned on the tape in the tape unit contains a copy of RTL-64

Note that CONT cannot be used with compiled programs. SYS 2061 can be used instead of CONT. When a compiled program is stopped then variables can be printed on the screen (for debugging) just like uncompiled programs.

ERRORS — The compiler performs exhaustive checks whilst compiling and reports all errors found. For each error the line that contains the error is displayed followed by an error code (see the Appendix).

A different type of error message may occur during Pass 2 if a program contains a GOTO or a GOSUB to a line that does not exist. The error message takes the form of the line number of the line containing the GOTO or GOSUB followed by a "U", eg. 23510 U means that line 23510 contains a reference to an undefined line.

When a compiled program is run many further checks are made for errors that cannot be found during compilation. Such errors will generate similar messages to uncompiled programs (eg. NEXT WITHOUT FOR ERROR) but will give the address of the error rather than the line number.

The line number may be found by means of the ERROR LOCATE program supplied on the compiler tape. ERROR LOCATE uses the Line Number (LN) that may optionally be produced by the compiler. The procedure for locating the line in the program at which the error occurred is

— make a note of the address of the error;
— if the LN file was not created when the program was compiled then re-compile the program and create it;
— load and run ERROR LOCATE;
— when requested key in the program name and the address of the error;
— put the tape containing the appropriate LN file in the tape unit;
— ERROR LOCATE will scan the LN file and find and display the number of the line at which the error occurred.

COMPATIBILITY — DTL-BASIC 64 is 100% compatible with CBM Basic. This means that existing programs can be compiled without any alterations being necessary. DTL-BASIC 64 also keeps the layout of page zero, the variable and array lists the same as for the interpreter. This means that programs using machine code subroutines and PEEKS and POKES can also be compiled without alteration. The only proviso is that programs should not alter themselves or attempt to move the variable list by means of POKES.

DTL-BASIC 64 has a special feature to enable extensions to Basic to be compiled successfully. It is quite common for standard CBM Basic to be extended by means of extra statements types implemented by assembler routines in ROM or RAM. The way that extensions are handled by the compiler is as follows

— when the compiler finds a statement that starts with a character that it does not recognise then the compiler generates a warning message (because it is not sure whether the statement is an extension or a syntax error);
— the compiler assumes that the statement is an extension and embeds the statement in the compiled program exactly as it appears in the uncompiled program;
— when a compiled program is run then when any extension statements are to be obeyed control is passed to the interpreter which invokes the assembler subroutines to implement the extension. These subroutines work correctly because they find the memory organisation exactly the same as for uncompiled programs;

Note that assembler routines that implement extensions will not work correctly if they reside in memory locations used to hold RTL-64 ($AOOO — $BFFF) or the memory used by the fast garbage collection routine ($DOOO — $FFFF). The normal location for assembler routines is either $COOO to $CFFF or below $AOOO.

INTEGER ARITHMETIC — One very important way that the compiler speeds up programs is by using integer arithmetic wherever possible, ie. for operations between two integers. Even though the CBM interpreter supports integers it always converts them to reals (sometimes called floating point) before performing any arithmetic operation. Real arithmetic is very very much slower than integer arithmetic and for this reason the DTL-BASIC 64 provides true integer arithmetic.
To achieve as fast a compiled program as possible all real variables that can be should be converted to integers. One way of doing this is to put % characters after all references to the variables. This can be a lot of work so DTL-BASIC 64 includes a feature to do the conversion automatically during compilation. This is done by means of a directive to the compiler held in a REM statement, eg. REM ** CS(A,B,C) means Convert the Specified variables to integer (in this case variables and/or arrays A,B or C). Similarly REM ** CE(X,Y) means Convert all variables and arrays to integer excluding those named. Note that the directive MUST be at the start of the program (before any non-REM statements).
If a program using integers does not work correctly then recompile with the first statements of the program REM ** SI. This selects Special Integer (SI) mode which affects the operators divide and exponentiation. These operators differ from the other operators in that even when both parameters are integer they can give a real result. In normal compiled programs these operators will give an integer result for integer operands; this is much faster and is usually what is required but in some cases may not be, eg. when B% hold 3 the expression B%/2 * 4 will yield 6 on the interpreter and 4 when compiled in normal mode. The use of SI mode forces divide and exponentiation to give a real result in such cases and therefore gives the same result as the interpreter.

OTHER VERSIONS — DTL-BASIC 64 is also available on disk and this version has many advantages over the tape version, eg. compilation is faster and much more convenient, any size of program may be compiled, chained programs may share variables, the variable list may be located where desired in memory to leave space for such things as Sprite data etc., a special feature is provided for high speed Sprite movement.

A special version of the disk compiler DTL-PROTECTOR 64 is available that produces programs protected by a security key (or dongle) that fits on control port 2. For programs which are to be sold DTL-PROTECTOR provides a very high degree of security against software piracy.

A user of the tape version of DTL-BASIC 64 may upgrade to the disk compiler, contact Dataview Wordcraft for details.

Dataview Wordcraft can also supply separately the full compiler manual which describes both the disk and tape compilers. This manual gives a much more comprehensive description of the facilities of the compilers than is possible here. Contact Dataview Wordcraft for details.

The DTL-BASIC COMPILER is also available for Commodore 3000, 4000, 8000 and 700 series machines.

Dataview Wordcraft Ltd,
Radix House,
East Street,
Colchester,
Essex. CO1 2XB

APPENDIX

ERROR CODE LIST

ERROR NUMBER	CAUSE OF ERROR
1	syntax error
2	wrong type of operand
3	no 'TO' where one expected
4	illegal array subscript
5	no ')' where one expected
6	no '(' where one expected
7	no ',' where one expected
8	no ';' where one expected
9	no 'THEN' or 'GOTO' where one expected
10	no 'GOTO' or 'GOSUB'where one expected
11	no 'FN' where one expected
12	constant too big(either > 255 or < 0)
13	expression too complex (shouldn't occur if program is OK on Interpreter)
14	syntax error in expression
15	too many ')'s
16	illegal operator in string expression
17	type mismatch
18	illegal statement type (CONT or LIST)
19	program too big (shouldn't occur for disk based versions if program is OK on Interpreter)

APPENDIX

ERROR CODE LIST

ERROR NUMBER	CAUSE OF ERROR
20	a function name must be real
22	FOR variable cannot be an array element
23	wrong number of subscripts
24	integer too big
25	negative number illegal
26	cannot set ST,TI,DS or DS$
27	function variable must be real
28	no function where one expected
29	no operator or separator where one expected
30	type mismatch in relational expression
31	no line number where one expected
32	no operand where one expected
33	illegal CS or CE statement
34	bracket missing from CS or CE statement
35	too many conversion variables (> 128)
36	error in CS or CE;no ',' or '= >' after name
37	error in CS or CE;no '%' where one expected
38	converted name clash in CS or CE
40	no '=' where one expected
41	default array found in overlay
42	too much DATA text (maximum amount of DATA text is approximately 8500 bytes for the single drive compiler and 6500 bytes for the tape compiler — there is no limit for the dual drive compiler)