DEFT Pascal Compiler

Compile

1 Introduction

The **DEFT Pascal** Compiler is a program that allows you to create machine language programs from Pascal language source programs created with **DEFT Edit** or your own ASCII file text editor. The **DEFT Pascal** Compiler's features include:

- Generation of machine language programs, directly executable by the 6809 micro-processor, from Pascal language statements and declarations. Compiled programs can run many times faster than interpretive BASIC programs.
- Practically all of standard Pascal's language elements are supported.
- Program source files may be read from either cassette or disk with the resulting object files written to either cassette, disk, or the printer.
- Powerful compiler directives which provide the user with valuable compilation and source listing options, such as the option of having the assembler language representations of Pascal statements printed between the Pascal statements on the compiled program's source listing.
- Fully recursive compilation, which yields such flexibility as no fixed limitations on the number of dimensions to an array or table.
- Supports generation of recursive applications; programs that contain procedures that call themselves.

2 DEFT Pascal Compiler Operation

The command *LOADM "PASCAL":EXEC* will load the **DEFT Pascal** Compiler into memory from disk drive 0 and begin its execution, which is in two phases. In the first phase you will see the **DEFT Pascal** Compiler's screen with all of its prompts. This phase prompts the user to enter information required by the compiler for program compilation.

Upon the entry of the last prompted field, **DEFT Pascal** begins its second phase of operation. In this phase **DEFT Pascal** reads the source module file, parses the program statements, generates the corresponding machine instructions, saves the machine program version in an object module file, and generates the program source listing. After completing this phase **DEFT Pascal** has finished its execution which is marked by the return of the BASIC OK prompt.

Each **DEFT Pascal** Compiler prompt and its possible replies are described in the following sections.

2.1 SOURCE:

SOURCE requires the entry of the name of the source file which contains the Pascal language program that is to be compiled. The default file name extension is PAS. This means that if there is no extension specified with the entered file name, then the compiler adds the default extension of PAS to the file name before searching for that file.

2.2 OBJECT:

OBJECT requires the name of the object file that is to be created by the **DEFT Pascal** Compiler to hold the newly created program object module. This can be either on tape or disk or the name can be ommitted entirely if you do not wish to create an object file. The default extension is OBJ. If you do not specify an extension with the file name entered here, then the **DEFT Pascal** Compiler will add the default "OBJ" extension to your file name prior to actually creating that file.

2.3 LIST:

LIST requires the name of the source listing file which is to be created by **DEFT Pascal** in its second phase of operation. This can be tape, disk, screen or printer or the name can be ommitted entirely if you do not wish to create a list file. In this case only source

lines

with errors and the corresponding error messages will be output to the screen.

The default extension is LST. If you do not specify an extension with the file name entered here, then **DEFT Pascal** will add the default LST extension to your file name prior to actually creating that file.

2.4 DEBUG?:

DEBUG? asks you whether you wish to have debug information included in the resulting object file. If you intend to use **DEFT Debugger** to debug this program, then a *Y* response should be entered. A yes response to this question results in **DEFT Pascal** adding the debugger symbolic linkages to your program, therefore making the resulting object module larger than it otherwise would have been. If you don't want the debug information included, you can answer this prompt with either an "N", or "n". Anything other than "N" or "n" (for No) is taken to be "Y" (for Yes).

The debug information will make your program significantly bigger but will allow you to symbolically debug your resulting program if you answer the **DEFT Linker's** *debugger?* question yes. If you specify yes to the **DEFT Pascal** compiler's *debug?* question and No to the **DEFT Linker's** *debugger?* question, then the debug information will still be in the final binary image even though the debugger module is not present.

2.5 DIRECTIVE:

DIRECTIVE requires any **DEFT Pascal** Compiler directive that you would like to include before any source lines are read. The following section *Compiler Controls* describes all the possible compiler controls that you could enter here.

2.6 Compiler Execution

After you answer the *DIRECTIVE* prompt, the program will begin executing. The compiler requires that the file PASCALIB/EXT be present on disk drive 0 when the *SOURCE:* prompt is answered. When the compiler is finished executing, control will return to BASIC and you will get the OK prompt.

3 Source Listing

The following is a brief description of the **DEFT Pascal** Compiler's source listing.

1. *Header* - This is the first line at the top of the source listing followed by the page number for that page of the listing

2. *Title* - This is the second line from the top of the source listing. The contents of this line are dictated by the programmer with a *title* directive.

3. *Subtitle* - This is the third line from the top of the source listing. The contents of this line are dictated by the programmer with a *subtitle* directive.

4. *Nesting Levels* - The first column of numbers printed with each line is actually two separate nesting levels:

   - The first one is the *procedure* nesting level. This identifies what level of *procedure* the current line of code is known in.

   - The second number is the *begin* nesting level. This identifies how many *begins* have been encountered so far with no matching *ends*.

5. *Program Location Counter* - This second column is a hexadecimal representation of the program address at which that line's executable statement will begin. All other numbers printed on the listing are decimal.

6. *Symbol Table* - A list of all the symbols that were defined within a Pascal block is produced at the end of each block. This list contains a number of fields for each of these symbols. Following are all the column headings and a description of the information printed under each heading:

   - *SYMBOL* - This is the symbol name.

   - *CLASS* - This identifies what kind of Pascal language element this symbol represents.

   - *STRUCT* - For structured types and variables, this column identifies what their structure is *(array, record, set, pointer* or *file).*

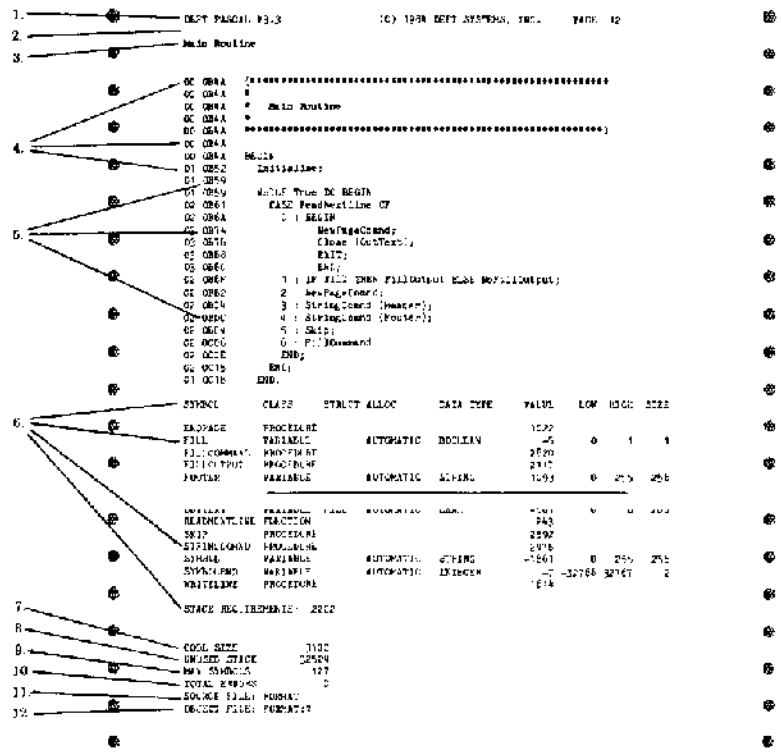
   - *ALLOC* - For variables, this column represents the allocation of that variable. Any *external* procedures or functions will have EXTERNAL printed here. Symbols which are fields within a *record* will have the name of the corresponding *record* printed here.

- *DATA TYPE* - For variables, types and constants, The Pascal *type* specified for the data element represented by this symbol is printed under this heading.
- *VALUE* - This identifies the value of the symbol. For *static* variables, procedures, functions, labels and string constants it is the relative offset from the beginning of the module. For automatic variables it is the offset within the stack frame. For non-string constants, it is the value of the constant.
- *LOW* - This heading identifies the lowest or smallest value to which the data in a type or variable may be set. For arrays, it is the lowest possible subscript.
- *HIGH* - This heading identifies the highest or largest value to which the data in a type or variable may be set. For arrays, it is the highest possible subscript.
- *SIZE* - For variables, types and constants, this is the number of bytes of memory represented by the Pascal *type*.
- *STACK REQUIREMENTS:* - This title precedes the estimation of the number of bytes of stack space required to activate this block.

7. *CODE SIZE* - This is the fifth from the last line printed on the source listing. Following is the number of bytes of memory that the program will require when it is loaded.
8. *UNUSED STACK* - The following number is the amount of stack space available but unused by the compiler itself. As you create more symbols and deeper levels of nesting in your program, this number will grow smaller. This stack space essentially represents the limits of the compiler for number of symbols and levels of nesting (of all kinds).
9. *MAX SYMBOLS* - The following number is the maximum number of symbols known at any point in the Pascal source program. Due to pre-defined symbols and the definitions in PascalIB/EXT, there will always be over 60 symbols defined in a program. Note that each symbol definition takes up about 30 bytes of compiler stack space.

10. *TOTAL ERRORS* - This is the number of compilation errors.
11. *SOURCE FILE* - Following this is the name of the source file containing the program source statements which generated this listing.
12. *OBJECT FILE* - Following this is the name of the file which contained the program object at the end of this compilation.

Sample Listing



1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.

```
DEFT PASCAL V3.3                (C) 1984 DEFT SYSTEMS, INC.     PAGE  12

Main Routine

00 084A    {**********************************************************
00 084A    *
00 084A    *   Main Routine
00 084A    *
00 084A    **********************************************************}
00 084A
00 084A    BEGIN
01 0852      Initialize;
01 0859
01 0859      WHILE True DO BEGIN
02 0861        CASE ReadNextLine OF
02 086A          0 : BEGIN
03 0874                NewPageComnd;
03 0876                Close (OutText);
03 0888                EXIT;
03 088C                END;
02 088F          1 : IF FILL THEN FillOutput ELSE NoFillOutput;
02 08B2          2 : NewPageComnd;
02 08C4          3 : StringComnd (Header);
02 08DC          4 : StringComnd (Footer);
02 08E4          5 : Skip;
02 08EC          6 : FillCommand
02 08EE          END;
02 0C15        END;
01 0C1B      END.

SYMBOL        CLASS      STRUCT ALLOC        DATA TYPE       VALUE    LOW  HIGH  SIZE

ENDPAGE       PROCEDURE                                       1022
FILL          VARIABLE          AUTOMATIC    BOOLEAN            -5      0     1     1
FILLCOMMAND   PROCEDURE                                       2520
FILLOUTPUT    PROCEDURE                                       2317
FOOTER        VARIABLE          AUTOMATIC    STRING           1093      0   255   256

[illegible]   VARIABLE   FILE   AUTOMATIC    [illegible]    [illegible]  0     0   [illegible]
READNEXTLINE  FUNCTION                                         943
SKIP          PROCEDURE                                       2592
STRINGCOMND   PROCEDURE                                       2976
SYMBOL        VARIABLE          AUTOMATIC    STRING          -1961      0   255   255
SYMBOLEND     VARIABLE          AUTOMATIC    INTEGER            -7 -32768 32767     2
WRITELINE     PROCEDURE                                       1614

STACK REQUIREMENTS:  2202

CODE SIZE            3102
UNUSED STACK        32524
MAX SYMBOLS           127
TOTAL ERRORS            0
SOURCE FILE: FORMAT
OBJECT FILE: FORMAT:?
```

4 Compiler Controls

Compiler controls are those instructions included in your source code or in the *DIRECTIVE:* prompt which direct the compiler's operation rather than the resulting program's operation. A compiler control is a source line with a percent sign (%) as the first character in the line. The control itself is a single character following the %. Any required parameters then follow the control character. For those controls not requiring parameters, additional controls may be included in consecutive columns. The % is not required in the DIRECTIVE prompt.

4.1 Listing Control

DEFT Pascal normally produces a source line listing file. The List (L) and Nolist (N) compiler controls allow you to control which portions of the source lines are included in this listing. These controls are additive; that is, if you include more than one list or nolist control in a row, it takes an equivalent number of the other to cancel its effects.

This additive nature gives you the ability to pre-cancel an imbedded nolist command with a preceding list command and vice-versa. This is very convenient when using copy files (see below). For example, **DEFT Pascal** copies by default the file PASCALIB/EXT which has a nolist control at the beginning of the file and a list command at the end. You can "unsuppress" its listing by including a list (L) control in response to the DIRECTIVE prompt in the compiler start-up screen.

4.2 Assembler Listing Control

DEFT Pascal is a true Pascal source to 6809 object code compiler. As such, it can produce a listing of the corresponding assembly language code that would be required to produce the same object. The default condition for the compiler is to not produce this assembly language listing. The compiler control used to turn on this listing is the plus sign (+). The compiler control used to turn it off is the minus sign (-).

4.3 Top of Page

The source listing produced by the **DEFT Pascal** Compiler normally prints 55 lines per page. However, you can force the compiler to start a new page at any point by including the eject (E) compiler control.

4.4 Title and Subtitle

Included at the top of each page produced by the compiler is the compiler's name, copyright notice and page number. In addition, on the following two lines you can specify a title (T) and subtitle (S). The remainder of the line on which the control is specified becomes the title or subtitle. Following are examples:

```
%T This is a Title String
%S This is a Subtitle String
```

Note that the presence of either control implies an eject (E). Blanks immediately following the control up to the first non-blank are suppressed in the actual title or subtitle.

In addition to printing at the top of each page, the title string is also included as a comment statement in the resulting object file. It will then also appear in **DEFT Linker's** listing file.

4.5 Copy

Sometimes it is desirable not to include your entire program in a single source file even though you wish to compile it as a single unit. This may be due to limitations of the editor or to allow common definitions for *interface* modules (see *Separate Compilation).*

The copy (C) compiler control allows you to tell the compiler where additional source lines should be taken from. The remainder of the control line is considered to be the file name of a Pascal source file. The compiler will read all the lines in the specified source file before reading the next line in the current source file. Example:

```
%C GRAPHINT:1
```

This line causes the file GRAPHINT/PAS on disk drive 1 to be completely read before reading the next line in the current file. Note that copy controls can be nested. That is, a file that is copied may itself contain a copy control. This nesting is only allowed to two levels.

Compile