

Free Pascal :  
User's Guide

---

User's Guide for Free Pascal, Version 2.4.2rc1  
Document version 2.4  
July 2012

Michaël Van Canneyt  
Florian Klämpfl

---











|          |   |            |
|----------|---|------------|
| C.7      | Errors of assembling/linking stage . . . . .    | 163        |
| C.8      | Executable information messages. . . . .        | 165        |
| C.9      | Linker messages . . . . .                       | 165        |
| C.10     | Unit loading messages. . . . .                  | 165        |
| C.11     | Command line handling errors . . . . .          | 168        |
| C.12     | Whole program optimization messages . . . . .   | 170        |
| C.13     | Assembler reader errors. . . . .                | 172        |
| C.13.1   | General assembler errors . . . . .              | 172        |
| C.13.2   | I386 specific errors . . . . .                  | 175        |
| C.13.3   | m68k specific errors. . . . .                   | 176        |
| <b>D</b> | <b>Run-time errors</b>                          | <b>177</b> |
| <b>E</b> | <b>A sample <code>gdb.ini</code> file</b>       | <b>181</b> |
| <b>F</b> | <b>Options and settings</b>                     | <b>182</b> |
| <b>G</b> | <b>Getting the latest sources or installers</b> | <b>184</b> |
| G.1      | Download via Subversion . . . . .               | 184        |
| G.2      | Downloading a source zip . . . . .              | 185        |
| G.3      | Downloading a snapshot . . . . .                | 185        |

# Chapter 1

## Introduction

### 1.1 About this document

This is the user's guide for Free Pascal. It describes the installation and use of the Free Pascal compiler on the different supported platforms. It does not attempt to give an exhaustive list of all supported commands, nor a definition of the Pascal language. Look at the [Reference Guide](#) for these things. For a description of the possibilities and the inner workings of the compiler, see the [Programmer's Guide](#). In the appendices of this document you will find lists of reserved words and compiler error messages (with descriptions).

This document describes the compiler as it is/functions at the time of writing. First consult the README and FAQ files, distributed with the compiler. The README and FAQ files are, in case of conflict with this manual, authoritative.

### 1.2 About the compiler

Free Pascal is a 32- and 64-bit Pascal compiler. The current version (2.2) can compile code for the following processors:

- Intel i386 and higher (i486, Pentium family and higher)
- AMD64/x86\_64
- PowerPC
- PowerPC64
- SPARC
- ARM
- The m68K processor is supported by an older version.

The compiler and Run-Time Library are available for the following operating systems:

- DOS
- LINUX
- AMIGA (version 0.99.5 only)



- <http://community.freepascal.org:10000/> is a forum site where questions can be posted.

Other than that, some mirrors exist.

Finally, if you think something should be added to this manual (entirely possible), please do not hesitate and contact me at [michael@freepascal.org](mailto:michael@freepascal.org).

Let's get on with something useful.

## Chapter 2

# Installing the compiler

### 2.1 Before Installation : Requirements

#### 2.1.1 Hardware requirements

The compiler needs at least one of the following processors:

1. An Intel 80386 or higher processor. A coprocessor is not required, although it will slow down your program's performance if you do floating point calculations without a coprocessor, since emulation will be used.
2. An AMD64 or EMT64 processor.
3. A PowerPC processor.
4. A SPARC processor
5. An ARM processor.
6. Older FPC versions exist for the motorola 68000 processor, but these are no longer maintained.

Memory and disk requirements:

1. 8 Megabytes of free memory. This is sufficient to allow compilation of small programs.
2. Large programs (such as the compiler itself) will require at least 64 MB. of memory, but 128MB is recommended. (Note that the compiled programs themselves do not need so much memory.)
3. At least 80 MB free disk space. When the sources are installed, another 270 MB are needed.

#### 2.1.2 Software requirements

##### Under DOS

The DOS distribution contains all the files you need to run the compiler and compile Pascal programs.

##### Under UNIX

Under UNIX systems (such as LINUX) you need to have the following programs installed :























You can use the `-xs` switch to let the compiler do this stripping automatically at program compile time. (The switch has no effect when compiling units.)

Another technique to reduce the size of a program is to use smartlinking. Normally, units (including the system unit) are linked in as a whole. It is however possible to compile units such that they can be smartlinked. This means that only the functions and procedures that are actually used are linked in your program, leaving out any unnecessary code. The compiler will turn on smartlinking with the `-xx` (see page [31](#)) switch. This technique is described in full in the programmers guide.





























This is equivalent to

```
-Fu/usr/lib/fpc/2.2.2/rtl/linux
```

if the compiler version is 2.2.2 and the target OS is LINUX.

These replacements are valid on the command line and also in the configuration file.

On the linux command line, you must be careful to escape the \$ since otherwise the shell will attempt to expand the variable for you, which may have undesired effects.





## 6.2 Navigating in the IDE

The IDE can be navigated both with the keyboard and with a mouse, if the system is equipped with a mouse.

### 6.2.1 Using the keyboard

All functionality of the IDE is available through use of the keyboard.

- It is used for typing and navigating through the sources.
- Editing commands such as copying and pasting text.
- Moving and resizing windows.
- It can be used to access the menu, by pressing ALT and the appropriate highlighted menu letter, or by pressing F10 and navigating through the menu with the arrow keys. More information on the menu can be found in section 6.4, page 43.
- Many commands in the IDE are bound to shortcuts, i.e. typing a special combination of keys will execute a command immediately.

**Remark:**

- When working in a LINUX X-Term or through a telnet session, the key combination with ALT may not be available. To remedy this, the CTRL-Z combination can be typed first. This means that e.g. ALT-X can be replaced by CTRL-Z X.
- Alternatively, you can try the key combination ESC-X for ALT-X when working on LINUX.
- A complete reference of all keyboard shortcuts can be found in section 6.14, page 88.

### 6.2.2 Using the mouse

If the system is equipped with a mouse, it can be used to work with the IDE. The left button is used to select menu items, press buttons, select text blocks etc.

The right mouse button is used to access the local menu, if available. Holding down the CTRL or ALT key and clicking the right button will execute user defined functions. See section 6.12.4, page 85.

**Remark:**

1. Occasionally, the manual uses the term "drag the mouse". This means that the mouse is moved while the left mouse button is being pressed.
2. The action of mouse buttons may be reversed, i.e. the actions of the left mouse button can be assigned to the right mouse button and vice versa<sup>2</sup>. Throughout the manual, it is assumed that the actions of the mouse buttons are not reversed.
3. The mouse is not always available, even if a mouse is installed:
  - The IDE is running under LINUX through a telnet connection from a WINDOWS machine.
  - The IDE is running under LINUX in an X-term under X-windows. In this case it depends on the terminal program: under Konsole (the KDE terminal) it works.

---

<sup>2</sup>See section 6.12.4, page 85 for more information on how to reverse the actions of the mouse buttons.

4. On Windows, the console has an option 'Quick edit', allowing text to be copied to the clipboard by selecting text in the console window. If this mode is enabled, the mouse will not work. The 'Quick edit' option should be disabled in the console window's properties in order for the IDE to receive mouse events.

### 6.2.3 Navigating in dialogs

Dialogs usually have a lot of elements in them such as buttons, edit fields, memo fields, list boxes and so on. To activate one of these fields, choose one of the following methods:

1. Click on the element with the mouse.
2. Press the TAB key till the focus reaches the element.
3. Press the highlighted letter in the element's label. If the focus is currently on an element that allows editing, then ALT should be pressed simultaneously with the highlighted letter. For a button, the action associated with the button will then be executed.

Inside edit fields, list boxes and memos, navigation is carried out with the usual arrow key commands.

## 6.3 Windows

Nowadays, working with windowed applications should be no problem for most WINDOWS and LINUX users. Nevertheless, the following section describes how the windows work in order to derive the most benefit from the Free Pascal IDE.

### 6.3.1 Window basics

A common IDE window is displayed in figure (6.2).

Figure 6.2: A common IDE window



The window is surrounded by a so-called *frame*, the white double line around the window.

At the top of the window 4 things are displayed:

















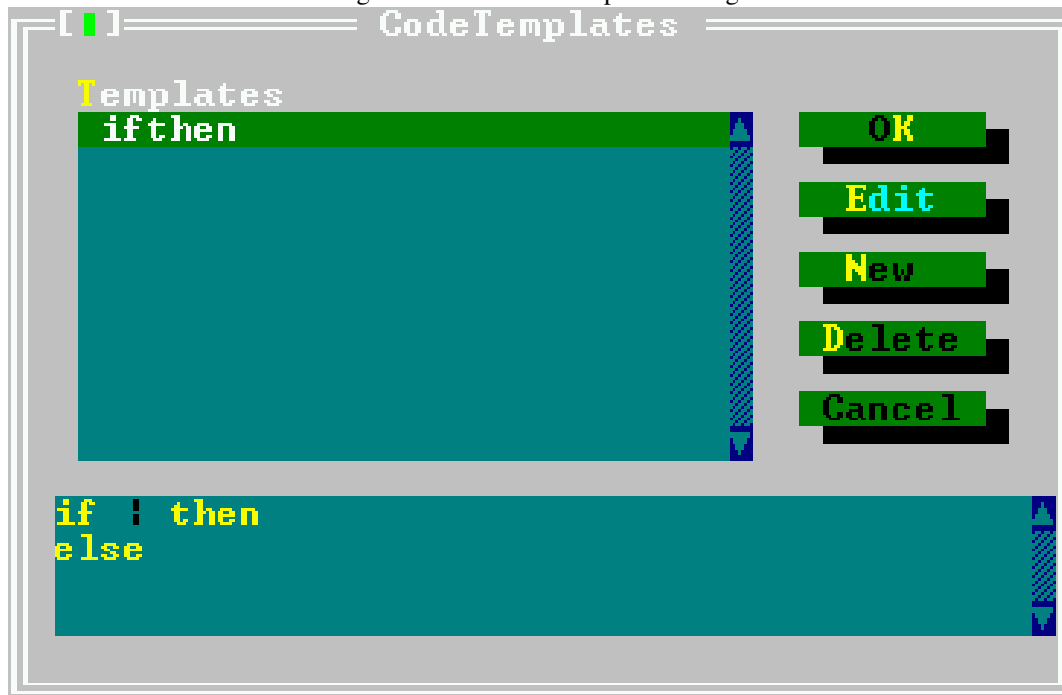








Figure 6.6: The code templates dialog.



All templates are saved and are available the next time the IDE is started.

**Remark:** Duplicates are not allowed. If an attempt is made to add a duplicate name to the list, an error will occur.

## 6.6 Searching and replacing

The IDE allows you to search for text in the active editor window. To search for text, one of the following can be done:

1. Select "**Search|Find**" in the menu.
2. Press CTRL-Q F.

After that, the dialog shown in figure (6.7) will pop up, and the following options can be entered:

**Text to find** The text to be searched for. If a block was active when the dialog was started, the first line of this block is proposed.

**Case sensitive** When checked, the search is case sensitive.

**Whole words only** When checked, the search text must appear in the text as a complete word.

**Direction** The direction in which the search must be conducted, starting from the specified origin.

**Scope** Specifies if the search should be on the whole file, or just the selected text.

**Origin** Specifies if the search should start from the cursor position or the start of the scope.









































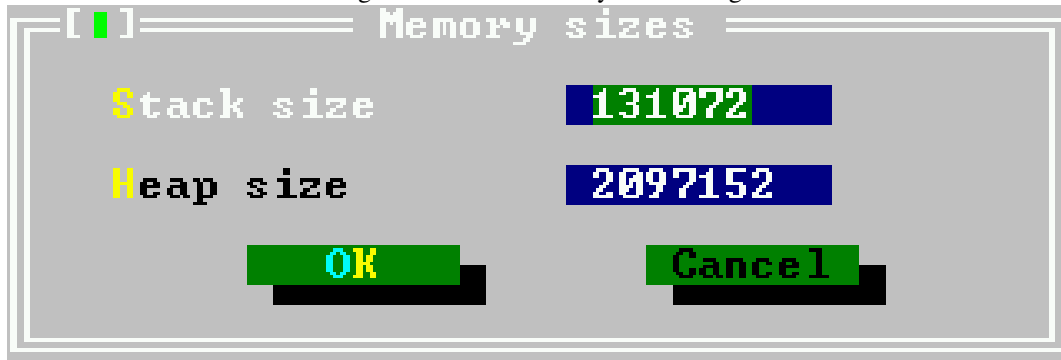








Figure 6.30: The memory sizes dialog



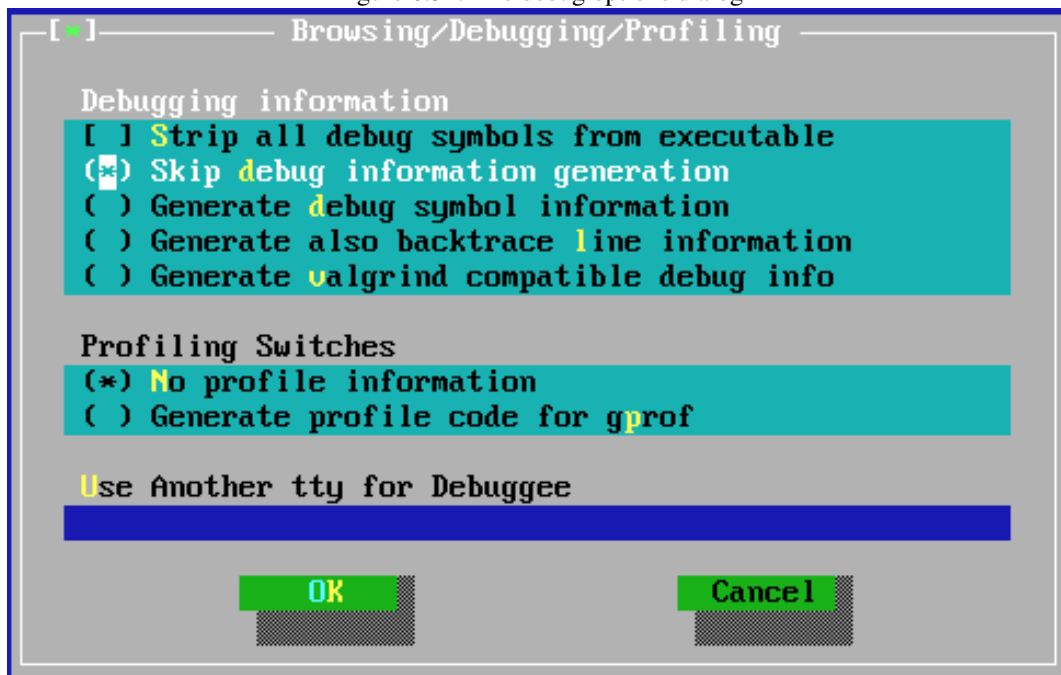
**Stack size** Sets the size of the stack in bytes (option `-Cs` on the command line). This size may be ignored on some systems.

**Heap size** Sets the size of the heap in bytes; (option `-Ch` on the command line). Note that the heap grows dynamically as much as the OS allows.

### 6.11.7 Debug options

In the debug options dialog (reachable via "**Options|Debugger**"), some options for inclusion of debug information in the binary can be set; it is also possible to add additional compiler options in this dialog. The debug options dialog is shown in figure (6.31).

Figure 6.31: The debug options dialog



The following options can be set:





















Table 6.8: Block commands

| Command                               | Shortcut key | Alternative |
|---------------------------------------|--------------|-------------|
| Goto Beginning of selected text       | CTRL-Q-B     |             |
| Goto end of selected text             | CTRL-Q-K     |             |
| Select current line                   | CTRL-K-L     |             |
| Print selected text                   | CTRL-K-P     |             |
| Select current word                   | CTRL-K-T     |             |
| Delete selected text                  | CTRL-DEL     | CTRL-K-Y    |
| Copy selected text to cursor position | CTRL-K-C     |             |
| Move selected text to cursor position | CTRL-K-V     |             |
| Copy selected text to clipboard       | CTRL-INS     |             |
| Move selected text to the clipboard   | SHIFT-DEL    |             |
| Indent block one column               | CTRL-K-I     |             |
| Unindent block one column             | CTRL-K-U     |             |
| Insert text from clipboard            | SHIFT-INSERT |             |
| Insert file                           | CTRL-K-R     |             |
| Write selected text to file           | CTRL-K-W     |             |
| Uppercase current block               | CTRL-K-N     |             |
| Lowercase current block               | CTRL-K-O     |             |
| Uppercase word                        | CTRL-K-E     |             |
| Lowercase word                        | CTRL-K-F     |             |

Table 6.9: Change selection

| Command   | Shortcut key           | Alternative       |
|---|------------------------|-------------------|
| Mark beginning of selected text                     | CTRL-K-B               |                   |
| Mark end of selected text                           | CTRL-K-K               |                   |
| Remove selection                                    | CTRL-K-Y               |                   |
| Extend selection one char to the left               | SHIFT-ARROW LEFT       |                   |
| Extend selection one char to the right              | SHIFT-ARROW RIGHT      |                   |
| Extend selection to the beginning of the line       | SHIFT-POS1             |                   |
| Extend selection to the end of the line             | SHIFT-END              |                   |
| Extend selection to the same column in the last row | SHIFT-ARROW UP         |                   |
| Extend selection to the same column in the next row | SHIFT-ARROW DOWN       |                   |
| Extend selection to the end of the line             | SHIFT-END              |                   |
| Extend selection one word to the left               | CTRL-SHIFT-ARROW LEFT  |                   |
| Extend selection one word to the right              | CTRL-SHIFT-ARROW RIGHT |                   |
| Extend selection one page up                        | SHIFT-PAGEUP           |                   |
| Extend selection one page down                      | SHIFT-PAGEDOWN         |                   |
| Extend selection to the beginning of the file       | CTRL-SHIFT-POS1        | CTRL-SHIFT-PAGEUP |
| Extend selection to the end of the file             | CTRL-SHIFT-END         | CTRL-SHIFT-PAGEUP |

Table 6.10: Misc. commands

| Command            | Shortcut key                   | Alternative |
|--------------------|--------------------------------|-------------|
| Save file          | F2                             | CTRL-K-S    |
| Open file          | F3                             |             |
| Search             | CTRL-Q-F                       |             |
| Search again       | CTRL-L                         |             |
| Search and replace | CTRL-Q-A                       |             |
| Set mark           | CTRL-K-N (where n can be 0..9) |             |
| Goto mark          | CTRL-Q-N (where n can be 0..9) |             |
| Undo               | ALT-BACKSPACE                  |             |







```
function a : longint;

begin
  a:=12;
  while a>4 do
    begin
      {...}
    end;
  end;
end;
```

The example above would work with TP, but the compiler would assume that the `a>4` is a recursive call. If a recursive call is actually what is desired, you must append `()` after the function name:

```
function a : longint;

begin
  a:=12;
  { this is the recursive call }
  if a()>4 then
    begin
      {...}
    end;
  end;
end;
```

3. In Free Pascal, there is partial support of Delphi constructs. (See the [Programmer's Guide](#) for more information on this).
4. The Free Pascal `exit` call accepts a return value for functions.

```
function a : longint;

begin
  a:=12;
  if a>4 then
    begin
      exit(a*67); {function result upon exit is a*67 }
    end;
  end;
end;
```

5. Free Pascal supports function overloading. That is, you can define many functions with the same name, but with different arguments. For example:

```
procedure DoSomething (a : longint);
begin
  {...}
end;

procedure DoSomething (a : real);
begin
  {...}
end;
```

You can then call procedure `DoSomething` with an argument of type `Longint` or `Real`. This feature has the consequence that a previously declared function must always be defined with the header completely the same:

```
procedure x (v : longint); forward;

{...}

procedure x; { This will overload the previously declared x }
begin
{...}
end;
```

This construction will generate a compiler error, because the compiler didn't find a definition of `procedure x (v : longint);`. Instead you should define your procedure `x` as:

```
procedure x (v : longint);
{ This correctly defines the previously declared x }
begin
{...}
end;
```

The command line option `-So` (see page 33) disables overloading. When you use it, the above will compile, as in Turbo Pascal.

6. Operator overloading. Free Pascal allows operator overloading, e.g. you can define the '+' operator for matrices.
7. On FAT16 and FAT32 systems, long file names are supported.

### 7.2.3 Turbo Pascal compatibility mode

When you compile a program with the `-Mtp` switch, the compiler will attempt to mimic the Turbo Pascal compiler in the following ways:

- Assigning a procedural variable doesn't require an `@` operator. One of the differences between Turbo Pascal and Free Pascal is that the latter requires you to specify an address operator when assigning a value to a procedural variable. In Turbo Pascal compatibility mode, this is not required.
- Procedure overloading is disabled. If procedure overloading is disabled, the function header doesn't need to repeat the function header.
- Forward defined procedures don't need the full parameter list when they are defined. Due to the procedure overloading feature of Free Pascal, you must always specify the parameter list of a function when you define it, even when it was declared earlier with `Forward`. In Turbo Pascal compatibility mode, there is no function overloading; hence you can omit the parameter list:

```
Procedure a (L : Longint); Forward;

...

Procedure a ; { No need to repeat the (L : Longint) }

begin
  ...
end;
```

- Recursive function calls are handled differently. Consider the following example:

```
Function expr : Longint;  
  
begin  
    ...  
    Expr:=L;  
    Writeln (Expr);  
    ...  
end;
```

In Turbo Pascal compatibility mode, the function will be called recursively when the `writeln` statement is processed. In Free Pascal, the function result will be printed. In order to call the function recursively under Free Pascal, you need to implement it as follows :

```
Function expr : Longint;  
  
begin  
    ...  
    Expr:=L;  
    Writeln (Expr());  
    ...  
end;
```

- Any text after the final `End.` statement is ignored. Normally, this text is processed too.
- You cannot assign procedural variables to untyped pointers; so the following is invalid:

```
a: Procedure;  
b: Pointer;  
begin  
    b := a; // Error will be generated.
```

- The `@` operator is typed when applied on procedures.
- You cannot nest comments.

**Remark:** The `MemAvail` and `MaxAvail` functions are no longer available in Free Pascal as of version 2.0. The reason for this incompatibility follows:

On modern operating systems,<sup>1</sup> the idea of "Available Free Memory" is not valid for an application. The reasons are:

1. One processor cycle after an application asked the OS how much memory is free, another application may have allocated everything.
2. It is not clear what "free memory" means: does it include swap memory, does it include disk cache memory (the disk cache can grow and shrink on modern OS'es), does it include memory allocated to other applications but which can be swapped out, etc.

Therefore, programs using `MemAvail` and `MaxAvail` functions should be rewritten so they no longer use these functions, because it does not make sense any more on modern OS'es. There are 3 possibilities:

---

<sup>1</sup>The DOS extender GO32V2 falls under this definition of "modern" because it can use paged memory and run in multi-tasking environments.

1. Use exceptions to catch out-of-memory errors.
2. Set the global variable "ReturnNilIfGrowHeapFails" to `True` and check after each allocation whether the pointer is different from `Nil`.
3. Don't care and declare a dummy function called `MaxAvail` which always returns `High (LongInt)` (or some other constant).

### 7.2.4 A note on long file names under DOS

Under WINDOWS 95 and higher, long filenames are supported. Compiling for the WINDOWS target ensures that long filenames are supported in all functions that do file or disk access in any way.

Moreover, Free Pascal supports the use of long filenames in the system unit and the `Dos` unit also for go32v2 executables. The system unit contains the boolean variable `LFNSupport`. If it is set to `True` then all system unit functions and `Dos` unit functions will use long file names if they are available. This should be so on WINDOWS 95 and 98, but not on WINDOWS NT or WINDOWS 2000. The system unit will check this by calling DOS function 71A0h and checking whether long filenames are supported on the C: drive.

It is possible to disable the long filename support by setting the `LFNSupport` variable to `False`; but in general it is recommended to compile programs that need long filenames as native WINDOWS applications.

## 7.3 Porting Delphi code

Porting Delphi code should be quite painless. The `Delphi` mode of the compiler tries to mimic Delphi as closely as possible. This mode can be enabled using the `-Mdelphi` command line switch, or by inserting the following code in the sources before the `unit` or `program` clause:

```
{ $IFDEF FPC }
{ $MODE DELPHI }
{ $ENDIF FPC }
```

This ensures that the code will still compile with both Delphi and FPC.

Nevertheless, there are some things that will not work. Delphi compatibility is relatively complete up to Delphi 7. New constructs in higher versions of Delphi (notably, the versions that work with .NET) are not supported.

### 7.3.1 Missing language constructs

At the level of language compatibility, FPC is very compatible with Delphi: it can compile most of FreeCLX, the free Widget library that was shipped with Delphi 6, Delphi 7 and Kylix.

Currently, the only missing language constructs are:

1. Dynamic methods are actually the same as `virtual`.
2. `Const` for a parameter to a procedure does not necessarily mean that the variable or value is passed by reference.
3. Packages are not supported.



- Likewise, by default enumeration types are of different size than in Delphi. Here again, the size can be specified using directives or command line switches.
- In general, one should not make assumptions about the internal structure of complex types such as records, objects, classes and their associated structure. For example, the VMT table layout is different, the alignment of fields in a record may be different, etc.
- The same is true for basic types: on other processors the high and low bytes of a word or integer may not be at the same location as on an Intel processor (the endianness is different).
- Names of local variables and method arguments are not allowed to match the name of a property or field of the class: this is bad practise, as there can be confusion as to which of the two is meant.

### 7.3.3 Delphi compatibility mode

Switching on Delphi compatibility mode has the following effect:

1. Support for Classes, exceptions and threadvars is enabled.
2. The `objpas` is loaded as the first unit. This unit redefines some basic types: `Integer` is 32-bit for instance.
3. The address operator (`@`) is no longer needed to set event handlers (i.e. assign to procedural variables or properties).
4. Names of local variables and method parameters in classes can match the name of properties or field of the class.
5. The `String` keyword implies `AnsiString` by default.
6. Operator overloading is switched off.

### 7.3.4 Best practices for porting

When encountering differences in Delphi/FPC calls, the best thing to do is not to insert `IFDEF` statements whenever a difference is encountered, but to create a separate unit which is only used when compiling with FPC. The missing/incompatible calls can then be implemented in that unit. This will keep the code more readable and easier to maintain.

If a language construct difference is found, then the Free Pascal team should be contacted and a bug should be reported.

## 7.4 Writing portable code

Free Pascal is designed to be cross-platform. This means that the basic RTL units are usable on all platforms, and the compiler behaves the same on all platforms (as far as possible). The Object Pascal language is the same on all platforms. Nevertheless, FPC comes with a lot of units that are not portable, but provide access to all possibilities that a platform provides.

The following are some guidelines to consider when writing portable code:

- Avoid system-specific units. The system unit, the objects and classes units and the `SysUtils` unit are guaranteed to work on all systems. So is the `DOS` unit, but that is deprecated.

- Avoid direct hardware access. Limited, console-like hardware access is available for most platforms in the Video, Mouse and Keyboard units.
- Do not use hard-coded filename conventions. See below for more information on this.
- Make no assumptions on the internal representation of types. Various processors store information in different ways ('endianness').
- If system-specific functionality is needed, it is best to separate this out in a single unit. Porting efforts will then be limited to re-implementing this unit for the new platform.
- Don't use assembler, unless you have to. Assembler is processor specific. Some instructions will not work even on the same processor family.
- Do not assume that pointers and integers have the same size. They do on an Intel 32-bit processor, but not necessarily on other processors. The `PtrInt` type is an alias for the integer type that has the same size as a pointer. `SizeInt` is used for all size-related issues.

The system unit contains some constants which describe file access on a system:

**AllFilesMask** a file mask that will return all files in a directory. This is `*` on Unix-like platforms, and `*.*` on dos and windows like platforms.

**LineEnding** A character or string which describes the end-of-line marker used on the current platform. Commonly, this is one of `#10`, `#13#10` or `#13`.

**LFNSupport** A boolean that indicates whether the system supports long filenames (i.e. is not limited to MS-DOS 8.3 filenames).

**DirectorySeparator** The character which acts as a separator between directory parts of a path.

**DriveSeparator** For systems that support drive letters, this is the character that is used to separate the drive indication from the path.

**PathSeparator** The character used to separate items in a list (notably, a PATH).

**maxExitCode** The maximum value for a process exitcode.

**MaxPathLen** The maximum length of a filename, including a path.

**FileNameCaseSensitive** A boolean that indicates whether filenames are handled case sensitively.

**UnusedHandle** A value used to indicate an unused/invalid file handle.

**StdInputHandle** The value of the standard input file handle. This is not always 0 (zero), as is commonly the case on Unices.

**StdOutputHandle** The value of the standard output file handle. This is not always 1, as is commonly the case on Unices.

**StdErrorHandle** The value of the standard diagnostics output file handle. This is not always 2, as is commonly the case on Unices.

**CtrlZMarksEOF** A boolean that indicates whether the `#26` character marks the end of a file (an old MS-DOS convention).

To ease writing portable filesystem code, the Free Pascal file routines in the system unit and `sysutils` unit treat the common directory separator characters (`/` and `\`) as equivalent. That means that if you use `/` on a WINDOWS system, it will be transformed to a backslash, and vice versa.

This feature is controlled by 2 (pre-initialized) variables in the system unit:

**AllowDirectorySeparators** A set of characters which, when used in filenames, are treated as directory separators. They are transformed to the `DirectorySeparator` character.

**AllowDriveSeparators** A set of characters which, when used in filenames, are treated as drive separator characters. They are transformed to the `DriveSeparator` character.

## Chapter 8

# Utilities that come with Free Pascal

Besides the compiler and the runtime Library, Free Pascal comes with some utility programs and units. Here we list these programs and units.

### 8.1 Demo programs and examples

A suite of demonstration programs comes included with the Free Pascal distribution. These programs have no other purpose than to demonstrate the capabilities of Free Pascal. They are located in the `demo` directory of the sources.

All example programs mentioned in the documentation are available. Check out the directories that are beneath the same directory as the `demo` directory. The names of these directories end on `ex`. There you will find all example sources.

### 8.2 `fpcmake`

`fpcmake` is the Free Pascal makefile constructor program.

It reads a `Makefile.fpc` configuration file and converts it to a `Makefile` suitable for reading by GNU `make` to compile your projects. It is similar in functionality to GNU `autoconf` or `lmake` for making X projects.

`fpcmake` accepts filenames of makefile description files as its command line arguments. For each of these files it will create a `Makefile` in the same directory where the file is located, overwriting any other existing file.

If no options are given, it just attempts to read the file `Makefile.fpc` in the current directory and tries to construct a makefile from it. Any previously existing `Makefile` will be erased.

The format of the `fpcmake` configuration file is described in great detail in the appendices of the [Programmer's Guide](#).

### 8.3 `fpdoc` - Pascal Unit documenter

`fpdoc` is a program which generates fully cross-referenced documentation for a unit. It generates documentation for each identifier found in the unit's interface section. The generated documentation can be in many formats, for instance HTML, RTF, Text, man page and LaTeX. Unlike other documentation tools, the documentation can be in a separate file (in XML format), so the sources aren't

cluttered with documentation. Its companion program `makeskel` creates an empty XML file with entries for all identifiers, or it can update an existing XML file, adding entries for new identifiers.

`fpdoc` and `makeskel` are described in the [FPDoc Reference Guide](#).

## 8.4 h2pas - C header to Pascal Unit converter

`h2pas` attempts to convert a C header file to a Pascal unit. it can handle most C constructs that one finds in a C header file, and attempts to translate them to their Pascal counterparts.

See below (constructs) for a full description of what the translator can handle. The unit with Pascal declarations can then be used to access code written in C.

The output of the `h2pas` program is written to a file with the same name as the C header file that was used as input, but with the extension `.pp`. The output file that `h2pas` creates can be customized in a number of ways by means of many options.

### 8.4.1 Options

The output of `h2pas` can be controlled with the following options:

- d** Use `external`; for all procedure and function declarations.
- D** Use `external libname name 'func_name'` for function and procedure declarations.
- e** Emit a series of constants instead of an enumeration type for the C `enum` construct.
- i** Create an include file instead of a unit (omits the unit header).
- l libname** specify the library name for external function declarations.
- o outfile** Specify the output file name. Default is the input file name with the extension replaced by `.pp`.
- p** Use the letter `P` in front of pointer type parameters instead of `*`.
- s** Strip comments from the input file. By default comments are converted to comments, but they may be displaced, since a comment is handled by the scanner.
- t** Prepend typedef type names with the letter `T` (used to follow Borland's convention that all types should be defined with `T`).
- v** Replace pointer parameters with call by reference parameters. Use with care because some calls can expect a `Nil` pointer.
- w** Header file is a win32 header file (adds support for some special macros).
- x** Handle `SYS_TRAP` of the PalmOS header files.

### 8.4.2 Constructs

The following C declarations and statements are recognized:

**defines** Defines are changed into Pascal constants if they are simple defines. Macros are changed - wherever possible - to functions; however the arguments are all integers, so these must be changed manually. Simple expressions in define statements are recognized, as are most arithmetic operators: addition, subtraction, multiplication, division, logical operators, comparison operators, shift operators. The C construct `( A ? B : C )` is also recognized and translated to a Pascal construct with an IF statement. (This is buggy, however).

**preprocessor statements** The conditional preprocessing commands are recognized and translated into equivalent Pascal compiler directives. The special

```
#ifdef __cplusplus
```

is also recognized and removed.

**typedef** A typedef statement is changed into a Pascal type statement. The following basic types are recognized:

- `char` changed to `char`.
- `float` changed to `real` (=double in Free Pascal).
- `int` changed to `longint`.
- `long` changed to `longint`.
- `long int` changed to `longint`.
- `short` changed to `integer`.
- `unsigned` changed to `cardinal`.
- `unsigned char` changed to `byte`.
- `unsigned int` changed to `cardinal`.
- `unsigned long int` changed to `cardinal`.
- `unsigned short` changed to `word`.
- `void` ignored.

These types are also changed if they appear in the arguments of a function or procedure.

**functions and procedures** Functions and procedures are translated as well. Pointer types may be changed to call by reference arguments (using the `var` argument) by using the `-p` command line argument. Functions that have a variable number of arguments are changed to a function with a `cvar` modifier. (This used to be the array of `const` argument.)

**specifiers** The `extern` specifier is recognized; however it is ignored. The `packed` specifier is also recognised and changed with the `PACKRECORDS` directive. The `const` specifier is also recognized, but is ignored.

**modifiers** If the `-w` option is specified, then the following modifiers are recognized:

```
STDCALL
CDECL
CALLBACK
PASCAL
WINAPI
APIENTRY
WINGDIAPI
```

as defined in the win32 headers. If additionally the `-x` option is specified then the

```
SYS_TRAP
```

specifier is also recognized.

**enums** Enum constructs are changed into enumeration types. Bear in mind that, in C, enumeration types can have values assigned to them. Free Pascal also allows this to a certain degree. If you know that values are assigned to enums, it is best to use the `-e` option to change the enumerations to a series of integer constants.

**unions** Unions are changed to variant records.

**structs** Structs are changed to Pascal records, with C packing.

















## Chapter 9

# Units that come with Free Pascal

Here we list the units that come with the Free Pascal distribution. Since there is a difference in the supplied units per operating system, we first describe the generic ones, then describe those which are operating system specific.

### 9.1 Standard units

The following units are standard and are meant to be ported to all platforms supported by Free Pascal. A brief description of each unit is also given.

**charset** A unit to provide mapping of character sets.

**cmem** Using this unit replaces the Free Pascal memory manager with the memory manager of the C library.

**crt** This unit is similar to the unit of the same name of Turbo Pascal. It implements writing to the console in color, moving the text cursor around and reading from the keyboard.

**dos** This unit provides basic routines for accessing the operating system. This includes file searching, environment variables access, getting the operating system version, getting and setting the system time. It is to note that some of these routines are duplicated in functionality in the `sysutils` unit.

**dynlibs** Provides cross-platform access to loading dynamical libraries.

**getopts** This unit gives you the GNU `getopts` command line arguments handling mechanism. It also supports long options.

**graph** *This unit is deprecated.* This unit provides basic graphics handling, with routines to draw lines on the screen, display text etc. It provides the same functions as the Turbo Pascal unit.

**heaptrc** a unit which debugs the heap usage. When the program exits, it outputs a summary of the used memory, and dumps a summary of unreleased memory blocks (if any).

**keyboard** provides basic keyboard handling routines in a platform independent way, and supports writing custom drivers.

**macpas** This unit implements several functions available only in MACPAS mode. This unit should not be included; it's automatically included when the MACPAS mode is used.

**math** This unit contains common mathematical routines (trigonometric functions, logarithms, etc.) as well as more complex ones (summations of arrays, normalization functions, etc.).

**matrix** A unit providing matrix manipulation routines.

**mmx** This unit provides support for `mmx` extensions in your code.

**mouse** Provides basic mouse handling routines in a platform independent way, and supports writing custom drivers.

**objects** This unit provides the base object for standard Turbo Pascal objects. It also implements File and Memory stream objects, as well as sorted and non-sorted collections, and string streams.

**objpas** Is used for Delphi compatibility. You should never load this unit explicitly; it is automatically loaded if you request Delphi mode.

**printer** This unit provides all you need for rudimentary access to the printer using standard I/O routines.

**sockets** This gives the programmer access to sockets and TCP/IP programming.

**strings** This unit provides basic string handling routines for the `pchar` type, comparable to similar routines in standard C libraries.

**system** This unit is available for all supported platforms. It includes among others, basic file I/O routines, memory management routines, all compiler helper routines, and directory services routines.

**strutils** Offers a lot of extended string handling routines.

**dateutils** Offers a lot of extended date/time handling routines for almost any date and time math.

**sysutils** Is an alternative implementation of the `sysutils` unit of Delphi. It includes file I/O access routines which takes care of file locking, date and string handling routines, file search, date and string conversion routines.

**typinfo** Provides functions to access runtime Type Information, just like Delphi.

**variants** Provides basic variant handling.

**video** Provides basic screen handling in a platform independent way, and supports writing custom drivers.

## 9.2 Under DOS

**emu387** This unit provides support for the coprocessor emulator.

**go32** This unit provides access to capabilities of the GO32 DOS extender.

**ports** This implements the various `port []` constructs for low-level I/O.



## 9.5 Under OS/2

**doscalls** Interface to doscalls.dll.

**dive** Interface to dive.dll

**emx** Provides access to the EMX extender.

**pm\*** Interface units for the Presentation Manager (PM) functions (GUI).

**viocalls** Interface to viocalls.dll screen handling library.

**moucalls** Interface to moucalls.dll mouse handling library.

**kbdcalls** Interface to kbdcalls.dll keyboard handling library.

**moncalls** Interface to moncalls.dll monitoring handling library.

**winsock** Provides access to the (emulated) WINDOWS sockets API Winsock.

**ports** This implements the various `port []` constructs for low-level I/O.

## 9.6 Unit availability

Standard unit availability for each of the supported platforms is given in the FAQ / Knowledge base.

## Chapter 10

# Debugging your programs

Free Pascal supports debug information for the GNU debugger `gdb`, or its derivatives `Insight` on `win32` or `ddd` on `LINUX`. It can write 2 kinds of debug information:

**stabs** The old debug information format.

**dwarf** The new debug information format.

Both are understood by GDB.

This chapter briefly describes how to use this feature. It doesn't attempt to describe completely the GNU debugger, however. For more information on the workings of the GNU debugger, see the `GDB User Manual`.

Free Pascal also supports `gprof`, the GNU profiler. See section [10.4](#) for more information on profiling.

### 10.1 Compiling your program with debugger support

First of all, you must be sure that the compiler is compiled with debugging support. Unfortunately, there is no way to check this at run time, except by trying to compile a program with debugging support.

To compile a program with debugging support, just specify the `-g` option on the command line, as follows:

```
fpc -g hello.pp
```

This will incorporate debugging information in the executable generated from your program source. You will notice that the size of the executable increases substantially because of this<sup>1</sup>.

Note that the above will only incorporate debug information *for the code that has been generated* when compiling `hello.pp`. This means that if you used some units (the system unit, for instance) which were not compiled with debugging support, no debugging support will be available for the code in these units.

There are 2 solutions for this problem.

1. Recompile all units manually with the `-g` option.
2. Specify the 'build' option (`-B`) when compiling with debugging support. This will recompile all units, and insert debugging information in each of the units.

---

<sup>1</sup>A good reason not to include debug information in an executable you plan to distribute.

The second option may have undesirable side effects. It may be that some units aren't found, or compile incorrectly due to missing conditionals, etc.

If all went well, the executable now contains the necessary information with which you can debug it using GNU `gdb`.

## 10.2 Using `gdb` to debug your program

To use `gdb` to debug your program, you can start the debugger, and give it as an option the *full* name of your program:

```
gdb hello
```

Or, under DOS:

```
gdb hello.exe
```

This starts the debugger, and the debugger immediately loads your program into memory, but it does not run the program yet. Instead, you are presented with the following (more or less) message, followed by the `gdb` prompt '`(gdb)`':

```
GNU gdb 6.6.50.20070726-cvs
Copyright (C) 2007 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-suse-linux".
(gdb)
```

The actual prompt will vary depending on your operating system and installed version of `gdb`, of course.

To start the program you can use the `run` command. You can optionally specify command line parameters, which will then be fed to your program, for example:

```
(gdb) run -option -anotheroption needed_argument
```

If your program runs without problems, `gdb` will inform you of this, and return the exit code of your program. If the exit code was zero, then the message '`Program exited normally`' is displayed.

If something went wrong (a segmentation fault or such), `gdb` will stop the execution of your program, and inform you of this with an appropriate message. You can then use the other `gdb` commands to see what happened. Alternatively, you can instruct `gdb` to stop at a certain point in your program, with the `break` command.

Here is a short list of `gdb` commands, which you are likely to need when debugging your program:

**quit** Exit the debugger.

**kill** Stop a running program.

**help** Give help on all `gdb` commands.

**file** Load a new program into the debugger.



4. Objects are difficult to handle, mainly because `gdb` is oriented towards C and C++. The workaround implemented in Free Pascal is that object methods are represented as functions, with an extra parameter `this` (all lowercase!). The name of this function is a concatenation of the object type and the function name, separated by two underscore characters.

For example, the method `TPoint.Draw` would be converted to `TPOINT__DRAW`, and you could stop at it by using:

```
break TPOINT__DRAW
```

5. Global overloaded functions confuse `gdb` because they have the same name. Thus you cannot set a breakpoint at an overloaded function, unless you know its line number, in which case you can set a breakpoint at the starting line number of the function.

## 10.4 Support for `gprof`, the GNU profiler

You can compile your programs with profiling support. For this, you just have to use the compiler switch `-pg`. The compiler will insert the necessary stuff for profiling.

When you have done this, you can run your program as you would normally run it:

```
yourexe
```

Where `yourexe` is the name of your executable.

When your program finishes, a file called `gmon.out` is generated. Then you can start the profiler to see the output. You can benefit from redirecting the output to a file, because it could be quite a lot:

```
gprof yourexe > profile.log
```

Hint: you can use the `-flat` option to reduce the amount of output of `gprof`. It will then only output the information about the timings.

For more information on the GNU profiler `gprof`, see its manual.

## 10.5 Detecting heap memory leaks

Free Pascal has a built in mechanism to detect memory leaks. There is a plug-in unit for the memory manager that analyses the memory allocation/deallocation and prints a memory usage report after the program exits.

The unit that does this is called `heaptrc`. If you want to use it, you should include it as the first unit in your `uses` clause. Alternatively, you can supply the `-gh` switch to the compiler, and it will include the unit automatically for you.

After the program exits, you will get a report looking like this:

```
Marked memory at 0040FA50 invalid
Wrong size : 128 allocated 64 freed
0x00408708
0x0040CB49
0x0040C481
Call trace for block 0x0040FA50 size 128
0x0040CB3D
0x0040C481
```

The output of the `heaptrc` unit is customizable by setting some variables. Output can also be customized using environment variables.

You can find more information about the usage of the `heaptrc` unit in the [Unit Reference](#).

## 10.6 Line numbers in run-time error backtraces

Normally, when a run-time error occurs, you are presented with a list of addresses that represent the call stack backtrace, i.e. the addresses of all procedures that were invoked when the run-time error occurred.

This list is not very informative, so there exists a unit that generates the file names and line numbers of the called procedures using the addresses of the stack backtrace. This unit is called `lineinfo`.

You can use this unit by giving the `-gl` option to the compiler. The unit will be automatically included. It is also possible to use the unit explicitly in your `uses` clause, but you must make sure that you compile your program with debug info.

Here is an example program:

```
program testline;

procedure generateerror255;

begin
    runerror(255);
end;

procedure generateanerror;

begin
    generateerror255;
end;

begin
    generateanerror;
end.
```

When compiled with `-gl`, the following output is generated:

```
Runtime error 255 at 0x0040BDE5
0x0040BDE5  GENERATEERROR255,   line 6 of testline.pp
0x0040BDF0  GENERATEANERROR,   line 13 of testline.pp
0x0040BE0C  main,   line 17 of testline.pp
0x0040B7B1
```

This is more understandable than the normal message. Make sure that all units you use are compiled with debug info, because if they are not, no line number and filename can be found.

## 10.7 Combining `heaptrc` and `lineinfo`

If you combine the `lineinfo` and the `heaptrc` information, then the output of the `heaptrc` unit will contain the names of the files and line numbers of the procedures that occur in the stack backtrace.

In such a case, the output will look something like this:

```
Marked memory at 00410DA0 invalid
Wrong size : 128 allocated 64 freed
  0x004094B8
  0x0040D8F9  main,   line 25 of heapex.pp
  0x0040D231
Call trace for block 0x00410DA0 size 128
  0x0040D8ED  main,   line 23 of heapex.pp
  0x0040D231
```

If lines without filename / line number occur, this means there is a unit which has no debug info included (in the above case, the getmem call itself).

## Appendix A

# Alphabetical listing of command line options

The following is an alphabetical listing of all command line options, as generated by the compiler:

```
Free Pascal Compiler version 2.4.2 [2012/07/25] for i386
Copyright (c) 1993-2010 by Florian Klaempfl
/home/abuild/rpmbuild/BUILD/fpcbuild-2.4.2/fpcsrc/compiler/ppc386 [options] <inputfi
Put + after a boolean switch option to enable it, - to disable it
-a      The compiler doesn't delete the generated assembler file
-al      List sourcecode lines in assembler file
-an      List node info in assembler file
-ap      Use pipes instead of creating temporary assembler files
-ar      List register allocation/release info in assembler file
-at      List temp allocation/release info in assembler file
-A<x>   Output format:
-Adefault Use default assembler
-Aas      Assemble using GNU AS
-Anasmcoff COFF (Go32v2) file using Nasm
-Anasmelf  ELF32 (Linux) file using Nasm
-Anasmwin32Win32 object file using Nasm
-AnasmwdosxWin32/WDOSX object file using Nasm
-Awasm     Obj file using Wasm (Watcom)
-Anasmobj  Obj file using Nasm
-Amasm     Obj file using Masm (Microsoft)
-Atasm     Obj file using Tasm (Borland)
-Aelf      ELF (Linux) using internal writer
-Acoff     COFF (Go32v2) using internal writer
-Apecoff   PE-COFF (Win32) using internal writer
-b       Generate browser info
-bl      Generate local symbol info
-B       Build all modules
-C<x>    Code generation options:
-Ca<x>   Select ABI, see fpc -i for possible values
-Cb      Generate big-endian code
-Cc<x>   Set default calling convention to <x>
-CD      Create also dynamic library (not supported)
-Ce      Compilation with emulated floating point opcodes
-Cf<x>   Select fpu instruction set to use, see fpc -i for possible values
```



```

-gw          Generate DWARFv2 debug information (same as -gw2)
-gw2         Generate DWARFv2 debug information
-gw3         Generate DWARFv3 debug information
-i           Information
-iD          Return compiler date
-iV          Return short compiler version
-iW          Return full compiler version
-iSO         Return compiler OS
-iSP         Return compiler host processor
-iTO         Return target OS
-iTP         Return target processor
-I<x>        Add <x> to include path
-k<x>        Pass <x> to the linker
-l           Write logo
-M<x>        Set language mode to <x>
-Mfpc        Free Pascal dialect (default)
-Mobjfpc     FPC mode with Object Pascal support
-Mdelphi     Delphi 7 compatibility mode
-Mtp         TP/BP 7.0 compatibility mode
-Mmacpas     Macintosh Pascal dialects compatibility mode
-n           Do not read the default config files
-N<x>        Node tree optimizations
-Nu          Unroll loops
-o<x>        Change the name of the executable produced to <x>
-O<x>        Optimizations:
-O-          Disable optimizations
-O1          Level 1 optimizations (quick and debugger friendly)
-O2          Level 2 optimizations (-O1 + quick optimizations)
-O3          Level 3 optimizations (-O2 + slow optimizations)
-Oa<x>=<y>    Set alignment
-Oo[NO]<x>   Enable or disable optimizations, see fpc -i for possible values
-Op<x>       Set target cpu for optimizing, see fpc -i for possible values
-OW<x>       Generate whole-program optimization feedback for optimization <x>,
-Ow<x>       Perform whole-program optimization <x>, see fpc -i for possible values
-Os          Optimize for size rather than speed
-pg          Generate profile code for gprof (defines FPC_PROFILE)
-R<x>        Assembler reading style:
-Rdefault    Use default assembler for target
-Ratt        Read AT&T style assembler
-Rintel      Read Intel style assembler
-S<x>        Syntax options:
-S2          Same as -Mobjfpc
-Sc          Support operators like C (*=, +=, /= and -=)
-Sa          Turn on assertions
-Sd          Same as -Mdelphi
-Se<x>       Error options. <x> is a combination of the following:
    <n> : Compiler halts after the <n> errors (default is 1)
    w : Compiler also halts after warnings
    n : Compiler also halts after notes
    h : Compiler also halts after hints
-Sg          Enable LABEL and GOTO (default in -Mtp and -Mdelphi)
-Sh          Use ansistrings by default instead of shortstrings
-Si          Turn on inlining of procedures/functions declared as "inline"
-Sk          Load fpcylix unit

```

---

## APPENDIX A. ALPHABETICAL LISTING OF COMMAND LINE OPTIONS

---

-SI<x> Set interface style to <x>  
    -SIcom COM compatible interface (default)  
    -SIcorba CORBA compatible interface  
-Sm Support macros like C (global)  
-So Same as -Mtp  
-Ss Constructor name must be init (destructor must be done)  
-St Allow static keyword in objects  
-Sx Enable exception keywords (default in Delphi/ObjFPC modes)  
-s Do not call assembler and linker  
    -sh Generate script to link on host  
    -st Generate script to link on target  
    -sr Skip register allocation phase (use with -alr)  
-T<x> Target operating system:  
    -Temx OS/2 via EMX (including EMX/RSX extender)  
    -Tfreebsd FreeBSD  
    -Tgo32v2 Version 2 of DJ Delorie DOS extender  
    -Tlinux Linux  
    -Tnetbsd NetBSD  
    -Tnetware Novell Netware Module (clib)  
    -Tnetwlibc Novell Netware Module (libc)  
    -Topenbsd OpenBSD  
    -Tos2 OS/2 / eComStation  
    -Tsunos SunOS/Solaris  
    -Tsymbian Symbian OS  
    -Twatcom Watcom compatible DOS extender  
    -Twdosx WDOSX DOS extender  
    -Twin32 Windows 32 Bit  
    -Twince Windows CE  
-u<x> Undefines the symbol <x>  
-U Unit options:  
    -Un Do not check where the unit name matches the file name  
    -Ur Generate release unit files (never automatically recompiled)  
    -US Compile a system unit  
-v<x> Be verbose. <x> is a combination of the following letters:  
    e : Show errors (default)           0 : Show nothing (except errors)  
    w : Show warnings                   u : Show unit info  
    n : Show notes                      t : Show tried/used files  
    h : Show hints                      c : Show conditionals  
    i : Show general info               d : Show debug info  
    l : Show linenumbers               r : Rhide/GCC compatibility mode  
    s : Show time stamps               q : Show message numbers  
    a : Show everything                x : Executable info (Win32 only)  
    b : Write file names messages      p : Write tree.log with parse tree  
        with full path                v : Write fpcdebug.txt with  
                                      lots of debugging info  
    m<x>,<y> : Don't show messages numbered <x> and <y>  
-W<x> Target-specific options (targets)  
    -Wb Create a bundle instead of a library (Darwin)  
    -WB Create a relocatable image (Windows)  
    -WC Specify console type application (EMX, OS/2, Windows)  
    -WD Use DEFFILE to export functions of DLL or EXE (Windows)  
    -We Use external resources (Darwin)  
    -WF Specify full-screen type application (EMX, OS/2)  
    -WG Specify graphic type application (EMX, OS/2, Windows)

|        |  |
|--------|--|
| -Wi    | Use internal resources (Darwin)                                      |
| -WN    | Do not generate relocation code, needed for debugging (Windows)      |
| -WR    | Generate relocation code (Windows)                                   |
| -WX    | Enable executable stack (Linux)                                      |
| -X     | Executable options:  |
| -Xc    | Pass --shared/-dynamic to the linker (BeOS, Darwin, FreeBSD, Linux)  |
| -Xd    | Do not use standard library search path (needed for cross compile)   |
| -Xe    | Use external linker  |
| -Xg    | Create debuginfo in a separate file and add a debuglink section to o |
| -XD    | Try to link units dynamically (defines FPC_LINK_DYNAMIC)             |
| -Xi    | Use internal linker  |
| -Xm    | Generate link map  |
| -XM<x> | Set the name of the 'main' program routine (default is 'main')       |
| -XP<x> | Prepend the binutils names with the prefix <x>                       |
| -Xr<x> | Set the linker's rlink-path to <x> (needed for cross compile, see t  |
| -XR<x> | Prepend <x> to all linker search paths (BeOS, Darwin, FreeBSD, Linu  |
| -Xs    | Strip all symbols from executable                                    |
| -XS    | Try to link units statically (default, defines FPC_LINK_STATIC)      |
| -Xt    | Link with static libraries (-static is passed to linker)             |
| -XX    | Try to smartlink units (defines FPC_LINK_SMART)                      |
| -?     | Show this help   |
| -h     | Shows this help without waiting                                      |

## Appendix B

# Alphabetical list of reserved words

|              |                |               |
|--------------|----------------|---------------|
| absolute     | far            | popstack      |
| abstract     | file           | private       |
| and          | finally        | procedure     |
| array        | for            | program       |
| as           | forward        | property      |
| asm          | function       | protected     |
| assembler    | goto           | public        |
| begin        | if             | raise         |
| break        | implementation | record        |
| case         | in             | reintroduce   |
| cdecl        | index          | repeat        |
| class        | inherited      | self          |
| const        | initialization | set           |
| constructor  | inline         | shl           |
| continue     | interface      | shr           |
| cppclass     | interrupt      | stdcall       |
| deprecated   | is             | string        |
| destructor   | label          | then          |
| div          | library        | to            |
| do           | mod            | true          |
| downto       | name           | try           |
| else         | near           | type          |
| end          | nil            | unimplemented |
| except       | not            | unit          |
| exit         | object         | until         |
| export       | of             | uses          |
| exports      | on             | var           |
| external     | operator       | virtual       |
| experimental | or             | while         |
| fail         | otherwise      | with          |
| false        | packed         | xor           |

## Appendix C

# Compiler messages

This appendix is meant to list all the compiler messages. The list of messages is generated from the compiler source itself, and should be fairly complete. At this point, only assembler errors are not in the list.

For an explanation of how to control the messages, section [5.1.2](#), page 25.

### C.1 General compiler messages

This section gives the compiler messages which are not fatal, but which display useful information. The number of such messages can be controlled with the various verbosity level `-v` switches.

**Compiler: arg1** When the `-vt` switch is used, this line tells you what compiler is used.

**Compiler OS: arg1** When the `-vd` switch is used, this line tells you what the source operating system is.

**Info: Target OS: arg1** When the `-vd` switch is used, this line tells you what the target operating system is.

**Using executable path: arg1** When the `-vt` switch is used, this line tells you where the compiler looks for its binaries.

**Using unit path: arg1** When the `-vt` switch is used, this line tells you where the compiler looks for compiled units. You can set this path with the `-Fu` option.

**Using include path: arg1** When the `-vt` switch is used, this line tells you where the compiler looks for its include files (files used in `{ $I xxx }` statements). You can set this path with the `-Fi` option.

**Using library path: arg1** When the `-vt` switch is used, this line tells you where the compiler looks for the libraries. You can set this path with the `-Fl` option.

**Using object path: arg1** When the `-vt` switch is used, this line tells you where the compiler looks for object files you link in (files used in `{ $L xxx }` statements). You can set this path with the `-Fo` option.

**Info: arg1 lines compiled, arg2 sec arg3** When the `-vi` switch is used, the compiler reports the number of lines compiled, and the time it took to compile them (real time, not program time).

**Fatal: No memory left** The compiler doesn't have enough memory to compile your program. There are several remedies for this:

- If you're using the build option of the compiler, try compiling the different units manually.
- If you're compiling a huge program, split it up into units, and compile these separately.
- If the previous two don't work, recompile the compiler with a bigger heap. (You can use the `-Ch` option for this, `-Ch` (see page 28).)

**Info: Writing Resource String Table file: arg1** This message is shown when the compiler writes the Resource String Table file containing all the resource strings for a program.

**Error: Writing Resource String Table file: arg1** This message is shown when the compiler encounters an error when writing the Resource String Table file.

**Info: Fatal:** Prefix for Fatal Errors.

**Info: Error:** Prefix for Errors.

**Info: Warning:** Prefix for Warnings.

**Info: Note:** Prefix for Notes.

**Info: Hint:** Prefix for Hints.

**Error: Path "arg1" does not exist** The specified path does not exist.

**Fatal: Compilation aborted** Compilation was aborted.

**bytes code** The size of the generated executable code, in bytes.

**bytes data** The size of the generated program data, in bytes.

**Info: arg1 warning(s) issued** Total number of warnings issued during compilation.

**Info: arg1 hint(s) issued** Total number of hints issued during compilation.

**Info: arg1 note(s) issued** Total number of notes issued during compilation.

## C.2 Scanner messages.

This section lists the messages that the scanner emits. The scanner takes care of the lexical structure of the pascal file, i.e. it tries to find reserved words, strings, etc. It also takes care of directives and conditional compilation handling.

**Fatal: Unexpected end of file** This typically happens in one of the following cases:

- The source file ends before the final `end.` statement. This happens mostly when the `begin` and `end` statements aren't balanced;
- An include file ends in the middle of a statement.
- A comment was not closed.

**Fatal: String exceeds line** There is a missing closing `'` in a string, so it occupies multiple lines.

**Fatal: illegal character "arg1" (arg2)** An illegal character was encountered in the input file.

**Fatal: Syntax error, "arg1" expected but "arg2" found** This indicates that the compiler expected a different token than the one you typed. It can occur almost anywhere it is possible to make an error against the Pascal language.







- Error: Mode switch "arg1" not allowed here** A mode switch has already been encountered, or, in the case of option `-Mmacpas`, a mode switch occurs after `UNIT`.
- Error: Compile time variable or macro "arg1" is not defined.** Thus the conditional compile time expression cannot be evaluated. Only in mode `MacPas`.
- Error: UTF-8 code greater than 65535 found** Free Pascal handles UTF-8 strings internally as widestrings, i.e. the char codes are limited to 65535.
- Error: Malformed UTF-8 string** The given string isn't a valid UTF-8 string.
- UTF-8 signature found, using UTF-8 encoding** The compiler found a UTF-8 encoding signature (`$ef`, `$bb`, `$bf`) at the beginning of a file, so it interprets it as a UTF-8 file.
- Error: Compile time expression: Wanted arg1 but got arg2 at arg3** The type-check of a compile time expression failed.
- Note: APPTYPE is not supported by the target OS** The `{ $APPTYPE }` directive is supported by certain operating systems only.
- Error: Illegal optimization specified "arg1"** You specified an optimization with the `{ $OPTIMIZATION xxx }` directive, and the compiler didn't recognize the optimization you specified.
- Warning: SETPEFLAGS is not supported by the target OS** The `{ $SETPEFLAGS }` directive is not supported by the target OS.
- Warning: IMAGEBASE is not supported by the target OS** The `{ $IMAGEBASE }` directive is not supported by the target OS.
- Warning: MINSTACKSIZE is not supported by the target OS** The `{ $MINSTACKSIZE }` directive is not supported by the target OS.
- Warning: MAXSTACKSIZE is not supported by the target OS** The `{ $MAXSTACKSIZE }` directive is not supported by the target OS.
- Error: Illegal state for \$WARN directive** Only `ON` and `OFF` can be used as state with a `{ $WARN }` compiler directive.
- Error: Illegal set packing value** Only `0`, `1`, `2`, `4`, `8`, `DEFAULT` and `NORMAL` are allowed as pack-set parameters.
- Warning: PIC directive or switch ignored** Several targets, such as `WINDOWS`, do not support nor need `PIC`, so the `PIC` directive and switch are ignored.
- Warning: The switch "arg1" is not supported by the currently selected target** Some compiler switches like `$E` are not supported by all targets.
- Warning: Framework-related options are only supported for Darwin/Mac OS X** Frameworks are not a known concept, or at least not supported by `FPC`, on operating systems other than `Darwin/Mac OS X`.
- Error: Illegal minimal floating point constant precision "arg1"** Valid minimal precisions for floating point constants are `default`, `32` and `64`, which mean respectively minimal (usually 32 bit), 32 bit and 64 bit precision.
- Warning: Overriding name of "main" procedure multiple times, was previously set to "arg1"** The name for the main entry procedure is specified more than once. Only the last name will be used.





**Warning: use extended syntax of NEW and DISPOSE for instances of objects** If you have a pointer `a` to an object type, then the statement `new(a)` will not initialize the object (i.e. the constructor isn't called), although space will be allocated. You should issue the `new(a, init)` statement. This will allocate space, and call the constructor of the object.

**Warning: use of NEW or DISPOSE for untyped pointers is meaningless**

**Error: use of NEW or DISPOSE is not possible for untyped pointers** You cannot use `new(p)` or `dispose(p)` if `p` is an untyped pointer because no size is associated to an untyped pointer. It is accepted for compatibility in `TP` and `DELPHI` modes, but the compiler will still warn you if it finds such a construct.

**Error: class identifier expected** This happens when the compiler scans a procedure declaration that contains a dot, i.e., an object or class method, but the type in front of the dot is not a known type.

**Error: type identifier not allowed here** You cannot use a type inside an expression.

**Error: method identifier expected** This identifier is not a method. This happens when the compiler scans a procedure declaration that contains a dot, i.e., an object or class method, but the procedure name is not a procedure of this type.

**Error: function header doesn't match any method of this class "arg1"** This identifier is not a method. This happens when the compiler scans a procedure declaration that contains a dot, i.e., an object or class method, but the procedure name is not a procedure of this type.

**procedure/function arg1** When using the `-vd` switch, the compiler tells you when it starts processing a procedure or function implementation.

**Error: Illegal floating point constant** The compiler expects a floating point expression, and gets something else.

**Error: FAIL can be used in constructors only** You are using the `fail` keyword outside a constructor method.

**Error: Destructors can't have parameters** You are declaring a destructor with a parameter list. Destructor methods cannot have parameters.

**Error: Only class methods can be referred with class references** This error occurs in a situation like the following:

```
Type :  
    Tclass = Class of Tobject;  
  
Var C : Tclass;  
  
begin  
    ...  
    C.free
```

`Free` is not a class method and hence cannot be called with a class reference.

**Error: Only class methods can be accessed in class methods** This is related to the previous error. You cannot call a method of an object from inside a class method. The following code would produce this error:



- Error: Class isn't a parent class of the current class** When calling inherited methods, you are trying to call a method of a non-related class. You can only call an inherited method of a parent class.
- Error: SELF is only allowed in methods** You are trying to use the `self` parameter outside an object's method. Only methods get passed the `self` parameters.
- Error: Methods can be only in other methods called direct with type identifier of the class** A construction like `sometype.somemethod` is only allowed in a method.
- Error: Illegal use of ':'** You are using the format `:` (colon) 2 times on an expression that is not a real expression.
- Error: range check error in set constructor or duplicate set element** The declaration of a set contains an error. Either one of the elements is outside the range of the set type, or two of the elements are in fact the same.
- Error: Pointer to object expected** You specified an illegal type in a `new` statement. The extended syntax of `new` needs an object as a parameter.
- Error: Expression must be constructor call** When using the extended syntax of `new`, you must specify the constructor method of the object you are trying to create. The procedure you specified is not a constructor.
- Error: Expression must be destructor call** When using the extended syntax of `dispose`, you must specify the destructor method of the object you are trying to dispose of. The procedure you specified is not a destructor.
- Error: Illegal order of record elements** When declaring a constant record, you specified the fields in the wrong order.
- Error: Expression type must be class or record type** A `with` statement needs an argument that is of the type `record` or `class`. You are using `with` on an expression that is not of this type.
- Error: Procedures can't return a value** In Free Pascal, you can specify a return value for a function when using the `exit` statement. This error occurs when you try to do this with a procedure. Procedures cannot return a value.
- Error: constructors and destructors must be methods** You're declaring a procedure as destructor or constructor, when the procedure isn't a class method.
- Error: Operator is not overloaded** You're trying to use an overloaded operator when it is not overloaded for this type.
- Error: Impossible to overload assignment for equal types** You cannot overload assignment for types that the compiler considers as equal.
- Error: Impossible operator overload** The combination of operator, arguments and return type are incompatible.
- Error: Re-raise isn't possible there** You are trying to re-raise an exception where it is not allowed. You can only re-raise exceptions in an `except` block.
- Error: The extended syntax of new or dispose isn't allowed for a class** You cannot generate an instance of a class with the extended syntax of `new`. The constructor must be used for that. For the same reason, you cannot call `dispose` to de-allocate an instance of a class, the destructor must be used for that.









```
const
  p : procedure;stdcall=nil;
  p : procedure stdcall=nil;
```

**Error: The value for a property index must be of an ordinal type** The value you use to index a property must be of an ordinal type, for example an integer or enumerated type.

**Error: Procedure name too short to be exported** The length of the procedure/function name must be at least 2 characters long. This is because of a bug in dlltool which doesn't parse the .def file correctly with a name of length 1.

**Error: No DEFFILE entry can be generated for unit global vars**

**Error: Compile without -WD option** You need to compile this file without the -WD switch on the command line.

**Fatal: You need ObjFpc (-S2) or Delphi (-Sd) mode to compile this module** You need to use {\$MODE OBJFPC} or {\$MODE DELPHI} to compile this file. Or use the corresponding command line switch, either -Mobjfpc or -MDelphi.

**Error: Can't export with index under arg1** Exporting of functions or procedures with a specified index is not supported on this target.

**Error: Exporting of variables is not supported under arg1** Exporting of variables is not supported on this target.

**Error: Improper GUID syntax** The GUID indication does not have the proper syntax. It should be of the form

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where each X represents a hexadecimal digit.

**Warning: Procedure named "arg1" not found that is suitable for implementing the arg2.arg3** The compiler cannot find a suitable procedure which implements the given method of an interface. A procedure with the same name is found, but the arguments do not match.

**Error: interface identifier expected** This happens when the compiler scans a class declaration that contains interface function name mapping code like this:

```
type
  TMyObject = class(TObject, IDispatch)
    function IUnknown.QueryInterface=MyQueryInterface;
    ....
```

and the interface before the dot is not listed in the inheritance list.

**Error: Type "arg1" can't be used as array index type** Types like qword or int64 aren't allowed as array index type.

**Error: Con- and destructors aren't allowed in interfaces** Constructor and destructor declarations aren't allowed in interfaces. In the most cases method QueryInterface of IUnknown can be used to create a new interface.





```
...
procedure p1;
label
  l1;

  procedure p2;
  begin
    goto l1; // This goto ISN'T allowed
  end;

begin
  p2
l1:
end;
...
```

**Fatal: Procedure too complex, it requires too many registers** Your procedure body is too long for the compiler. You should split the procedure into multiple smaller procedures.

**Error: Illegal expression** This can occur under many circumstances. Usually when trying to evaluate constant expressions.

**Error: Invalid integer expression** You made an expression which isn't an integer, and the compiler expects the result to be an integer.

**Error: Illegal qualifier** One of the following is happening :

- You're trying to access a field of a variable that is not a record.
- You're indexing a variable that is not an array.
- You're dereferencing a variable that is not a pointer.

**Error: High range limit < low range limit** You are declaring a subrange, and the high limit is less than the low limit of the range.

**Error: Exit's parameter must be the name of the procedure it is used in** Non local exit is not allowed. This error occurs only in mode MacPas.

**Error: Illegal assignment to for-loop variable "arg1"** The type of a `for` loop variable must be an ordinal type. Loop variables cannot be reals or strings. You also cannot assign values to loop variables inside the loop (Except in Delphi and TP modes). Use a `while` or `repeat` loop instead if you need to do something like that, since those constructs were built for that.

**Error: Can't declare local variable as EXTERNAL** Declaring local variables as external is not allowed. Only global variables can reference external variables.

**Error: Procedure is already declared EXTERNAL** The procedure is already declared with the `EXTERNAL` directive in an interface or forward declaration.

**Warning: Implicit uses of Variants unit** The Variant type is used in the unit without any `uses` unit using the Variants unit. The compiler has implicitly added the Variants unit to the uses list. To remove this warning the Variants unit needs to be added to the uses statement.

**Error: Class and static methods can't be used in INTERFACES** The specifier `class` and directive `static` can't be used in interfaces because all methods of an interface must be public.











**Error: Illegal constant passed to internal math function** The constant argument passed to a `ln` or `sqrt` function is out of the definition range of these functions.

**Error: Can't take the address of constant expressions** It is not possible to get the address of a constant expression, because they aren't stored in memory. You can try making it a typed constant. This error can also be displayed if you try to pass a property to a var parameter.

**Error: Argument can't be assigned to** Only expressions which can be on the left side of an assignment can be passed as call by reference arguments.

Remark: Properties can be used on the left side of an assignment, nevertheless they cannot be used as arguments.

**Error: Can't assign local procedure/function to procedure variable** It's not allowed to assign a local procedure/function to a procedure variable, because the calling convention of a local procedure/function is different. You can only assign local procedure/function to a void pointer.

**Error: Can't assign values to an address** It is not allowed to assign a value to an address of a variable, constant, procedure or function. You can try compiling with `-So` if the identifier is a procedure variable.

**Error: Can't assign values to const variable** It's not allowed to assign a value to a variable which is declared as a `const`. This is normally a parameter declared as `const`. To allow changing the value, pass the parameter by value, or a parameter by reference (using `var`).

**Error: Array type required** If you are accessing a variable using an index '`[<x>]`' then the type must be an array. In FPC mode a pointer is also allowed.

**Error: interface type expected, but got "arg1"** The compiler expected to encounter an interface type name, but got something else. The following code would produce this error:

```
Type
  TMyStream = Class(TStream, Integer)
```

**Hint: Mixing signed expressions and longwords gives a 64bit result** If you divide (or calculate the modulus of) a signed expression by a longword (or vice versa), or if you have overflow and/or range checking turned on and use an arithmetic expression (`+`, `-`, `*`, `div`, `mod`) in which both signed numbers and longwords appear, then everything has to be evaluated in 64-bit arithmetic which is slower than normal 32-bit arithmetic. You can avoid this by typecasting one operand so it matches the result type of the other one.

**Warning: Mixing signed expressions and cardinals here may cause a range check error** If you use a binary operator (`and`, `or`, `xor`) and one of the operands is a longword while the other one is a signed expression, then, if range checking is turned on, you may get a range check error because in such a case both operands are converted to longword before the operation is carried out. You can avoid this by typecasting one operand so it matches the result type of the other one.

**Error: Typecast has different size (arg1 -> arg2) in assignment** Type casting to a type with a different size is not allowed when the variable is used in an assignment.

**Error: enums with assignments can't be used as array index** When you declared an enumeration type which has C-like assignments, such as in the following:

```
Tenum = (a,b,e:=5);
```









- Hint: Local arg1 "arg2" is not used** A local symbol is never used.
- Note: Private field "arg1.arg2" is never used** The indicated private field is defined, but is never used in the code.
- Note: Private field "arg1.arg2" is assigned but never used** The indicated private field is declared and assigned to, but never read.
- Note: Private method "arg1.arg2" never used** The indicated private method is declared but is never used in the code.
- Error: Set type expected** The variable or expression is not of type `set`. This happens in an `in` statement.
- Warning: Function result does not seem to be set** You can get this warning if the compiler thinks that a function return value is not set. This will not be displayed for assembler procedures, or procedures that contain assembler blocks.
- Warning: Type "arg1" is not aligned correctly in current record for C** Arrays with sizes not multiples of 4 will be wrongly aligned for C structures.
- Error: Unknown record field identifier "arg1"** The field doesn't exist in the record/object definition.
- Warning: Local variable "arg1" does not seem to be initialized** This message is displayed if the compiler thinks that a variable will be used (i.e. it appears in the right-hand side of an expression) when it was not initialized first (i.e. appeared in the left-hand side of an assignment).
- Warning: Variable "arg1" does not seem to be initialized** This message is displayed if the compiler thinks that a variable will be used (i.e. it appears in the right-hand side of an expression) when it was not initialized first (i.e. appeared in the left-hand side of an assignment).
- Error: identifier idents no member "arg1"** This error is generated when an identifier of a record, field or method is accessed while it is not defined.
- Hint: Found declaration: arg1** You get this when you use the `-vh` switch. In the case of an overloaded procedure not being found. Then all candidate overloaded procedures are listed, with their parameter lists.
- Error: Data element too large** You get this when you declare a data element whose size exceeds the prescribed limit (2 Gb on 80386+/68020+ processors).
- Error: No matching implementation for interface method "arg1" found** There was no matching method found which could implement the interface method. Check argument types and result type of the methods.
- Warning: Symbol "arg1" is deprecated** This means that a symbol (a variable, routine, etc...) which is declared as `deprecated` is used. Deprecated symbols may no longer be available in newer versions of the unit / library. Use of this symbol should be avoided as much as possible.
- Warning: Symbol "arg1" is not portable** This means that a symbol (a variable, routine, etc...) which is declared as `platform` is used. This symbol's value, use and availability is platform specific and should not be used if the source code must be portable.
- Warning: Symbol "arg1" is not implemented** This means that a symbol (a variable, routine, etc...) which is declared as `unimplemented` is used. This symbol is defined, but is not yet implemented on this specific platform.
- Error: Can't create unique type from this type** Only simple types like ordinal, float and string types are supported when redefining a type with `type newtype = type oldtype;`.

- Hint: Local variable "arg1" does not seem to be initialized** This message is displayed if the compiler thinks that a variable will be used (i.e. it appears in the right-hand side of an expression) when it was not initialized first (i.e. it did not appear in the left-hand side of an assignment).
- Hint: Variable "arg1" does not seem to be initialized** This message is displayed if the compiler thinks that a variable will be used (i.e. it appears in the right-hand side of an expression) when it was not initialized first (i.e. it did not appear in the left-hand side of an assignment).
- Warning: Function result variable does not seem to be initialized** This message is displayed if the compiler thinks that the function result variable will be used (i.e. it appears in the right-hand side of an expression) before it is initialized (i.e. before it appeared in the left-hand side of an assignment).
- Hint: Function result variable does not seem to be initialized** This message is displayed if the compiler thinks that the function result variable will be used (i.e. it appears in the right-hand side of an expression) before it is initialized (i.e. it appears in the left-hand side of an assignment).
- Warning: Variable "arg1" read but nowhere assigned** You have read the value of a variable, but nowhere assigned a value to it.
- Hint: Found abstract method: arg1** When getting a warning about constructing a class/object with abstract methods you get this hint to assist you in finding the affected method.
- Warning: Symbol "arg1" is experimental** This means that a symbol (a variable, routine, etc...) which is declared as `experimental` is used. Experimental symbols might disappear or change semantics in future versions. Usage of this symbol should be avoided as much as possible.
- Warning: Forward declaration "arg1" not resolved, assumed external** This happens if you declare a function in the `interface` of a unit in `macpas` mode, but do not implement it.
- Warning: Symbol "arg1" is belongs to a library** This means that a symbol (a variable, routine, etc...) which is declared as `library` is used. Library symbols may not be available in other libraries.
- Warning: Symbol "arg1" is deprecated: "arg2"** This means that a symbol (a variable, routine, etc...) which is declared as `deprecated` is used. Deprecated symbols may no longer be available in newer versions of the unit / library. Use of this symbol should be avoided as much as possible.
- Error: Cannot find an enumerator for the type "arg1"** This means that compiler cannot find an appropriate enumerator to use in the `for-in` loop. To create an enumerator you need to define an operator enumerator or add a public or published `GetEnumerator` method to the class or object definition.
- Error: Cannot find a "MoveNext" method in enumerator "arg1"** This means that compiler cannot find a public `MoveNext` method with the `Boolean` return type in the enumerator class or object definition.
- Error: Cannot find a "Current" property in enumerator "arg1"** This means that compiler cannot find a public `Current` property in the enumerator class or object definition.

## C.6 Code generator messages

This section lists all messages that can be displayed if the code generator encounters an error condition.



**Error: Jump in or outside of an exception block** It is not allowed to jump in or outside of an exception block like `try..finally..end;`. For example, the following code will produce this error:

```
label 1;

...

try
  if not(final) then
    goto 1;    // this line will cause an error
  finally
    ...
end;
1:
...
```

**Error: Control flow statements aren't allowed in a finally block** It isn't allowed to use the control flow statements `break`, `continue` and `exit` inside a finally statement. The following example shows the problem:

```
...
try
  p;
finally
  ...
  exit; // This exit ISN'T allowed
end;
...
```

If the procedure `p` raises an exception the finally block is executed. If the execution reaches the `exit`, it's unclear what to do: exit the procedure or search for another exception handler.

**Warning: Parameters size exceeds limit for certain cpu's** This indicates that you are declaring more than 64K of parameters, which might not be supported on other processor targets.

**Warning: Local variable size exceed limit for certain cpu's** This indicates that you are declaring more than 32K of local variables, which might not be supported on other processor targets.

**Error: Local variables size exceeds supported limit** This indicates that you are declaring more than 32K of local variables, which is not supported by this processor.

**Error: BREAK not allowed** You're trying to use `break` outside a loop construction.

**Error: CONTINUE not allowed** You're trying to use `continue` outside a loop construction.

**Fatal: Unknown compilerproc "arg1". Check if you use the correct run time library.** The compiler expects that the runtime library contains certain subroutines. If you see this error and you didn't change the runtime library code, it's very likely that the runtime library you're using doesn't match the compiler in use. If you changed the runtime library this error means that you removed a subroutine which the compiler needs for internal use.

**Fatal: Cannot find system type "arg1". Check if you use the correct run time library.** The compiler expects that the runtime library contains certain type definitions. If you see this error and you didn't change the runtime library code, it's very likely that the runtime library you're using doesn't match the compiler in use. If you changed the runtime library this error means that you removed a type which the compiler needs for internal use.

**Hint: Inherited call to abstract method ignored** This message appears only in Delphi mode when you call an abstract method of a parent class via `inherited;`. The call is then ignored.

**Error: Goto label "arg1" not defined or optimized away** The label used in the goto definition is not defined or optimized away by the unreachable code elimination.

## C.7 Errors of assembling/linking stage

This section lists errors that occur when the compiler is processing the command line or handling the configuration files.

**Warning: Source operating system redefined** The source operating system is redefined.

**Info: Assembling (pipe) arg1** Assembling using a pipe to an external assembler.

**Error: Can't create assembler file: arg1** The mentioned file can't be created. Check if you have access permissions to create this file.

**Error: Can't create object file: arg1** The mentioned file can't be created. Check if you have got access permissions to create this file.

**Error: Can't create archive file: arg1** The mentioned file can't be created. Check if you have access permissions to create this file.

**Error: Assembler arg1 not found, switching to external assembling** The assembler program was not found. The compiler will produce a script that can be used to assemble and link the program.

**Using assembler: arg1** An informational message saying which assembler is being used.

**Error: Error while assembling exitcode arg1** There was an error while assembling the file using an external assembler. Consult the documentation of the assembler tool to find out more information on this error.

**Error: Can't call the assembler, error arg1 switching to external assembling** An error occurred when calling an external assembler. The compiler will produce a script that can be used to assemble and link the program.

**Info: Assembling arg1** An informational message stating which file is being assembled.

**Info: Assembling with smartlinking arg1** An informational message stating which file is being assembled using smartlinking.

**Warning: Object arg1 not found, Linking may fail !** One of the object files is missing, and linking will probably fail. Check your paths.

**Warning: Library arg1 not found, Linking may fail !** One of the library files is missing, and linking will probably fail. Check your paths.

**Error: Error while linking** Generic error while linking.

**Error: Can't call the linker, switching to external linking** An error occurred when calling an external linker. The compiler will produce a script that can be used to assemble and link the program.

**Info: Linking arg1** An informational message, showing which program or library is being linked.

**Error: Util arg1 not found, switching to external linking** An external tool was not found. The compiler will produce a script that can be used to assemble and link or postprocess the program.

**Using util arg1** An informational message, showing which external program (usually a postprocessor) is being used.

**Error: Creation of Executables not supported** Creating executable programs is not supported for this platform, because it was not yet implemented in the compiler.

**Error: Creation of Dynamic/Shared Libraries not supported** Creating dynamically loadable libraries is not supported for this platform, because it was not yet implemented in the compiler.

**Info: Closing script arg1** Informational message showing when writing of the external assembling and linking script is finished.

**Error: resource compiler "arg1" not found, switching to external mode** An external resource compiler was not found. The compiler will produce a script that can be used to assemble, compile resources and link or postprocess the program.

**Info: Compiling resource arg1** An informational message, showing which resource is being compiled.

**unit arg1 can't be statically linked, switching to smart linking** Static linking was requested, but a unit which is not statically linkable was used.

**unit arg1 can't be smart linked, switching to static linking** Smart linking was requested, but a unit which is not smart-linkable was used.

**unit arg1 can't be shared linked, switching to static linking** Shared linking was requested, but a unit which is not shared-linkable was used.

**Error: unit arg1 can't be smart or static linked** Smart or static linking was requested, but a unit which cannot be used for either was used.

**Error: unit arg1 can't be shared or static linked** Shared or static linking was requested, but a unit which cannot be used for either was used.

**Calling resource compiler "arg1" with "arg2" as command line** An informational message showing which command line is used for the resource compiler.

**Error: Error while compiling resources** The resource compiler or converter returned an error.

**Error: Can't call the resource compiler "arg1", switching to external mode** An error occurred when calling a resource compiler. The compiler will produce a script that can be used to assemble, compile resources and link or postprocess the program.

**Error: Can't open resource file "arg1"** An error occurred resource file cannot be opened.

**Error: Can't write resource file "arg1"** An error occurred resource file cannot be written.





**Recompiling arg1, source found only** When you use the `-vu` flag, these messages tell you why the current unit is recompiled.

**Recompiling unit, static lib is older than ppufile** When you use the `-vu` flag, the compiler warns if the static library of the unit is older than the unit file itself.

**Recompiling unit, shared lib is older than ppufile** When you use the `-vu` flag, the compiler warns if the shared library of the unit is older than the unit file itself.

**Recompiling unit, obj and asm are older than ppufile** When you use the `-vu` flag, the compiler warns if the assembler or object file of the unit is older than the unit file itself.

**Recompiling unit, obj is older than asm** When you use the `-vu` flag, the compiler warns if the assembler file of the unit is older than the object file of the unit.

**Parsing interface of arg1** When you use the `-vu` flag, the compiler warns that it starts parsing the interface part of the unit.

**Parsing implementation of arg1** When you use the `-vu` flag, the compiler warns that it starts parsing the implementation part of the unit.

**Second load for unit arg1** When you use the `-vu` flag, the compiler warns that it starts recompiling a unit for the second time. This can happen with interdependent units.

**PPU Check file arg1 time arg2** When you use the `-vu` flag, the compiler shows the filename and date and time of the file on which a recompile depends.

**Warning: Can't recompile unit arg1, but found modified include files** A unit was found to have modified include files, but some source files were not found, so recompilation is impossible.

**File arg1 is newer than PPU file arg2** A modified source file for a compiler unit was found.

**Trying to use a unit which was compiled with a different FPU mode** Trying to compile code while using units which were not compiled with the same floating point format mode. Either all code should be compiled with FPU emulation on, or with FPU emulation off.

**Loading interface units from arg1** When you use the `-vu` flag, the compiler warns that it is starting to load the units defined in the interface part of the unit.

**Loading implementation units from arg1** When you use the `-vu` flag, the compiler warns that it is starting to load the units defined in the implementation part of the unit.

**Interface CRC changed for unit arg1** When you use the `-vu` flag, the compiler warns that the CRC calculated for the interface has been changed after the implementation has been parsed.

**Implementation CRC changed for unit arg1** When you use the `-vu` flag, the compiler warns that the CRC calculated has been changed after the implementation has been parsed.

**Finished compiling unit arg1** When you use the `-vu` flag, the compiler warns that it has finished compiling the unit.

**Add dependency of arg1 to arg2** When you use the `-vu` flag, the compiler warns that it has added a dependency between the two units.

**No reload, is caller: arg1** When you use the `-vu` flag, the compiler warns that it will not reload the unit because it is the unit that wants to load this unit.

**No reload, already in second compile: arg1** When you use the `-vu` flag, the compiler warns that it will not reload the unit because it is already in a second recompile.

**Flag for reload: arg1** When you use the `-vu` flag, the compiler warns that it has to reload the unit.

**Forced reloading** When you use the `-vu` flag, the compiler warns that it is reloading the unit because it was required.

**Previous state of arg1: arg2** When you use the `-vu` flag, the compiler shows the previous state of the unit.

**Already compiling arg1, setting second compile** When you use the `-vu` flag, the compiler warns that it is starting to recompile a unit for the second time. This can happen with interdependent units.

**Loading unit arg1** When you use the `-vu` flag, the compiler warns that it starts loading the unit.

**Finished loading unit arg1** When you use the `-vu` flag, the compiler warns that it finished loading the unit.

**Registering new unit arg1** When you use the `-vu` flag, the compiler warns that it has found a new unit and is registering it in the internal lists.

**Re-resolving unit arg1** When you use the `-vu` flag, the compiler warns that it has to recalculate the internal data of the unit.

**Skipping re-resolving unit arg1, still loading used units** When you use the `-vu` flag, the compiler warns that it is skipping the recalculation of the internal data of the unit because there is no data to recalculate.

**Unloading resource unit arg1 (not needed)** When you use the `-vu` flag, the compiler warns that it is unloading the resource handling unit, since no resources are used.

**Error: Unit arg1 was compiled using a different whole program optimization feedback input (arg2, arg3); recompile it**  
When a unit has been compiled using a particular whole program optimization (wpo) feedback file (`-FW<x> -OW<x>`), this compiled version of the unit is specialised for that particular compilation scenario and cannot be used in any other context. It has to be recompiled before you can use it in another program or with another wpo feedback input file.

## C.11 Command line handling errors

This section lists errors that occur when the compiler is processing the command line or handling the configuration files.

**Warning: Only one source file supported, changing source file to compile from "arg1" into "arg2"**  
You can specify only one source file on the command line. The last one will be compiled, others will be ignored. This may indicate that you forgot a `'-'` sign.

**Warning: DEF file can be created only for OS/2** This option can only be specified when you're compiling for OS/2.

**Error: nested response files are not supported** You cannot nest response files with the `@file` command line option.

**Fatal: No source file name in command line** The compiler expects a source file name on the command line.

**Note: No option inside arg1 config file** The compiler didn't find any option in that config file.

**Error: Illegal parameter: arg1** You specified an unknown option.

**Hint: -? writes help pages** When an unknown option is given, this message is displayed.







**Fatal: Cannot find "arg1" or "arg2" to extract symbol liveness information from linked program**

Certain symbol liveness collectors need a helper program to extract the symbol information from the linked program. This helper program is normally 'nm', which is part of the GNU binutils.

**Error: Error during reading symbol liveness information produced by "arg1"** An error occurred during the reading of the symbol liveness file that was generated using the 'nm' or 'objdump' program. The reason can be that it was shorter than expected, or that its format was not understood.

**Fatal: Error executing "arg1" (exitcode: arg2) to extract symbol information from linked program**

Certain symbol liveness collectors need a helper program to extract the symbol information from the linked program. The helper program produced the reported error code when it was run on the linked program.

**Error: Collection of symbol liveness information can only help when using smart linking, use -CX -XX**

Whether or not a symbol is live is determined by looking whether it exists in the final linked program. Without smart linking/dead code stripping, all symbols are always included, regardless of whether they are actually used or not. So in that case all symbols will be seen as live, which makes this optimization ineffective.

**Error: Cannot create specified whole program optimisation feedback file "arg1"** The compiler is unable to create the file specified using the -FW parameter to store the whole program optimisation information.

## C.13 Assembler reader errors.

This section lists the errors that are generated by the inline assembler reader. They are *not* the messages of the assembler itself.

### C.13.1 General assembler errors

**Divide by zero in asm evaluator** This fatal error is reported when a constant assembler expression performs a division by zero.

**Evaluator stack overflow, Evaluator stack underflow** These fatal error is reported when a constant assembler expression is too big to be evaluated by the constant parser. Try reducing the number of terms.

**Invalid numeric format in asm evaluator** This fatal error is reported when a non-numeric value is detected by the constant parser. Normally this error should never occur.

**Invalid Operator in asm evaluator** This fatal error is reported when a mathematical operator is detected by the constant parser. Normally this error should never occur.

**Unknown error in asm evaluator** This fatal error is reported when an internal error is detected by the constant parser. Normally this error should never occur.

**Invalid numeric value** This warning is emitted when a conversion from octal, binary or hexadecimal to decimal is outside of the supported range.

**Escape sequence ignored** This error is emitted when a non ANSI C escape sequence is detected in a C string.

**Asm syntax error - Prefix not found** This occurs when trying to use a non-valid prefix instruction.





### C.13.2 i386 specific errors

**repeat prefix and a segment override on <= i386 ...** A problem with interrupts and a prefix instruction may occur and may cause false results on 386 and earlier computers.

**Fwait can cause emulation problems with emu387** This warning is reported when using the FWAIT instruction. It can cause emulation problems on systems which use the em387.dxe emulator.

**You need GNU as version >= 2.81 to compile this MMX code** MMX assembler code can only be compiled using GAS v2.8.1 or later.

**NEAR ignored**

**FAR ignored** NEAR and FAR are ignored in the Intel assemblers, but are still accepted for compatibility with the 16-bit code model.

**Invalid size for MOVSX/MOVZX**

**16-bit base in 32-bit segment**

**16-bit index in 32-bit segment** 16-bit addressing is not supported. You must use 32-bit addressing.

**Constant reference not allowed** It is not allowed to try to address a constant memory address in protected mode.

**Segment overrides not supported** Intel style (eg: rep ds stosb) segment overrides are not supported by the assembler parser.

**Expressions of the form [sreg:reg...] are currently not supported** To access a memory operand in a different segment, you should use the sreg:[reg...] syntax instead of [sreg:reg...]

**Size suffix and destination register do not match** In intel AT&T syntax, you are using a register size which does not concord with the operand size specified.

**Invalid assembler syntax. No ref with brackets**

**Trying to use a negative index register**

**Local symbols not allowed as references**

**Invalid operand in bracket expression**

**Invalid symbol name:**

**Invalid Reference syntax**

**Invalid string as opcode operand:**

**Null label references are not allowed**

**Using a defined name as a local label**

**Invalid constant symbol**

**Invalid constant expression**

**/ at beginning of line not allowed**

**NOR not supported**

**Invalid floating point register name**

**Invalid floating point constant:**

**Asm syntax error - Should start with bracket**

**Asm syntax error - register:**

**Asm syntax error - in opcode operand**

**Invalid String expression**

**Constant expression out of bounds**

**Invalid or missing opcode**

**Invalid real constant expression**

**Parenthesis are not allowed**

**Invalid Reference**

**Cannot use \_\_SELF outside a method**

**Cannot use \_\_OLDEBP outside a nested procedure**

**Invalid segment override expression**

**Strings not allowed as constants**

**Switching sections is not allowed in an assembler block**

**Invalid global definition**

**Line separator expected**

**Invalid local common definition**

**Invalid global common definition**

**assembler code not returned to text**

**invalid opcode size**

**Invalid character: <**

**Invalid character: >**

**Unsupported opcode**

**Invalid suffix for intel assembler**

**Extended not supported in this mode**

**Comp not supported in this mode**

**Invalid Operand:**

**Override operator not supported**

### **C.13.3 m68k specific errors.**

**Increment and Decrement mode not allowed together** You are trying to use dec/inc mode together.

**Invalid Register list in movem/fmovem** The register list is invalid. Normally a range of registers should be separated by - and individual registers should be separated by a slash.

**Invalid Register list for opcode**

**68020+ mode required to assemble**

## Appendix D

# Run-time errors

Applications generated by Free Pascal might generate run-time errors when certain abnormal conditions are detected in the application. This appendix lists the possible run-time errors and gives information on why they might be produced.

- 1 Invalid function number** An invalid operating system call was attempted.
- 2 File not found** Reported when trying to erase, rename or open a non-existent file.
- 3 Path not found** Reported by the directory handling routines when a path does not exist or is invalid. Also reported when trying to access a non-existent file.
- 4 Too many open files** The maximum number of files currently opened by your process has been reached. Certain operating systems limit the number of files which can be opened concurrently, and this error can occur when this limit has been reached.
- 5 File access denied** Permission to access the file is denied. This error might be caused by one of several reasons:
- Trying to open for writing a file which is read-only, or which is actually a directory.
  - File is currently locked or used by another process.
  - Trying to create a new file, or directory while a file or directory of the same name already exists.
  - Trying to read from a file which was opened in write-only mode.
  - Trying to write from a file which was opened in read-only mode.
  - Trying to remove a directory or file while it is not possible.
  - No permission to access the file or directory.
- 6 Invalid file handle** If this happens, the file variable you are using is trashed; it indicates that your memory is corrupted.
- 12 Invalid file access code** Reported when a reset or rewrite is called with an invalid `FileMode` value.
- 15 Invalid drive number** The number given to the `GetDir` or `ChDir` function specifies a non-existent disk.
- 16 Cannot remove current directory** Reported when trying to remove the currently active directory.



- 204 Invalid pointer operation** You will get this if you call `Dispose` or `FreeMem` with an invalid pointer (notably, `Nil`).
- 205 Floating point overflow** You are trying to use or produce real numbers that are too large.
- 206 Floating point underflow** You are trying to use or produce real numbers that are too small.
- 207 Invalid floating point operation** Can occur if you try to calculate the square root or logarithm of a negative number.
- 210 Object not initialized** When compiled with range checking on, a program will report this error if you call a virtual method without having called its object's constructor.
- 211 Call to abstract method** Your program tried to execute an abstract virtual method. Abstract methods should be overridden, and the overriding method should be called.
- 212 Stream registration error** This occurs when an invalid type is registered in the objects unit.
- 213 Collection index out of range** You are trying to access a collection item with an invalid index (objects unit).
- 214 Collection overflow error** The collection has reached its maximal size, and you are trying to add another element (objects unit).
- 215 Arithmetic overflow error** This error is reported when the result of an arithmetic operation is outside of its supported range. Contrary to Turbo Pascal, this error is only reported for 32-bit or 64-bit arithmetic overflows. This is due to the fact that everything is converted to 32-bit or 64-bit before doing the actual arithmetic operation.
- 216 General Protection fault** The application tried to access invalid memory space. This can be caused by several problems:
1. Dereferencing a `nil` pointer.
  2. Trying to access memory which is out of bounds (for example, calling `move` with an invalid length).
- 217 Unhandled exception occurred** An exception occurred, and there was no exception handler present. The `sysutils` unit installs a default exception handler which catches all exceptions and exits gracefully.
- 219 Invalid typecast** Thrown when an invalid typecast is attempted on a class using the `as` operator. This error is also thrown when an object or class is typecast to an invalid class or object and a virtual method of that class or object is called. This last error is only detected if the `-CR` compiler option is used.
- 222 Variant dispatch error** No dispatch method to call from variant.
- 223 Variant array create** The variant array creation failed. Usually when there is not enough memory.
- 224 Variant is not an array** This error occurs when a variant array operation is attempted on a variant which is not an array.
- 225 Var Array Bounds check error** This error occurs when a variant array index is out of bounds.
- 227 Assertion failed error** An assertion failed, and no `AssertErrorProc` procedural variable was installed.
- 229 Safecall error check** This error occurs if a safecall check fails, and no handler routine is available.

- 231 Exception stack corrupted** This error occurs when the exception object is retrieved and none is available.
- 232 Threads not supported** Thread management relies on a separate driver on some operating systems (notably, Unixes). The unit with this driver needs to be specified on the uses clause of the program, preferably as the first unit (`cthreads` on unix).

## Appendix E

### A sample gdb.ini file

Here you have a sample `gdb.ini` file listing, which gives better results when using `gdb`. Under LINUX you should put this in a `.gdbinit` file in your home directory or the current directory.

```
set print demangle off
set gnutarget auto
set verbose on
set complaints 1000
dir ./rtl/dosv2
set language c++
set print vtbl on
set print object on
set print sym on
set print pretty on
disp /i $eip

define pst
set $pos=&$arg0
set $strlen = {byte}$pos
print {char}&$arg0.st@($strlen+1)
end

document pst
    Print out a Pascal string
end
```

## Appendix F

# Options and settings

In table (F.1) a summary of available boolean compiler directives and the corresponding command line options are listed. Other directives and the corresponding options are shown in table (F.2). For more information about the command-line options, see chapter 5, page 24. For more information about the directives, see the [Programmer's Guide](#).

Table F.1: Boolean Options and directives

| Short     | long                      | Opt | Explanation                    |
|-----------|---------------------------|-----|--------------------------------|
| \$A [+/-] | \$ALIGN [ON/OFF]          |     | Data alignment                 |
| \$B [+/-] | \$BOOLEVAL [ON/OFF]       |     | Boolean evaluation mode        |
| \$C [+/-] | \$ASSERTIONS [ON/OFF]     | -Sa | Include assertions             |
| \$D [+/-] | \$DEBUGINFO [ON/OFF]      | -g  | Include debug info             |
| \$E [+/-] |                           |     | Coprocessor emulation          |
| \$F [+/-] |                           |     | Far or near function (ignored) |
| \$G [+/-] |                           |     | Generate 80286 code (ignored)  |
|           | \$GOTO [ON/OFF]           | -Sg | Support GOTO and Label         |
|           | \$HINTS [ON/OFF]          | -vh | Show hints                     |
| \$H [+/-] | \$LONGSTRINGS [ON/OFF]    | -Sh | Use ansistrings                |
| \$I [+/-] | \$IOCHECKS [ON/OFF]       | -Ci | Check I/O operation result     |
|           | \$INLINE [ON/OFF]         | -Si | Allow inline code              |
| \$L [+/-] | \$LOCALSYMBOLS [ON/OFF]   |     | Local symbol information       |
| \$M [+/-] | \$TYPEINFO [ON/OFF]       |     | Generate RTTI for classes      |
|           | \$MMX [ON/OFF]            |     | Intel MMX support              |
| \$N [+/-] |                           |     | Floating point support         |
|           | \$NOTES [ON/OFF]          | -vn | Emit notes                     |
| \$O [+/-] |                           |     | Support overlays (ignored)     |
| \$P [+/-] | \$OPENSTRINGS [ON/OFF]    |     | Support open strings           |
| \$Q [+/-] | \$OVERFLOWCHECKS [ON/OFF] | -Co | Overflow checking              |
| \$R [+/-] | \$RANGECHECKS [ON/OFF]    | -Cr | Range checks                   |
| \$S [+/-] |                           | -Ct | Stack checks                   |
|           | \$SMARTLINK [ON/OFF]      | -CX | Use smartlinking               |
|           | \$STATIC [ON/OFF]         | -St | Allow use of static            |
| \$T [+/-] | \$TYPEDADDRESS [ON/OFF]   |     | Typed addresses                |

Table F.2: Options and directives

| Short        | long           | Opt | Explanation                    |
|--------------|----------------|-----|--------------------------------|
|              | \$APPTYPE      | -W  | Application type (Win32/OS2)   |
|              | \$ASMMODE      | -R  | Assembler reader mode          |
|              | \$DEFINE       | -d  | Define symbol                  |
|              | \$DESCRIPTION  |     | Set program description        |
|              | \$ELSE         |     | Conditional compilation switch |
|              | \$ENDIF        |     | Conditional compilation end    |
|              | \$FATAL        |     | Report fatal error             |
|              | \$HINT         |     | Emit hint message              |
| \$I file     | \$INCLUDE      |     | Include file or literal text   |
|              | \$IF           |     | Conditional compilation start  |
|              | \$IFDEF NAME   |     | Conditional compilation start  |
|              | \$IFNDEF       |     | Conditional compilation start  |
|              | \$IFOPT        |     | Conditional compilation start  |
|              | \$INCLUDEPATH  | -Fi | Set include path               |
|              | \$INFO         |     | Emit information message       |
| \$L file     | \$LINK         |     | Link object file               |
|              | \$LIBRARYPATH  | -Fl | Set library path               |
|              | \$LINKLIB name |     | Link library                   |
| \$M MIN, MAX | \$MEMORY       |     | Set memory sizes               |
|              | \$MACRO        | -Sm | Allow use of macros            |
|              | \$MESSAGE      |     | Emit message                   |
|              | \$MODE         |     | Set compatibility mode         |
|              | \$NOTE         |     | Emit note message              |
|              | \$OBJECTPATH   | -Fo | Set object path                |
|              | \$OUTPUT       | -A  | Set output format              |
|              | \$PACKENUM     |     | Enumeration type size          |
|              | \$PACKRECORDS  |     | Record element alignment       |
|              | \$SATURATION   |     | Saturation (ignored)           |
|              | \$STOP         |     | Stop compilation               |
|              | \$UNDEF        | -u  | Undefine symbol                |

## Appendix G

# Getting the latest sources or installers

Free Pascal is under continuous development. From time to time, a new set of installers with what are considered stable sources are made: these are the releases. They can be downloaded from the Free Pascal website. The downloads usually contain the sources from which the release is made.

If for some reason, a newer set of files is needed - for instance, because certain bugs that prevent a program from functioning correctly have been fixed, it is possible to download the latest source files and recompile them.

Note that the latest sources may or may not compile: sometimes things get broken, and then the downloaded sources are useless. For the fixes branches (mentioned below) the sources should always compile, so it may be best to use these only.

There are 3 ways to get the latest version.

### G.1 Download via Subversion

All Free Pascal sources are in subversion, and can be downloaded anonymously from the Subversion server. With a suitable Subversion client, the following locations can be used:

```
http://svn.freepascal.org/svn/fpc/trunk/
```

This repository contains the latest sources of the compiler, RTL and packages. This is the active development branch.

The documentation and all examples from the documentation are in the following repository

```
http://svn.freepascal.org/svn/fpcdocs/trunk/
```

All the files needed to make a release can be retrieved from

```
http://svn.freepascal.org/svn/fpcbuild/trunk/
```

This repository contains external links to the other 2 repositories, and contains all scripts, demos and other files needed to construct a new release of Free Pascal.

Free Pascal maintains a fixes branch, which is used to create new releases after a major version change. The branches are located in

```
http://svn.freepascal.org/svn/fpc/branches/fpc_X_Y
```

