0764e4a0-0

COLLABORATORS

	TITLE : 0764e4a0-0	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

0764e4a0-0

1.1 FPL Compiler Environment Documentation

```
                              FPL Compiler Environment Documentation
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        FPL is (C) by FrexxWare 1992-1996. All rights reserved.

 This software has been written during a couple of hundred hours of our
spare time. To enable us to keep spending this amount of time and efforts
on products for your pleasure, please study the manual before asking
simple questions about trivial matters.


                Legal Status

                About this manual

                What is FPL Compiling

                Compiling Phases

                Compiled Format

                Requirements

                Installation

                Limitations

                File Naming Standards

                Things to Consider

                Using the Compiler
                  Preprocessor Instructions


                Source / Compiling the Compiler

                Authors of All This
```

Future

1.2 Legal Status

```
                FPL - A shared library interpreting/compiled script langauge.
 Copyright (C) 1992-1996 by FrexxWare. All rights reserved.
 Authors: Daniel Stenberg and Kjell Ericson.
                See separate section.
                 This program is free software; you may redistribute for non ↩
                    commercial
purposes only. Commercial programs must have a written permission from the
author to use FPL. FPL is *NOT* public domain! Any provided source code is
only for reference and for assurance  that users should be able to compile FPL
on any operating system he/she wants to use it in!

 You may not change, resource, patch files or in any way reverse engineer
anything in the FPL package.

 This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE.
```

1.3 About this Manual

Writing docs have always been one of the drawbacks with programming if you ask me, and therefore I never take more time than necessary to write the manuals for the projects I'm involved in. This is no exception.

I've written this manual with the help of Heddley (thanks lots for this lovely piece of program, Edd!), lots of loud music and by thinking of the users of all this that might not be able to use this if I don't produce some decriptive blurb about it.

If there is anything you miss, don't understand or think is fuzzy described somewhere in this manual, you're probably right. Please mail me about any kind of such occurance and I'll improve.

And besides, you always have the source to refer to if you're really eager to know any details not revieled in this manual! =)

1.4 What is Compiling

Compiling in general means that the source text is interpreted and converted into a format more easily understood and faster executed. Normally, compilers compile ASCII text files into machine code binary programs, using CPU instructions. The FPL Compiler is not like that, it compiles the standard ASCII text file into a FPL improved instruction set format, that is more easily understood and quicker read by the FPL interpreter engine.

Tests have proven that a general FPL program very similar or identical to the

Demo.FPL file included in the FPL distribution package, executes up to four times faster after it has been compiled (varying a little between computer systems too).

FPL-intensive programs (in which most or a big part of the execution time is occupied by the FPL engine) are of course the programs that'll get the biggest benefits from the compiling.

1.5 Compiling Phases

The compiling is done in several different phases. This is generally not important to you as a user, with exception for the preprocessor phase.

The compiling always (if not disabled through a command line parameter) starts with "preprocessing" the source file. Preprocessing means that all #-instructions (like #define, #ifdef, #include, #else, etc) are resolved. See the Preprocessor Instructions section for detailed information about those.

The next phase is the syntax-check-phase. The entire source file is now scanned again and the syntax/usage is controlled and piped further into the next phase. Any sever misuse of the language will be discovered in this phase and reported to the terminal.

The actual compiling phase converts the previous source format into the new one. This phase will also contain all sorts of optimizing in the future.

1.6 Compiled Format

Compiled files include information that might be of interest for you to know about. It is among other things:

* Lowest FPL version required to run the program.
* Version number of the compiler that generated the program.
* File name of the original FPL program source.
* Little/bin endian information. The compiled program does only work on machines with the same endian status that is stored in the compiled file. If you don't know what this means and don't intend to move compiled programs from one platform to another, just ignore this!

Optionally [debug] info:

* All external references gathered at one place. If this hunk is present, all these names will be checked on load-time, to make sure that no function in the file referes to a non-existing function/varible.
* Line number of the source file. To easier trace bugs back to the original source file.

One of the main gains with this gathered information is that a compiled

program can never get executed by a too old FPL version.

With this new, compiled custom format, we have introduced a free-form way to improve things in FPL in the future. By allowing the compiled format to change, we can improve compiling and add optimizing stages as long as we can program them.

1.7 Requirements

It might require some time from you, humble user, but I've disregarded that simple matter in the following section! ;)

The FPL Compiler Environment doesn't require much. Hardly anything actually! I just think it should be pointed out here that the FPL Compiler is of no use whatsoever without you using one or more programs that run(s) FPL programs. The 'FPLc' executable program requires the 'cpp' program to be put in the system path before run, that's all!

NOTE: the compiler is still designed to load the entire source file into memory before compiling is done, why the memory amount in your system might limit the size of FPL programs possible to compile!

1.8 Installation

Make sure the 'cpp' executable is in the system path before FPLc is run, or use the 'CPP' command line option to specify the complete path to that or a similar preprocessing program (any C-compiler's preprocessor is likely to work pretty smoothly together with FPLc).

1.9 Limitations

Some critical uses of FPL that is supported, but not supported in C, is no longer suppored when FPLCompiling. The most obvious one is continuation of string constants without the use of a backslash character (\), as in:

```
    string WorkedBefore = "line one
            ----->         line two";
```

that will cause the preprocessor to go crazy, instead use:

```
    string WorksNow = "line one\
            ----->     line two";
```

1.10 File Naming Standards

With this compiler, we have decided to set a file naming standard. We have decided that the default extensions used by any program that deals with FPL programs should be

.FPL - for uncompiled, interpreted regular ASCII text FPL sources

.FPC - for compiled, custom format FPL programs

Any other can in most cases very well be used, but these mentioned are the ones we recommend and those *we* will use for most of our work. The FPL interpreter will also feature mechanisms in which the .FPL/.FPC extentions will be required to enable them. That is i.e for the moment an automatic run-the-compiled-version-of-a-FPL-program-if-there-is-one-newer-than-the-FPL-source function.

1.11 Things to Consider

* The FPL Compiler is much more 'observant' than the interpreter. Some of the minor mistakes you've done in your programs and that have always worked when interpreting your FPL programs may cause the compiler to spit. Be aware of the fact that the interpreter never syntax check anything it doesn't interpret. Rubbish in between functions are never discovered.

* Typing mistakes such as spelling variable or functions names slightly different than they're actually named are generally *not* discovered by the compiler, but it will merely suppose it means accessing an out-of-file declaration of that variable/function. Use the VERBOSE command line flag for displaying external references to trap such kind of mistakes.

* Exported/trans-file variables and functions are much slower to access than local versions. Use local variables and functions as far as you can.

1.12 Using the Compiler

The FPL Compiler is only available from a shell prompt. [Amiga] Future versions may be WB-runnable.

Quick n' Easy
~~~~~~~~~~~~~
fplc <file>

More Detailed
~~~~~~~~~~~~~
Usage: FPLc [options] <file(s)>

Where available options are as follows:

COMMENTNEST - Allow nested comments in source files.

CPP <path> - Alter the name of the preprocessor program.

```
DEBUG <flag>  - Include debug info in the input. Flags available are:
                   LINE   - source file line number info

FILES         - The rest of the command line is treated as file names.

FILE          - The next argument is a file name.

NOVERSION     - Don't display compiler version information.

NOCPP         - Run the compiler without preprocessing the input
                file(s).

OUTPUT <name> - Output name or directory. To specify a directory, end
                the name with a '/' (or ':' on Amiga).

PPOPTS <opts> - Set preprocessor-specific options (passed on to the
                preprocessor invoke).

VERBOSE <flag> - Produce verbose report on the compiled code.
                <flag> defines which information:
                   PASS1  - Output from Pass1
                   PASS2  - Output from Pass2
                   PASS3  - Output from Pass3
                   FINAL  - Final result
                   FULL   - Full output
                You may be requested to run the compiler with this
                option if you have filed us a bug report!
```

The order of options and file names are not important.

1.13 Source / Compiling the Compiler

Included in this package (or if not, available on request from any of us authors) is the complete source code required to build the complete compiler, preprocessor and optimizer.

To compile it, it should be a case of selecting which makefile that suits your needs/equipment best and run 'make -f <makefile>'. If you experience compiling errors/warnings and fix them for a particular OS/compiler/machine, I would very much like to get my hands on those! If you can't fix them yourself, ask us, we might help you if it isn't too much and we aren't under too heavy work-load.

1.14 Authors of All This

FPL is mainly written by Daniel Stenberg. I started 1992 on writing the interpreter and has since then continously updated and developed it. FPL has been successfully compiled and run under at least 10 differen OSes, and more are very likely to be able to run it if anyone ever wanna port it...

This compiler development would never have become reality without the extensive help from Kjell Ericson and the work he put into the project. He

wrote the "3rd stage" of the compiler and is still kind of "responsible" for it (developing, bug fixing, etc).

Linus Nielsen helped me out on the somewhat fuzzy area of expression optimizing. "1+A+1" actually becomes "2+A" now! ;)

Anyone out there feels like giving us a hand in this project? Just send us a line, you don't even have to be an Amiga developer since all FPL with compiler is very much multi-platform.

Report all bugs you find and cannot fix yourself. Allthough we can't promise we'll fix them, we just as well might do it and in order to do that we have to know about them!

We are available at:

Daniel Stenberg
  Snail   Ankdammsgatan 36, S-171 43 Solna, Sweden
  Email   Daniel.Stenberg@sth.frontec.se
  FidoNet 2:201/328
  IRC   Bagder in #FrexxEd/#Amiga (Efnet)

Kjell Ericson
  Email   kjer@netcom.se
  FidoNet 2:201/328

1.15 Future of the FPL compiler

What the future of the FPL Compiler Environment will look like is very much up to you as users of this product to decide!

If you use this and have any kind of idea of what this product should have, do, support or what it lacks or shouldn't do, let us know. We're dying to get input and feedback. Below is just a quick bunch of ideas that have passed my mind...

The FPL Compiler Environment might get:

* Full source level debugger (breakpoints, view variables, stack trace)

* Better optimizer (peep hole, flow analyzer)

* Profiling possibilities (where does the program spend most time)

* Better way to specify external functions ('extern' declarations)

* Better output custom format (to improve execution speed yet more)