

07699640-0

Daniel Stenberg

COLLABORATORS

	<i>TITLE :</i> 07699640-0		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Daniel Stenberg	March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	07699640-0	1
1.1	FrexxEd main documentation	1
1.2	FrexxEd documentation index	1
1.3	Introduction	2
1.4	About the manuals	3
1.5	About the project	3
1.6	Bug reports	6
1.7	Distribution	8
1.8	File tree details	9
1.9	How to reach us	10
1.10	Ideas	11
1.11	Installation	12
1.12	Registration	12
1.13	Cripples in FrexxEd	13
1.14	Requirements	14
1.15	Starting FrexxEd	14
1.16	Support	15
1.17	Thanks	17
1.18	To do	18
1.19	Warranty	19
1.20	Block concepts	20
1.21	Character set	21
1.22	Controlling the display	23
1.23	Erasing text	24
1.24	File handling	24
1.25	GUI/Workbench	27
1.26	Inserting text	28
1.27	Point location	29
1.28	Running commands by name	29
1.29	Screen organization	30

1.30 Search and replace	32
1.31 Undoing changes	33
1.32 List gadget	34
1.33 Using multiple buffers	35
1.34 View concepts	36
1.35 Change font	37
1.36 Long lines	38
1.37 Customizing and Configuration overview	38
1.38 Change settings	38
1.39 Change startup environment	45
1.40 Change the FACT	45
1.41 Change action on keys	46
1.42 Change icons	47
1.43 Change the menu setup	47
1.44 Change mouse button actions	48
1.45 Patch internal functions	48
1.46 Programming FrexxEd: survey	51
1.47 Change keymap	52
1.48 Introduction to FPL	52
1.49 Programming FrexxEd	53
1.50 Storing FPL programs	53
1.51 Documenting FPL programs	53
1.52 Global symbols in FrexxEd	55
1.53 Listing functions	55
1.54 Startup functions	55
1.55 ARexx and FPL	56
1.56 Exceptions	56
1.57 AllEvents	57
1.58 AutoSave	57
1.59 FPL_Error	58
1.60 GetFile	58
1.61 IconDrop	58
1.62 ResizeWindow	59
1.63 SameName	59
1.64 SliderDrag	59
1.65 BufferKill	59

Chapter 1

07699640-0

1.1 FrexxEd main documentation

```
#####                                #####
##                                ##
##### ## ### ##### ## ## ## ## #####
## ##### ## ## ## ## ## ## ## ##
## ## ##### ## ## ## ## ## ## ##
## ## ## ## ## ## ## ## ## ##
## ## ## ## ## ## ## ## ## ##
## ## ##### ## ## ## ## #####
```

(C) by FrexxWare 1992-1994

Latest update: 25.10.94, for version 0.97B

An advanced, highly customizable, extensible, real-time, zero limitation, fully programmable, function driven full screen display editor.

FrexxEd is crippled SHAREWARE software.

~~~~~CONTINUE~~~~~

### 1.2 FrexxEd documentation index

#### \* INTRODUCTION

Introduction  
About~the~manuals  
About~the~project  
Bug~reports  
Distribution  
File~tree~details  
How~to~reach~us  
Ideas  
Installation  
Registration  
Requirements

#### \* GENERAL EDITING CONCEPTS

Block~concepts  
Character~set  
Controlling~the~display  
Erasing~text  
File~handling  
GUI/Workbench  
Inserting~text  
List~gadget  
Long~lines  
Point~location  
Running~commands~by~name

|                  |                        |
|------------------|------------------------|
| Starting~FrexxEd | Screen~organization    |
| Support          | Search~and~replace     |
| Thanks           | Undoing~changes        |
| Todo             | Using~multiple~buffers |
| Warranty         | View~concepts          |

## \* CUSTOMIZING & CONFIGURATING      \* PROGRAMMING FREXXED

|                             |                           |
|-----------------------------|---------------------------|
| Overview                    | Survey                    |
| Change~settings             | Introduction~to~FPL       |
| Change~fonts                | Programming~FrexxEd       |
| Change~startup~environment  | Storing~FPL~programs      |
| Change~the~FACT             | Documenting~FPL~programs  |
| Change~actions~on~keys      | Global~symbols~in~FrexxEd |
| Change~the~menu~setup       | Listing~functions         |
| Change~mouse~button~actions | Startup~functions         |
| Change~keymap               | ARexx~and~FPL             |
| Change~icons                | Exceptions                |
| Patch~internal~functions    | Function~reference        |

## 1.3 Introduction

FrexxEd is more than an editor that provides a set of functions and features. It's more like a shell, holding a text and enabling the user to perform whatever he/she wants to perform on it.

The intention of FrexxEd has been to create an editor which any user should be able to control to do and behave exactly as he/she thinks is the "right" way. If you want a requester querying whether you really should kill a changed buffer before actually doing so, it should be able to do, as well as \*not\* doing that querying. If anyone would like to set up an editor for editing EBCDIC text files, why shouldn't this be possible?!? If you would like the representation of standard ASCII characters to look different, we want it to be possible!

One of the main missions of making FrexxEd has been to make everything dynamic and to insert the ability to change every single behaviour. FrexxEd is entirely function driven, and all functions can be replaced by the user, e.g. the function that outputs 'a' when the a- key is pressed or the function that places the cursor when the mousebutton is pressed. All internal functions can be altered by the ability to hook them - make one of your own functions to get called before the actual function is called.

The most frequent question when using FrexxEd shouldn't be "WHAT is possible?" or "is XXXX possible?" but "HOW do I make XXX possible?".

FrexxEd is ShareWare. The evaluation copy features annoying~requesters and a few removed functions. You may not use this editor for testing a longer period than two or three weeks without paying the shareware fee. Registered users will get a keyfile (that will improve the functioning and remove the attention requesters) which will allow the user to use FrexxEd v1, including future revisions, without any further payment. Also, FrexxEd v2 will be cheaper for already registered users. We believe in showing the best sides of FrexxEd in the evaluation copy so that you know for sure what FrexxEd is, so therefore

there are only minor cripples inserted in FrexxEd. Don't let that fact encourage you to use it without paying, show us that you value small cripplings as it gives you better evaluation possibilities!

Register FrexxEd by sending 200 SEK, your name and address to us. Use the register form that comes along in this package. See~the~registration~section.

## 1.4 About the manuals

There should be sufficient information in this manual to allow every user to fully understand the features of FrexxEd, but if it isn't enough, do let me know! Just as the executable program, the documentation has to be developed.

For FPL knowledge and FPL programming, refer to FPLuser.guide in the fpl.library package. As FrexxEd concerns, everything that has to do with the FPL programming is described in the 'Programming~FrexxEd' chapter.

A lot of other software, trademarks and copyrighted stuff is mentioned in this manual. We do not claim to have written anything but the FrexxEd files (excluding regtools.library). For further information about any of the other softwares mentioned, ask anyone on the Net. They'll know.

To read the most frequently asked questions, consult the FrexxEd.FAQ document.

This manual is written by Daniel Stenberg. I have also found many inspiration sources in several other docs...

If we get enough requests for a printed manual, we will consider distributing one at a fair price.

## 1.5 About the project

FrexxEd is pronounced as if we were mixing the two words Fred and sex, using the first sound of Fred and the last of sex, then add the suffix "ed".

FrexxEd was developed on our A500s with three megabytes until late September 1992 when we bought ourselves one A3000 each, featuring many more megabytes. (420 kilobytes of source code took a while to compile on an A500...)

We used SAS/C 5.10 until the 6.0 was released. The 6.x with Mungwall, Enforcer and SegTracker is a superior development environment.

We started beta (alpha?) testing FrexxEd in the middle of May 1992.

The editor is coded in C with parts in assembler for efficiency.

What is FrexxEd?  
=====

FrexxEd is an advanced, highly customizable, extensible, real-time, zero limitation, fully programmable, function driven full screen display editor (not

---

a word processor) for editing text files (even though it's possible to edit any kind of file).

We say that FrexxEd is a "display" editor because normally the text being edited is visible on the screen and is updated automatically as you type your commands.

We call FrexxEd advanced because it provides facilities that go beyond simple insertion and deletion.

"Customizable" means that you can change the definitions of FrexxEd commands in many ways. For example, if you don't want FrexxEd to query if you kill a modified buffer, you can simply tell it so. Another sort of customization is rearrangement of the command set. For example, if you prefer the four basic cursor motion commands (up, down, left and right) on keys in a diamond pattern on the keyboard, you can have it.

"Extensible" and "fully programmable" means that you can go beyond simple customization and write entirely new commands (programs in the FPL language). FrexxEd is an "on-line extensible" system, which means that it is divided into many functions that call each other, any of which can be redefined in the middle of an editing session. Any part of FrexxEd can be replaced without making a separate copy of all of FrexxEd. Many of the editing commands of FrexxEd are written in FPL already; the exceptions could have been written in FPL but are written in C for improved efficiency. Although only a programmer can write an extension, anybody can use it when it's done.

We call it a "real-time" editor because the display is updated very frequently, usually after each character or pair of characters you type. This minimizes the amount of information you must keep in mind as you edit. (The term 'real-time' is, according to some, not used in its right sense here, but I think you all get my point!)

"Zero limitation" means that there are hardly no limits in amount or size in FrexxEd. Your amount of primary memory is the biggest limitation.

Every keystroke in FrexxEd invokes a function. Most keystrokes invoke the 'Output()' command which inserts the string/character stored in your AmigaDOS keymap for that key, but there is no real limit to what can be done with merely a simple keystroke. If FrexxEd cannot already do it, it can be programmed by the user to do it.

FrexxEd is not an every man text editor. It's for people with a large customizable need, brains and more than a 512KB or 1MB floppy system.

FrexxEd is ShareWare, coded with the intension to give the world a superb editor to everyone for a low price.

Who made FrexxEd?

=====

FrexxEd is written, designed, programmed, tested, organized, engineered, invented, mixed, composed and debugged by Daniel Stenberg and Kjell Ericson with additional concepts and design by Björn Stenberg.

We had help from Linus Nielsen who have coded certain parts of the huge project.

---



A lot of people have contributed in one way or another. The most important and those we remember are mentioned in the 'thanks' section.

We started our computer career on the Commodore 64 with coding demos in the year of 1987 and in July 1988 we gathered a few friends and started the group Horizon. We achieved a lot of success and popularity on the 64 for a few years and eventually we took the step upwards to the Amiga. Our only Amiga demo released (coded by Linus Nielsen and Jörgen Gustafsson) is called "Virtual Intelligence" and got very good reviews.

We produce our software together under the FrexxWare label.

FrexxEd is the first product of this dignity and size from FrexxWare. In the future, we will probably, and hopefully, bring more Frexx quality stuff to the public; everything from very large projects down to small but useful utilities.

We all work as professional programmers in different companies.

We also run a BBS.

Why make FrexxEd?  
=====

The work with this originally begun, in the late autumn 1991, because I got tired of the large amount of editors on the Amiga that lack many useful functions, have too many bugs (and then especially our favorite one) and not the least because it is an interesting programming project. Kjell was not hard to convince that this is a project worthy putting some time in.

It has also been a very good way to learn the Amiga system. Originally we didn't know much about how to make anything in this editor, but we have dig in manuals, searched through books, asked people and guessed the answers to the big secrets of Intuition, Exec, RawKeyConvert() and all our other "friends" that cooperate in FrexxEd.

God only knows when we'll find our work complete and finished.

Why use FrexxEd?  
=====

We give you almost every feature found in any other editor on the market, and if you by some reason would like to extend or modify the collection of commands, we give you (through FPL) an easy way to do so.

We give you thousands of hours of programming and development, hundreds and yet hundreds of hours of beta testing, nearly a megabyte of source code compiled into hundreds of kilobytes executable.

We give you an editor which is built upon the concepts discussed by dozens of real (programming) people and designed to be an easy and comfortable editing environment for much and plenty editing, which has resulted in several concepts and ideas that haven't earlier been seen in other editors.

We give you FrexxEd and more than 250 kilobytes of detailed documentation for a

---

very fair price.

## 1.6 Bug reports

Sometimes you will encounter a bug in FrexxEd. Although we cannot promise we can or will fix the bug (we might not even agree it's a bug!), we want to hear about any bugs you encounter in case we do want to fix them.

To make it possible for us to fix a bug, you have to report it. In order to do so you must know how to recognize one and how to report it.

What is a bug?

=====

If FrexxEd executes an illegal instruction, or dies with an operating system error message that indicates a problem in the program (as opposed to something like "disk full"), then it's most certainly a bug, or you're running the wrong version of FrexxEd (like: FrexxEd030 on a vanilla 68000 Amiga).

If FrexxEd updates the display in a way that does not correspond to what is in the buffer, then it is certainly a bug. If a command seems to do the wrong thing but the problem corrects itself if you run 'RedrawScreen()', it is a case of incorrect display updating.

Taking forever to complete a command can be a bug, but you must make certain that it was really FrexxEd's fault. Some commands simply take a long time. If the input was such that you KNOW it should have been processed quickly, report it as a bug. If you don't know whether the command should take a long time, find out by looking in the manual or by asking for assistance.

If a command you are familiar with causes a FrexxEd error message in a case where its usual definition ought to be reasonable, it is probably a bug.

If a command does the wrong thing, that is a bug. But be sure you know for certain what it ought to have done. If you aren't familiar with the command, or don't know for certain how the command is supposed to work, then it might actually be working right. Rather than jumping to conclusions, show the problem to someone who knows for certain.

Since AmigaDOS lacks the beauty called memory protection, memory allocated by FrexxEd might just as well get rendered by another process than FrexxEd itself! Therefore you must be sure that there is no other program multitasking together with FrexxEd that causes the bug. Start up FrexxEd with as few of those background processes as possible and see if the suspected bug is still around.

Finally, a command's intended definition may not be best for editing with. This is a very important sort of problem, but it is also a matter of judgment. Also, it is easy to come to such a conclusion out of ignorance of some of the existing features. It is probably best not to complain about such a problem until you have checked the documentation in the usual ways, feel confident that you understand it, and know for certain that what you want is not available. If you are not sure what the command is supposed to do after a careful reading of the manual, check the index and glossary for any terms that may be unclear. If you still do not understand, this indicates a bug in the manual. The manual's job is to make everything clear. It is just as important to report

documentation bugs as to report program bugs.

#### How to report a bug =====

When you decide that there is a bug, it is important to report it and to report it in a useful way. A useful bug report is an exact description of what commands you typed, starting with the shell command to run FrexxEd, until the problem occurred.

The most important principle in reporting a bug is to report FACTS, not hypotheses or categorizations. It is always easier to report the facts, but people seem to prefer to strain to posit explanations and report them instead. If the explanations are based on guesses about how FrexxEd is implemented, they will be useless; we will have to try to figure out what the facts must have been to lead to such speculations. Sometimes this is impossible. In any case, that is unnecessary and time-consuming work for us.

#### Example:

Suppose that you type 'amiga-o' opening the file ":gorp/baz.ugh", which (you know) happens to be rather large, and FrexxEd prints out 'I feel pretty today'. The best way to report the bug is with a sentence like the preceding one, because it gives all the facts and nothing but the facts.

Do not assume that the problem is due to the size of the file and say: "When I open a large file, FrexxEd prints out 'I feel pretty today'." This is what we mean with "guessing explanations". The problem is just as likely to be due to the fact that there is a 'z' in the file name. If this is so, then when we got your report, we would try out the problem with some "large file", probably with no 'z' in its name, and not find anything wrong. There is no way in the world that we could guess that we should try opening a file with a 'z' in its name.

Alternatively, the problem might be due to the fact that the file starts with exactly 25 spaces. For this reason, you should make sure that you inform us of the exact contents of any file that is needed to reproduce the bug. What if the problem only occurs when you have typed the 'PageUp()' command previously? This is why we ask you to give the exact sequence of characters you typed since starting to use FrexxEd.

Rather than saying "if I have three characters on the line," say "after I type '<ENTER> A B C <ENTER> <C-p>'," if that is the way you entered the text.

Check whether any programs you have loaded into the FPL world, including your 'FrexxEd.default' file, set any variables that may affect the functioning of FrexxEd. Also, see whether the problem happens in a freshly started FrexxEd without loading your 'Frexxed.default' file (start FrexxEd with the 'OMIT' switch.) If the problem does NOT occur then, it is essential that we know the contents of any programs that you must load into the FPL world in order to cause the problem to occur.

If the problem does depend on an init file or other FPL programs that are not part of the standard FrexxEd system, then you should make sure it is not a bug in those programs by complaining to their maintainers first. After they verify that they are using FrexxEd in a way that is supposed to work, they should report the bug.

---

If you can tell us a way to cause the problem without opening any files, please do so. This makes it much easier to debug. If you do need files, make sure you arrange for us to see their exact contents. For example, it can often matter whether there are spaces at the ends of lines, or a newline after the last line in the buffer (nothing ought to care whether the last line is terminated, but tell that to the bugs).

Amigas and AmigaDOS exist in extremely many different configurations. When reporting a suspect behavior, include information such as Amiga model, OS version, amount of memory (both chip and fast) and all other useful data such as processor, MMU and co-processors (I advise you to simply include the output made with Sysinfo or any equivalent program). Also make sure you always include the version number of the FrexxEd and fpl.library you are using.

If you got a Mungwall or Enforcer (or an equivalent debugging tool) hit, include the output.

Send bug reports to us by mail (electronic or snail), on our bulletin board or by calling us voice (most preferably by electronic mail).

Once again, we do not promise to fix the bug; but if the bug is serious, or ugly, or easy to fix, chances are we will want to.

## 1.7 Distribution

FrexxEd and all files in the FrexxEd package (except reqtools.library) are Copyright © 1992-1994 by FrexxWare and are, as stated earlier, delivered on "as is" basis without warranty of any kind.

You may copy and distribute verbatim copies of FrexxEd - the executable, the utilities and the documentations - to anyone you want in any media, provided that these files really remain unmodified (this does not include registered keyfiles, which are personal and may never be copied, given away or sold).

You may not by any cause make any changes to any of these files and if you decide to redistribute this package, make sure all files from the original package are included.

You may not by any cause take any fee when copying this, but the pure cost of the media, without special permission from the authors. I hate to see all those Freely Distributable programs sold for profit. I don't want anyone but the coders to get money for this project.

You may not include FrexxEd in any commercial product without special written permission from the authors.

You may not reuse parts of this package, nor disassemble, decompile, resource or in any way reverse engineer the programs.

FREXXED IS SHAREWARE and you must, if you are still using it after a short initial (two-three weeks) testing period, pay the shareware fee! FrexxEd has been developed (and still is) for hundreds and thousands of hours. We keep spending a significant amount of time and money in the project and you support the evolution by showing respect to our simple demands.

---

Registration also ensures your own copy of the very latest FrexxEd - the executable, documentations and future supporting programs (and other Frexx freely distributable software).

Registered users will receive a keyfile that will enable a few things that has been disabled in the evaluation copy of FrexxEd.

Register by sending 200 Swedish crowns and your name and address to us. See the 'Registration' chapter.

By showing respect to these few and simple rules, you'll secure the future updates and releases of FrexxEd.

## 1.8 File tree details

When installing FrexxEd, a large amount of files are copied and a few directories are created. Here follows some short description texts that describes what the files and directories are for:

| Name             | Purpose                                                                                                                                                                                                                                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ----             | -----                                                                                                                                                                                                                                                                                                                                    |
| Fred             | FrexxEd executable file                                                                                                                                                                                                                                                                                                                  |
| Freds            | FrexxEd frontend program, uses the ARexx program stored as REXX/FrexxEdStart.rx.                                                                                                                                                                                                                                                         |
| FrexxEd.txt      | History of changes                                                                                                                                                                                                                                                                                                                       |
| Install          | Installer script started from one of the following files with the 'install_' prefix.                                                                                                                                                                                                                                                     |
| install_XXX.sh   | <where XXX is the name of a language> Installer script invoker for the specific language when starting the installer from a shell.                                                                                                                                                                                                       |
| install_XXX.info | <where XXX is the name of a language> Installer script invoker for the specific language when starting the installer from Workbench.                                                                                                                                                                                                     |
| bin/FrexxKey     | Hotkey commodity program. By running this program, you can assign any AmigaDOS command line to any hotkey sequence. Example: 'FrexxKey HOTKEY "lamiga f" SYSTEM FrexxEd:fred' (FrexxKey is written by Daniel Stenberg and is Copyright (C) by FrexxWare 1994. Freely distributable for non-commercial purposes only.)                    |
| bin/Ftool2.lha   | FrexxTool archive. This archive contains the 'FrexxTool' program written by Linus Nielsen/Daniel Stenberg. FrexxTool opens up a window full of buttons on specified screen and performs actions when the buttons are pressed. A beautiful way to add button-activated actions to FrexxEd. Docs and examples is included in this package. |
| catalogs/#?      | If the executable 'fred' was selected to get put in FrexxEd: there will be a catalogs directory here holding those catalog files selected to get installed.<br>If you miss your native language and would want to add such a catalog to the FrexxEd package, get in touch!                                                               |

docs/ FrexxEd documentation directory.

docs/FPL.README This text describes shortly what FPL is.

docs/FPLuser.guide Amigaguide documentation copied from the FPL package. It features all details in how to program FPL. Variables, functions, syntaxes, keywords, expressions, statements, strings, comments, restrictions, features and much more.

docs/FrexxEd.FAQ Frequently Asked Questions regarding FrexxEd. This file answers to the most frequently asked questions about FrexxEd!

docs/FrexxEd.guide This amigaguide file. Holds all information about using FrexxEd in a general way. Additional information is found in the other two .guide files included here, and there are also links in this file to both of those.

docs/Functions.guide Amigaguide documentation holding the function reference. When you want to know what kind of parameters, return type or detailed description of a FrexxEd function for FPL programming, this is where you find it!

docs/Mailing.doc Short, detailed text describing how to subscribe to the FrexxEd mailing list.

docs/Register.form Fill out this form and include when you send your registration request to us.

FPL/ This directory contains all FPL programs that we include in the distribution package.

FPL/#?.FPL Runnable FPL programs.

FPL/#?.README Description files for the FPL programs with the same names but without the '.README' extensions. See the chapter Documenting~FPL~programs.

Icons/ Directory that holds the extension sensitive icon images that FrexxEd will use when creating icons for files. For details, see the File~handling chapter.

Macros/ Directory that by default holds the macros saved from within FrexxEd. (The SaveMacro() function in MacroIO.FPL)

Projects/ Directory that by default holds the project files saved from within FrexxEd. (The SaveProject() function in LoadSaveProject.FPL)

Rexx/ Directory that holds all FrexxEd's ARexx scripts.

Rexx/FrexxEdStart.rx ARexx script that can read a specified file into an already running FrexxEd, and optionally even wait until that file is killed.

## 1.9 How to reach us

For private matters/discussions/questions, drop us a private mail at our own:

The Holy Grail, +46-(0)8-6121258 (FidoNet 2:201/328).

Problems with FrexxEd, the utilities, FPL, bug reports or stuff related to that are dedicated to the public echo mail message areas or the frexxed mailing list; making it possible for everyone to share, learn and participate.

We're available by voice on the following numbers:

Daniel +46-(0)8-4705855  
Kjell +46-(0)8-870493

Email at:  
Daniel Daniel.Stenberg@sth.frontec.se

IRC talks in #FrexxEd, Daniel is known as 'Bagder'.

Old style mail address:

|                       |                        |
|-----------------------|------------------------|
| Daniel Stenberg       | Kjell Ericson          |
| Ankdammsgatan 36, 4tr | Härjedalsgatan 40, 1tr |
| S-171 43 SOLNA        | S-162 23 VÄLLINGBY     |
| SWEDEN                | SWEDEN                 |

Don't hesitate to send us your FPL contributions!

## 1.10 Ideas

The ideas of FrexxEd come from a lot of different editors for Amiga, MS-DOS, OS/2 and UNIX (but certainly not VM!) that have user configuration options, fancy functions and useful script languages.

The ones that have inspired us most are:

- \* GNU Emacs (under UNIX). From which I have stolen the FPL idea almost entirely, even though Emacs uses LISP and we use FPL... (I use(d) GNU Emacs/Epoch almost every day at my work on IBM and now on Frontec, and I just love it!)
  - \* GNU Emacs (on Amiga). Which indeed IS a neat editor but it's too much of a UNIX port and therefore doesn't use the best features of Amiga.
  - \* QEdit under MS-DOS. Having that cute little configuration program to redefine the actual .EXE file!
  - \* The Lattice Screen Editor (5.10). Having all major function keys definable. The SAS editor (6.0) even more.
  - \* Cygnus Editor v2.12. My favorite editor on amiga before FrexxEd, even though it includes some VERY annoying bugs and weird concepts. Has been updated, yes, but not enough in my opinion! Expensive.
  - \* Uedit v4.0. Which do include some kind of script language too, but the editor itself made me want to puke!
  - \* Turbotext. Which everybody I talk to speaks well about. I don't like it but have got some interesting views in how to solve different problems by watching it. (It's also a VERY expensive editor...)
  - \* Tked and other (lousy) shareware editors on the Fish-disks really give me the creeps. (Unregistered versions, I know.)
  - \* Edge, a late competitor. Seems to be able to do a lot. Commercial, with a huge price tag without enough motivation.
-

\* GoldEd. Another late competitor. No tabs, no undo, no decent block-marking. Yet people seem to like it... ?

... and many, many more editors that we've been testing through our years with computers (the majority have turned out to be more or less worthless). They've all learned us something: good editors are rare... We try to put all the best parts into one: FrexxEd.

We aren't at all alone. All since the very first day I told my friends for the first time we were coding on FrexxEd, we have received hundreds and yet hundreds of opinions about how really awesome editors should work. Thanks a \*LOT\* to everyone who has helped us with ideas and constructive criticism (like: "Ha ha ha, it BUUUUUUGS!!!").

If you have any ideas of modification of current FrexxEd function or perhaps a brand new idea, don't hesitate to contact one of us. FrexxEd is a living project and NEW IDEAS ARE ALWAYS WELCOME.

(I bet you've already asked yourself a few times where the hell we got the idea of such a name as 'FrexxEd'. The keyword 'Ed' explains itself and the word 'Frexx' originally comes from an internal joke between members of Horizon and has spread among our friends as a word for cool/good/groovy or something like that!)

## 1.11 Installation

FrexxEd comes with documentation files, registration form, FPL programs, locale catalogs, libraries and more...

Installing all this on your harddrive is done through workbench by double-clicking on the 'install\_<language>' icon (where <language> is your preferred language), and through CLI by entering: "install\_<language>.sh" in the FrexxEd directory. That installation procedure uses the Commodore-Amiga standard installation program 'installer'.

If you by some reason do not have that installer program, copy all files recursive manually to your destination directory, copy the libraries to LIBS: and add "assign FrexxED: <destination directory>" to your user-startup file.

## 1.12 Registration

FrexxEd is "Shareware". That means FrexxEd is copyrighted and that all users still using FrexxEd after an evaluation period of maximum three weeks, must pay the shareware fee and become a registered user.

The evaluation version of FrexxEd is not the complete editor. Registered users will be shipped a disk with the latest FrexxEd package and a personal keyfile that after installation improves FrexxEd and removes the few cripples inserted. There might also be other things added to the registered package that will never be included in the evaluation version.

---



Register by sending 200 SEK, 40 US\$ or 65 DM to:

|                       |    |                        |
|-----------------------|----|------------------------|
| Daniel Stenberg       | or | Kjell Ericson          |
| Ankdammsgatan 36, 4tr |    | Härjedalsgatan 40, 1tr |
| S-171 43 SOLNA        |    | S-162 23 VÄLLINGBY     |
| Sweden                |    | Sweden                 |

Fill out the registration~form included in this package and put it in the envelop you send.

The registration is considered done when we have received your money and your registration form.

Registration is only available at this cost to private persons as a single user license. Each user that wants to use FrexxEd must register. Commercial or company interests are treated separately and requires written permission.

Using your keyfile, you will be able to use FrexxEd version 1 and all its forthcoming updates as a registered user. Registered users of version 1 will get cheaper registration fees when/if version 2 eventually arrives.

Registration does not mean that we will send you all forthcoming updates. Registration means that you have the right to use all FrexxEd version 1 releases. When you register you will always get the latest version, but after that, you will have to ask us for updates or download them from your local BBS/ftp site. To get a single update on diskette, we will have to charge you for our extra expences.

Electronic mail replies is prioritized, preferred and will anyhow of course return much faster! (FidoNet: 2:201/328, email: dast@sth.frontec.se)

Functions that are not fully working without the keyfile are documented in the function reference with the CRIPPLE keyword followed by closer descriptions.

KEYFILES ARE PERSONAL! THEY MAY NOT BE RE-DISTRIBUTED, RENTED, SOLD, GIVEN AWAY, COPIED, REVERSE ENGINEERED OR IN ANY OTHER WAY LEAVE YOUR POSSESSION. IT SHOULD REMAIN AT YOUR PLACE, USED BY NONE BUT YOU.

Contacting us is the only way to optain a keyfile. No other keyfiles are valid as shareware licenses for FrexxEd.

NOTE:

Your name will be present in the keyfile, allowing us to trace the source of illegally copied keyfiles...

## 1.13 Cripples in FrexxEd

An unregistered FrexxEd features a few limitations/actions:

- \* No file longer than approximately 60KB can be loaded into FrexxEd.
- \* The export to clipboard function (StringToClip()) is missing.

\* An information requester will appear after a certain number of actions.

## 1.14 Requirements

FrexxEd requires

- \* regtools.library version 37 (For all requester handling. Do use the V38, it's wonderful! Copyright © by Nico Francois.)
- \* fpl.library version 9.1 (For all FPL-programs. Copyright © by FrexxWare.)
- \* AmigaDOS 2.04 Exec, Intuition, Dos, etc, etc...

FrexxEd could use

- \* powerpacker.library version 20 (To pack/unpack powerpacked files. Files with ".pp" extension automatically get packed when saved and all packed files get unpacked when loaded if powerpacker.library is present. Copyright © by Nico Francois.)
- \* xpkmaster.library version 1 (To pack/unpack xpk files. Copyright © by U. Dominik Mueller and Bryan Ford.)
- \* xpkXXXX.library version ? (Special algorithm xpk-libraries! Copyright © by their authors.)

FrexxEd hardly works on systems featuring 512KB, and less are entirely out of the question!

FrexxEd can be run on systems without hard drives, but extensive use of its powers can not be done without fast access to stored data = hard drive.

## 1.15 Starting FrexxEd

After you have installed FrexxEd, make sure your system path includes the FrexxEd executable directory.

Type "FrexxEd<enter>" on the shell command line or double click on the FrexxEd icon to start FrexxEd.

FrexxEd accepts a lot of command line/tooltype options.

When used from shell:

```
FrexxEd [BACKDROP] [COPYWB] [DIRECTORY <dir>] [INIT <file>] [OMIT] [PORTNAME]
[PRIO <num>] [SCREEN] [STARTUPFILE <file>] [WINDOW] [?] [FILE <file>]
[ [file1] [file2] [...] ]
```

When used from Workbench they are all entered as regular tooltypes.

The options are as follows:

|             |                                                                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------------------------------|
| BACKDROP    | Open FrexxEd in a backdrop window!                                                                                              |
| COPYWB      | Open FrexxEd with the same height, width, font and other species of the workbench screen. Overrides the init file.              |
| DIRECTORY   | Default directory.                                                                                                              |
| EXECUTE     | Executes the specified file as a FPL program after the startup procedure.                                                       |
| FILE        | The following parameters are file patterns. If you are using more keywords than 'FILE', they must be written to the left of it. |
| INIT        | Makes FrexxEd use the specified file name instead of the default init file.                                                     |
| OMIT        | Ignores searching for, and executing, any init file. Makes the startup a little bit faster.                                     |
| PORTNAME    | Set the ARexx port name of FrexxEd. If the given name isn't unique, FREXXED.<num> will be used.                                 |
| PRIO        | Makes FrexxEd run with given priority.                                                                                          |
| PUBSCREEN   | Makes FrexxEd open its screen as a public screen using the specified name. Default name is the same as the ARexx port name.     |
| SCREEN      | Open FrexxEd as a screen. Overrides the init file.                                                                              |
| STARTUPFILE | Makes FrexxEd use the specified startup file instead of the one specified in the init file.                                     |
| WINDOW      | Open FrexxEd as a window. Overrides the init file.                                                                              |
| ?           | A short text is presented describing the usage shortly.                                                                         |

FrexxEd also support wildcard file openings, which means that if you start FrexxEd with e.g. "FrexxEd #?.c", all files in the current directory with the suffix ".c" will be loaded into memory in one buffer each.

If you'd like to edit a file with the same name as one of the keywords, you have to write FILE <file name>.

## 1.16 Support

Bulletin Board System  
~~~~~

The FrexxEd support BBS (The Holy Grail) is a 24 hour, dual line, 1.6 Gigabyte bulletin board on which you can get in touch with the authors.

Problems, questions, discussions, thoughts, dreams, ideas, complaints, visions or whatever you want us to know about or want us to tell you about, are all

having especially dedicated public mail areas. The FrexxEd mail area is distributed as a standard FidoNet echo mail area and is available for everyone interested (R20_FREXXED)! FrexxEd is not always as easy to use as other editors on the market. We will be there with the answers to your questions. Hopefully that fidonet talk will go international.

The latest executables, news, docs and utilities are available for download. There are areas containing FPL routines wherein you can upload your own FPL creations.

Mailing list

~~~~~

For people outside of Sweden (but Swedes are welcome too!), we'll a keep mailing list for FrexxEd discussions and bug reporting.

You can subscribe or unsubscribe to the 'frexxedm' mailing list by sending email to -> frexxedm-request@aobh.xs4all.nl. <-.

The commands understood by the ListSERV program are:

#### HELP

Lists this file. This is also sent whenever a message to ListSERV is received from which no valid command could be parsed.

ADD [address] frexxedm

DELETE [address] frexxedm

Adds or deletes the given address to or from the list specified. Mail is sent to the address given to confirm the add or delete operation. If you omit the 'address' the command will assume the mailbox that is in the From: line of the message. Note that SUBSCRIBE is a synonym for ADD; UNSUBSCRIBE for DELETE.

A command must be the first word on each line in the message. Lines which do not start with a command word result in this file to be sent. If no commands were found in the entire message, that message is completely ignored. A single message may contain multiple commands; some commands may result in a separate reply (namely FAQ, HELP) while others will just write something to the logfile. You will always receive a response to any command found in the mail.

One short example:

To: frexxedm-request@aobh.xs4all.nl  
Subject:

ADD myaddress@mysite  
FAQ

will add myaddress@mysite to the "frexxedm" and send the FAQ back in return.

Please note that it IS possible to add or delete someone else's subscription to a mailing list. This facility is provided so that subscribers may alter their own subscriptions from a new or different computer account. There is therefore some potential for abuse; I have

---

chosen to limit this by mailing a confirmation notification of any addition or deletion to the address added or deleted including a copy of the message which requested the operation. At least you can find out who's doing it to you.

Note that you mail submissions to a mailing list by addressing mail to the list's name at this site (e.g., frexxedm@aobh.hacktic.nl). If you have any questions or need assistance, you can send email to postmaster@aobh.xs4all.nl for a human answer.

Yours sincerely,

Michiel Willems (Coordinator/moderator of the frexxedm mailing list)

Private mails

~~~~~

Of course we can correspond in private mails, but then other people who have the same question(s) as you do won't get any answer(s), and we'll have to answer to the same question(s) more often!

1.17 Thanks

Thanks for ideas, moral and physical support, good software and decent documentation go to:

My girlfriend...

Anja Zettinger - Who puts up with me coding all these hours. FrexxEd demands *many* hours of coding... She has also helped me to remove some of my worst abuses of the english language in this manual, and by simply existing, she constantly reminds me that life is wonderful!

...our co-programmer...

Linus Nielsen - Old friend, bugs, plenty brainstorming ideas and not the least: filehandler coding! (The file-handler won't appear until version 2 though.)

...R20_FREXXED and R20_FPL organizer..

Björn Stenberg - My brother!

...frexxedm coordinator (the mailing list)...

Michiel Willems - Dutch believer in the cause!

...some of the best beta testers in alphabetical order...

Mathias Axelsson - FPL user and AmiNet uploader.
 Stefan Boberg - A4000/A3000, lots of reports with "strange" HW.
 Pontus Hagland - Beta testing, FPL doubter (initially at least)!
 Tommy Hallgren - Protector of the Forth programming language... :)
 Magnus Hjelm - '030, concepts, ideas, complaints and reports!
 Kenneth Johansson - FPL and ARexx programming, icon drawings!
 Per-Anders Josefsson - Initial installer script.

Daniel Kahlin - Enforcing, ideas, errors, bugs and A3000!
 Per Klint - Good friend with an Amiga 4000 without MMU.
 Mathias Korsbäck - Lots of complaining! .-)
 Jonas Kvarnström - Bugfinding, Picasso II, overscan protector!
 Ola Lidholm - Never surrenders!
 Patrik Lundquist - Got into FPL really fast and beautifully!
 Lasse Mickos - Author of the future FrexxEd supported library
 EasyUI!
 Roger Nordin - A2024 (1024 x 1024), early V39, filehandler idea and
 Menu.FPL conversion to swedish!

FrexxEd wouldn't be the same without you guys!

...the locale translators...

Michiel Willems - (Dutch) Thanks for making the Menu.FPL too!
 Massimiliano De Otto - (Italian) A really good work!
 Rolf Kunisch - (German) There is still some english left!
 Robert Ramiega - (Polish) Not a standard language, but still...

...and the software producers...

Edd Dumbill - Author of Heddley which generated this very .guide
 file. Thanx for letting me use your beta versions
 and thereby finally complete this manual!!
 Nico Francois - Author of regtools.library and powerpacker.library
 which both cooperate to make FrexxEd a better
 product.
 Richard Stallman - Author of GNU Emacs which has given me much
 inspiration and from where I have borrowed a few
 parts of this documentation.
 Michael Sinz - Author of the excellent tool Enforcer.
 Carolyn Scheppner - Author of the AllocMem/FreeMem patcher Mungwall.
 SAS Institute - Creators of a real fancy compiler environment.

...and finally..

IBM - Where I used to work and from where I've got lots
 of inspiration to the documentation. (The
 InfoExplorer concept under AIX is the best online
 help system I've ever seen...)
 Frontec Railway Systems My current employer, which through its internet
 connection gives me much more contact and information
 with the rest of the [Amiga] world!

1.18 To do

FrexxEd is far from finished at this state. We develop FrexxEd constantly and we have a huge pile of ideas to implement, to code and to figure out how to code...

This is a short list of what we hope to implement sooner or later. This is only things from my fantasy of which perhaps nothing ever will become reality:

For version 2:

~~~~~

- \* Enable the function like `KeyPress("alt a");` to emulate an actual "alt a" press!
- \* Better menu building functions
- \* Mark rectangle should not be limited by the last line's length
- \* Arrows on the sliders
- \* Custom buttongadgets on the status lines
- \* Independent cursor move.
- \* Full regular expressions in search/replace
- \* Fold
- \* Timecontrolled function invokes
- \* Multi-window FrexxEd
- \* Ability to split buffers horizontally
- \* Much more internal multitasking, "multithreading". Ability to activate e.g. `about()` and when the window is still visible be able to scroll and edit the buffers.
- \* Better memory handling. Store buffers in a new/enhanced way
- \* Color-coded keywords/strings/comments/patterns... Control thorough a couple of functions how FrexxEd should visualize different things!
- \* The FrexxEd: filehandler. Access all files within FrexxEd through it. Both read and write.
- \* File notification support. Warn when a buffer's file gets changed on disk!
- \* Make resizing of a view squeeze the other smaller, instead of just let you drag the new view over the old ones.
- \* Enable better possibilitis to control the undo. Make FPL programs that do a lot of changes marked as ONE undo-step.
- \* Use a SIMPLEREFRESH window instead of the SMARTREFRESH used now to decrease the memory usage!

Following versions:

~~~~~

- * `FrexxCON:;` do a "newshell FrexxCON:" to get a shell in FrexxEd buffer!
- * Ability to edit files much larger than the internal memory by loading parts of it. (Caching of the most frequent viewed areas?) A homebrewed virtual memory system!
- * FrexxEd version using the current window, moving with escape sequences on standard out and using key input from standard in. Modem supporting kind of stuff. (FrexxEd as full screen editor in BBS programs?)
- * Ports to UNIX/Xwindows and OS/2.

1.19 Warranty

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAM AS PERMITTED BELOW, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL,

INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.20 Block concepts

There are many FrexxEd commands which operate on an arbitrary contiguous part of the current buffer. To specify the text for such a command to operate on, you set "the block start" at one end of it, and move the cursor to the other end. The text between the cursor and the mark is called "the block". You can move the cursor to adjust the boundaries of the region. It doesn't matter which one comes earlier in the text.

Once the block start has been set, it remains until it is set again. The mark remains fixed with respect to the preceding character if text is inserted or deleted in the buffer.

For example, if you wish to convert part of the buffer to all upper-case, you can use the 'UpCase' command, which operates on the text in the block. You can first go to the beginning of the text to be capitalized, type 'amiga-b' ('BlockMark') to put the block start there, move to the end, and then invoke 'UpCase'. Or, you can set the block start at the end of the text, move to the beginning, and then invoke UpCase().

Beside the common block there is also a rectangular block. The rectangular block affect rectangular areas of the text: all the characters between a certain pair of columns, in a certain range of lines. Rectangular blocks are useful with text in multicolumnar formats, such as perhaps code with comments at the right, or for changing text into or out of such formats. All commands works just as well on a rectangular block as on a common one.

Block operations such as BlockCopy() and BlockSave() affects the currently marked block if there exists one (without changing the current default "block buffer") and on the "block buffer" if there isn't.

FrexxEd supports an unlimited number of blocks.

Blocks, Buffers and Macros
=====

FrexxEd has a unique block concept. All existing blocks are simply buffers, but by having a little "block bit" set, they don't appear in regular editing. (Macros are in the same way buffers, but with another bit set!)

By having the blocks stored and visible like that, all functions that handles buffers can likewise handle blocks (and macros). Blocks can be edited like regular buffers and buffers can be inserted as if they were blocks!

Operating on the block
=====

Once you have created an active block, you can do many things to the text in it:

Note that all block operating functions also accept an optional block name which enables cutting of several blocks without losing the original ones and much more.

- * Kill it with `BlockCut()`. (It is first copied into the default "cut buffer".)
- * Save it in the default "cut buffer" with `BlockCopy()`.
- * Convert case with `UpCase()`~or~`DownCase()`.
- * Indent it with `BlockMove()`.
- * Append the block to an already existing "cut buffer" with `BlockCopyAppend()` or `BlockCutAppend()`.

Viewing/editing blocks
=====

If you're using the 'Menu.FPL' menu, supplied in the default FrexxEd package, there is an item under the 'Block' title, called something like 'Edit blocks'. Select that item, and then select one of the blocks in the list that appears, and voila! There's a block to edit...

1.21 Character set

FrexxEd's way of representing ASCII characters on screen
=====

Due to the great flexibility you get by changing system font, it's impossible for an application to have proper representations for every ASCII code to every available font.

If you take a look at other text editor they all have their own way of solving this problem. GNU Emacs uses only 7-bit ASCII and symbolizes all the rest as ^@ to ^Z (from ASCII 0 to 31) and \nnn, where nnn is an octal number (from ASCII 128 to 255). CygnusEd creates reversed @ to Z to represent 0 to 31 and runs the rest as usual. Some editors just hope that the system surrounding shows the letters in a comfortable way (which it usually doesn't!).

FrexxEd solves the problem by having it customizable! By using a FrexxEd ASCII Convert Table (FACT), which among other things defines what to view for each ASCII code.

Each ASCII code holds a string and a number of different flags. The flags tell FrexxEd which kind of character the ASCII code is. The flags are not exclusive but can be combined. Several functions act according to the flag settings of the different characters.

The FACT is easy changeable and editable using the `FACT()` function:

```
FACT(char, tags...);
```

'char' is the ASCII number of the character you'd like to change the definition

of (-1=End Of File and -2 = ContinuedLine).

The tags are as following:

~~~~~

- 'S', "string"      - The output string. See the string format description below.
  - '(', 'c'            - The character is a left parenthesis that should match the character 'c'.
  - ')', 'c'            - The character is a right parenthesis that should match the character 'c'.
  - 'W'                - The character is a word symbol (abcdEFGH).
  - ' '                - The character is a white space (space, tab).
  - '!'                - The character is a symbol (!"#%\$%).
  - 'N'                - The character genererates a newline. (This must only be modified before any buffers are loaded into FrexxEd.) ASCII code 10 usually does this. Not combinable with the TAB flag. The output string for that character will be printed before the newline action to enable "visual end of line".
  - 'T'                - The character genererates a tab step (as ASCII code 9 usually does). Not combinable with the 'N' tag. A TAB character will make the text to continue at the next tab stop. The space to the next tab stop will be filled up with the text of the output string.
  - 'U', 'c'            - The character is an uppercase with the matching lowercase 'c'.
  - 'L', 'c'            - The character is a lowercase with the matching uppercase 'c'.
  - 'E', 'X'            - Erase (clear) any of the flags 'S()W !NTUL' or all with a '-'.
- If 'S' is cleared, the default string will be set.

Examples:

```
/* Make a reversed EndOfFile character */
FACT(-1, 'S', "#REnd of file!");

/* 'a' has only word-flag */
FACT('a', 'W');

/* Tab is white-space, performs tab and outputs space */
FACT('\t', ' ', 'T', 'S', " ");

/* To make tab output "Banana" instead */
FACT('\t', ' ', 'T', 'S', "Banana");
```

```
/* Parentheses are set like this */
FACT('<', '!', '(', '>');
```

String format:  
=====

Zero length strings are supported. When the cursor is passing a zero length string, it will make a visual stop.

The string specified might contain characters not defined in the font. The appropriate "not-existent-in-the-font-character" will then appear instead of the wanted character, there is nothing we can do about it! If you don't want the character, change the FACT.

String controllers:

"#R" turns on reversed string for the following characters

"#U" turns on underline

"#B" turns on bold

"#I" turns on italic

"#r" turns off reverse

"#u" turns off underline

"#b" turns off bold

"#i" turns off italic

"#N" or "#n" resets to normal text.

(BOLD and UNDERLINE should not be combined, that goes for ITALIC with any other flag too.)

## 1.22 Controlling the display

Since only part of a large buffer fits in the window, FrexxEd tries to show the part that is likely to be interesting. The display control commands allow you to specify which part of the text you want to see.

If a buffer contains text that is too large to fit entirely within a window that is displaying the buffer, FrexxEd shows a contiguous section of the text. The section shown always contains the point.

"Scrolling" means moving text up or down in the window so that different parts of the text are visible. Scrolling forward means that text moves up, and new text appears at the bottom. Scrolling backward moves text down and new text appears at the top.

Scrolling happens automatically if you move point past the bottom or top

margins of the window. You can also explicitly request scrolling with the commands in this section or by dragging the knob of the vertical scrollbar.

```
`shift cursor down'  
    Scroll forward (a windowful number of lines) (PageDown()).  
  
`shift cursor up'  
    Scroll backward (PageUp()).
```

## 1.23 Erasing text

```
`backspace'  
    Delete the character before the cursor (Backspace()).  
  
`delete'  
    Delete the character after the cursor (Delete()).  
  
`amiga shift k'  
    Kill to the end of the line (DeleteEol()).  
  
`amiga k'  
    Kill a line at a time (DeleteLine()).  
  
`control delete'  
    Kill forward to the end of the next word (DeleteWord()).  
  
`control backspace'  
    Kill back to the beginning of the previous word  
    (BackspaceWord()).
```

The backspace key deletes the character before the cursor, another key, 'delete', deletes the character after the cursor, causing the rest of the text on the line to shift left. If 'delete' is typed at the end of a line, that line and the next line are joined together.

## 1.24 File handling

The basic unit of stored data on Amiga is the "file". To edit a file, you must tell FrexxEd to examine the file and prepare a buffer containing a copy of the file's text. This is called "loading" the file. Editing commands apply directly to text in the buffer; that is, to the copy inside FrexxEd. Your changes appear in the file itself only when you "save" the buffer back into the file.

Most FrexxEd commands that operate on a file require you to specify the file name. (Saving is an exception; the buffer knows which file name to use for that.) File names are specified using the file requester to make it easier to specify long file names and paths.

Each buffer has a default directory, normally the same as the directory where FrexxEd was started from. When FrexxEd reads a file name, if you do not specify a directory, the default directory is used. If you specify a directory in a relative fashion, with a name that does not start with a device name and a

---

colon (or without device name), it is interpreted with respect to the default directory.

For example, if you start FrexxEd standing in the directory named `'dh0:codingx/c/src'` and loads the file `'foo'`, which does not specify a directory, it is short for `'dh0:codingx/x/src/foo'`. `'//junk'` would stand for `'dh0:codingx/junk'`. `'new/foo'` would stand for the filename `'dh0:codingx/c/src/new/foo'`.

#### Loading files =====

"Loading" a file means copying its contents into FrexxEd where you can edit them. FrexxEd replaces the contents in the current buffer when you load a file. If more than one file was selected, the following files will be put in newly created buffers. We say that FrexxEd has loaded the file that it was created to hold. FrexxEd constructs the buffer name from the file name. If more than one buffer have the same name, FrexxEd will number them. For example, a file named `'/usr/rms/FrexxEd.tex'` would get a buffer named `'FrexxEd.tex'`. If there already is a buffer with that name, a unique name is constructed by appending `'(2)'`, `'(3)'`, and so on, using the lowest number that makes a name that is not already in use.

Each window's status line shows the name of the buffer that is being displayed in that window, so you can always tell what buffer you are editing.

The changes you make with FrexxEd are made in the FrexxEd buffer. They do not take effect in the file that you loaded, or any place permanent, until you "save" the buffer. Saving the buffer means that FrexxEd writes the current contents of the buffer into its loaded file.

If a buffer contains changes that have not been saved, the buffer is said to be "modified". This is important because it implies that some changes will be lost if the buffer is not saved. The status line displays whether the buffer is modified.

To load a file, use the command `Load()`. (With the name of the file you wish to load as argument).

If the specified file does not exist and could not be created, or cannot be read, then an error results. The error message is printed in the status line, and includes the file name which FrexxEd was trying to load.

What if you want to create a file? Just load it. FrexxEd prints `'(New File)'` in the status line but in other respects behaves as if you had loaded an existing empty file. If you make any changes and save them, the file is created.

If you load a nonexistent file unintentionally (because you typed the wrong file name), just make a new try and load the file you wanted.

If the file you specify is actually a directory, load will fail and display an error message in the status line.

Specifying a wildcard as a filename is quite all right. All files matching the pattern will be loaded.

---

Buffers names will always achieve the same case as the file has on disk when using load.

#### Saving files =====

"Saving" a buffer in FrexxEd means writing its contents back into the file that was loaded in the buffer. The entire path and filename will be echoed in status line when the buffer is saved.

``amiga w'`

Write the current buffer in its loaded file (Save()).

``amiga W'`

Save the current buffer in a specified file, and record that file as the one loaded in the buffer (SaveAs()).

After saving is finished, FrexxEd prints a message such as

Wrote df0:utils/foo/gnu.tasks

If the option 'save\_icon' is enabled for the saved buffer, FrexxEd will write an icon for that file. Read further below.

If FrexxEd is about to save a file and sees that the date of the latest version on disk does not match what FrexxEd last read or wrote, FrexxEd generates an exception that can be caught, notifying you about this fact. That is because it probably indicates a problem caused by simultaneous editing and requires your immediate attention.

#### Icons =====

When FrexxEd saves a buffer with the 'save\_icon' option enabled, it first checks that there is no icon currently stored for the saved file. If it is, FrexxEd won't do anything about it.

If there wasn't any previous icon, FrexxEd will check the extension (the text to the right of the rightmost dot '.') of the file name, and look in the directory "FrexxEd:icons/" for an icon that matches "<extension>.info". If such a file is found, `_that_` will be copied and used. If no extension icon is used in the file name, `".none.info"` will be used.

If FrexxEd fails to load such an extension icon, `".default.info"` will be used. If that also fails, the internal icon will be used.

'save\_icon' can be set to 'parent', which means that FrexxEd will only create an icon for the file if the file's parent directory has an icon! It was added to FrexxEd to enable transparently having directories with files without icons and some with icons. FrexxEd checks the parent dir to find out the situation in this case! The idea to this came from Roger Nordin.

The internal icon images (both the default text icon and the AppIcon image) are drawn by Kenneth Johansson! Thank you \*very\* much for them, dude!!

## Compression =====

FrexxEd supports the compression supported by the custom libraries 'xpk' and 'powerpacker'. To be able to use the compressions, you must have the proper libraries installed in LIBS: when FrexxEd is started.

If you have the libraries installed in LIBS: and load a packed file, FrexxEd will unpack the file when loading if the 'unpack' setting is enabled. Packed files get their 'pack\_type' setting set to the used type.

If the 'pack\_type' is "PP20" when a file is saved, it will be powerpacked before storage. All other 'pack\_type's will use regular XPK packing algorithms.

Loading packed files without having the libraries required installed, will make FrexxEd load the file just as it is stored on disk; most probably only a mess!

## Example: ~~~~~

We would like to pack a recently edited file with the XPK packer algorithm called NUKE. Then we make the right buffer the current one, select the menu item "Customizing->Settings->Locals" and enter "NUKE" in the 'pack\_type' field, press OK and then save the buffer as usual.

## Encryption =====

FrexxEd supports the encryptions supported by the custom library 'xpk'. To be able to use the encryption/decryption features, you must have the proper libraries installed in LIBS: at startup.

Use the local variable 'pack\_type' combined with writing your chosen password in the 'password' field to write a file in encrypted form. Reading such a file will force up a requester, prompting for password. Failing to enter the proper password will make FrexxEd fail to load the file.

Loading an encrypted file without the 'unpack' info variable set will make FrexxEd load and view the binary encrypted data.

## 1.25 GUI/Workbench

As an editor running in a Graphical User Interface (GUI) environment, FrexxEd simply has to support a number of different things regarding the workbench.

- \* FrexxEd is indeed startable from workbench and will read all files sent to it (when project files e.g. have specified FrexxEd as their default tool).
- \* FrexxEd accepts tooltypes with the same names and types as from the CLI command line.
- \* FrexxEd can be opened as an AppWindow, which means that it recognizes all

icons dropped in the FrexxEd window. The files are as default included in the current buffer at the current position. If FrexxEd fails in opening it's window as an AppWindow it will retry every time its window is reactivated.

- \* FrexxEd can open an AppIcon, which means that all icons dropped on FrexxEd's AppIcon will be included in the current buffer at the current position. If Workbench wasn't opened when FrexxEd was first started, a new try will be done each time the FrexxEd window is reactivated. AppIcon does serve its purpose best when using FrexxEd on another screen than Workbench.
- \* FrexxEd uses locale.library if present. If your selected language is among those we have catalogs for, you might get a great deal of all texts in FrexxEd using your own native language. If FrexxEd has no support for your native language and you would like to make it have it, get in touch!
- \* FrexxEd supports clipboard device communication. Cut or copy a block and then export it makes it available for the rest of the system. Using import block imports the system clipboard to the current blockbuffer.
- \* FrexxEd supports mouse block marking, vertical slider dragging (with instant window updates) and mouse positioning of the cursor.
- \* FrexxEd can be set to use a custom keymap.
- \* FrexxEd uses two different custom fonts. All user interface parts is fully font sensitive.
- \* FrexxEd allows different icons to be used when saving buffers with different extensions.

## 1.26 Inserting text

To insert printing characters into the text you are editing, just type them. This inserts the character into the buffer at the cursor (that is, at "point"). The cursor moves forward. Any characters after the cursor move forward too. If the text in the buffer is 'FOOBAR', with point before the 'B', then if you type 'XX', you get 'FOOXXBAR', with point still before the 'B'.

To "delete" text you have just inserted, use backspace. Backspace deletes the character BEFORE the cursor (not the one that the cursor is on top of or under; that is the character AFTER the cursor). The cursor and all characters after it move backwards. Therefore, if you type a printing character and then type backspace, they cancel out.

To end a line and start typing a new one, press enter. This inserts a newline character in the buffer. If point is in the middle of a line, enter splits the line. Typing backspace when the cursor is at the beginning of a line rubs out the newline before the line, thus joining the line with the preceding line.

Customization information: backspace runs the command named 'Backspace();' enter runs the command 'Output("\n")', and self-inserting printing characters run the command 'Output(char)', which inserts whatever character was typed to invoke it.

---



If you prefer to have text characters replace (overwrite) existing text rather than shove it to the right, you can enable overwrite mode (by setting 'insert\_mode' to Off).

## 1.27 Point location

To do more than insert characters, you have to know how to move point. There's always possible to move your mouse and click where you want to the cursor to be or use one of the following commands for moving it. (The default key-sequence first and the function invoked within parentheses.)

```
'cursor right'
  Move forward one character (CursorRight()).

'cursor left'
  Move backward one character (CursorLeft()).

'control cursor right'
  Move forward one word (CursorRightWord()).

'control cursor left'
  Move backward one word (CursorLeftWord()).

'cursor down'
  Move down one line, vertically (CursorDown()). This command attempts
  to keep the horizontal position unchanged, so if you start in
  the middle of one line, you end in the middle of the next.
  When on the last line of text, you won't come any further.

'cursor up'
  Move up one line, vertically (CursorUp()).

'amiga j'
  Request a number and move cursor to that line. Line 1 is the
  beginning of the buffer (GotoLine()).
  If you specify two numbers, the cursor move to the column the
  second number specify.
```

## 1.28 Running commands by name

The FrexxEd commands that are used often or that must be quick to type are bound to keys - short sequences of characters - for convenient use. Other FrexxEd commands that do not need to be brief are not bound to keys; to run them, you must refer to them by name.

A command name is, by convention, made up of one or more words, separated by writing each word with the first character in uppercase; for example, PageDown() or DeleteLine(). The use of English words makes the command name easier to remember than a key made up of obscure characters, even though it is more characters to type. Any command can be run by name, even if it is also runnable by keys.

---

The way to run a command by name is to start with 'shift escape' (invoking the function `Prompt()`), type the command in FPL syntax, and finish it with enter.

Double-clicking on a function name will make FrexxEd try to run that function with `"();"` added to the end of it. If that function accepts invokes without parameters, it will run OK, otherwise an error message will be brought to you!

All functions (even functions created by you through FPL) are possible to activate from the prompt (except the `Prompt()` command itself!).

Simply write the command line in FPL syntax and it'll be interpreted when you press enter.

`Prompt()` uses the `list~requester` with all available functions in a list ready to get clicked!

\* SEE the `FPLuser.guide` in the `fpl.library` package

## 1.29 Screen organization

FrexxEd divides the screen into several areas, each of which contains its own types of information. The largest area, of course, is the one in which you usually see the text you are editing.

When you are using FrexxEd, the screen is divided into a number of "views". Initially there is one text view occupying all but the last line. The text view can be subdivided vertically into multiple text views, each of which can be used for a different file. The view the cursor is in is the "current view" in which editing takes place. The other windows are just for reference unless you select one of them. (TECH NOTE: future versions of FrexxEd will have the ability to put single chosen buffers in their own `_real_` windows.)

Each view's last line is a status line describing what is going on in that window. Its primary purpose is to indicate what buffer is being displayed above it in the window; and whether the buffer's text has been changed on or not.

```
Point
=====
```

When FrexxEd is running, the cursor shows the location at which editing commands will take effect. This location is called "point". Other commands move point through the text, so that you can edit at different places in it.

While the cursor appears to point AT a character, point should be thought of as BETWEEN two characters; it points BEFORE the character that the cursor appears on top of. Sometimes people speak of "the cursor" when they mean "point", or speak of commands that move point as "cursor motion" commands.

If you are editing several files in FrexxEd, each file has its own point location(s). A file that is not being displayed remembers where point is so that it can be seen when you look at that file again.

When there are multiple text views, each view has its own point location. The cursor shows the location of point in the selected window. This also is how you

can tell which window is selected. If the same buffer appears in more than one window, point can be moved in each window independently.

The point location is stored in an "entry". By default each file has one entry to hold the cursor position, and if two or more views are viewing the same file they will have one entry each while visible. If a file isn't visible, only one entry is kept for that file.

To make FrexxEd able to remember several cursor positions in the same file, without viewing them, we introduce the SplitEntry() function. It will make a duplicate of the current entry and then there will exist two remembered cursor positions of that file. If a view holding such entry is removed, the cursor position will still be remembered.

Status line  
=====

Each text view's last line is a "status line" describing what is going on in that window. When there is only one text view, the status line appears at the bottom of the screen. The status line is in inverse video.

The "status line" is used to display small amounts of text for several purposes and when there's no need for texts, this gives information about the buffer being displayed in the window: the buffer's name, what position the cursor is in, whether the buffer's text has been changed, and a few other things...

If a command cannot be executed, it may print an "error message" in the status line.

Some commands print informative messages in the status line. Sometimes the message tells you what the command has done, when this is not obvious from looking at the text being edited. Sometimes the sole purpose of a command is to print a message giving you specific information. Commands that take a long time often display messages ending in '...' while they are working, and add 'done' at the end when they are finished.

The status line is build up as:

~~~~~

<File name> <Flags> Line: <num> Col: <num>

Description:

~~~~~

File name              Name of the buffer (sometimes replaced by a user message)

Flags                  Different flags:  
                       c = Buffer (C)hanged  
                       I = (I)nsert mode  
                       b = (b)lock is marked  
                       B = (B)lock is released  
                       M = (M)acro is being recorded  
                       P = (P)ack mode (the 'pack\_type' setting is not empty)

Line                    Current line number

Col                      Current column position

NOTE:

~~~~

Both "Line" and "Col" are localized strings that might differ to these in your native language.

1.30 Search and replace

Like other editors, FrexxEd has commands for searching and replacing occurrences of a string.

By invoking the requested search (as default on shift amiga s) or the requested replace (as default on amiga shift r) you'll be presented a holding up to the 100 most recent search and replace strings. Type the string you want to search for, or select on one in the list.

By clicking on the buttons in the window to the left of the search window, you can change the search/replace conditions.

Search/replace non-printable characters is done by using the C- style escape sequence standard:

| | |
|------|--|
| \a | - Alert (bell) |
| \b | - Backspace |
| \f | - Form feed (new page) |
| \n | - New-line |
| \r | - Carriage return |
| \t | - Horizontal tab |
| \v | - Vertical tab |
| \ | - Backslash |
| \xhh | - Where hh is a two digit hexadecimal value. |
| \nnn | - Where nnn is a three digit octal value. |

When replacing (without the 'no-prompt' flag), the cursor will be placed on each match and a line similar to '(y/n/q/a/g)' will be visible in the status line and FrexxEd will wait for your reaction. Press any of these keys:

| | |
|-----|---|
| 'y' | - Yes, replace this |
| 'n' | - No, don't replace this |
| 'q' | - Quit replacing (escape key is also accepted) |
| 'a' | - All (replace all without viewing the changes) |
| 'g' | - Global (replace all and view the changes) |

Search options

=====

* only Words

The string is an entire word and not any kind of substring.

* Wildcards

Wildcards in MS-DOS/UNIX Style is possible to use!

The wildcard checking is optionally line oriented.

"*" - matches a character sequence of any length
 "?" - matches one character
 "|" - logical OR.

Examples:

"hello|ninja" - searches for "hello" and "ninja".

"he*o" - matches all positions starting with "he" and ending with "o".

"A?C" - matches all substrings with a character inside an "A" and a "B".

The replace string can feature the same kind of wildcards and when a match is found, the wildcard(s) of the replace string is made the same as the one(s) that matches the search string.

EXAMPLES:

If you want to replace all strings matching "?aniel" with "?avid", strings like "Daniel" will be replaced with "David" and "raniei" with "ravid".

Replacing strings matching "D*iel" with "f*s" is perfectly ok; "Daniel" is replaced with "fans" and "Death in Kiel" is replaced by "feath in Ks".

Searching for "hello|ninja" and replacing with "ninja|hello" will make all the words "hello" in the text switch places with "ninja".

* Case sensitive

As default the search is case insensitive and with this flag enabled, FrexxEd will see a difference in e.g "hello" and "Hello".

* Forward search

Search forwards! (affects search ONLY, replace is always done forwards)

* Inside block

Search only within the currently marked block. If the block isn't marked, this flag won't be visible.

* Prompt replace (only replace flag)

Prompt for confirmation before replacing.

* Limit wildcard

If using wildcard option, this flag makes the wildcard line oriented instead of the buffer limited version.

1.31 Undoing changes

FrexxEd allows all changes made in the text of a buffer to be undone, up to a certain amount of change ('undo_max' number of bytes or 'undo_steps' number of steps). Each buffer records changes individually, and the undo command always

applies to the current buffer. Usually each editing command makes a separate entry in the undo records, but some commands such as 'replace' make many entries, and very simple commands such as self-inserting characters are often grouped to make undoing less tedious.

`'amiga u'`

Undo one batch of changes (usually, one command worth) ('Undo').

`'amiga U'`

Restart the undo session ('UndoRestart').

The command 'amiga u' is how you undo. The first time you give this command, it undoes the last change. Point moves to the text affected by the undo, so you can see what was undone.

Consecutive repetitions of the 'amiga u' commands undo earlier and earlier changes, back to the limit of what has been recorded. If all recorded changes have already been undone, the undo command prints an error message on the status line and does nothing.

Any command that somehow changes the buffer breaks the sequence of undo commands. Starting at this moment, the previous undo commands are considered ordinary changes that can themselves be undone. Thus, you can redo changes you have undone by typing 'amiga U' ('UndoRestart') and then using more undo commands.

If you notice that a buffer has been modified accidentally, the easiest way to recover is to type 'amiga u' repeatedly until the 'c' disappear from the front of the status line. At this time, all the modifications you made have been canceled. If you do not remember whether you changed the buffer deliberately, type 'amiga u' once, and when you see the last change you made undone, you will remember why you made it. If it was an accident, leave it undone. If it was deliberate, redo the change as described in the preceding paragraph.

Whenever an undo command makes the 'c' disappear from the status line, it means that the buffer contents are the same as they were when the file was last read in or saved.

Undo is switchable on/off locally for every single buffer using the setting 'undo'.

1.32 List gadget

Plenty functions featuring selection or input requesters use the same requester for ease and recognition. In some occasions it's used only to get a selection from you out of the list and in some to get a string (that also might be picked from the list), but in all cases you can control it 100% using the keyboard.

Special keystrokes:

=====

* amiga <?> Pressing amiga and a key (written in the window to the right of a check box, with an underline beneath) will toggle that check box's condition.

- * amiga v Paste the current block contents in the string gadget.
- * cursor down Select the word below the currently selected one.
 If no word was selected, the first one is chosen.
- * cursor up Select the word above the currently selected one.
- * shift cursor down Select the last word in the list.
- * shift cursor up Select the first word in the list.
- * escape The same action as clicking on the Cancel button.
- * return The same action as clicking on the Ok button.
- * tab (Only usable when using two input fields.)
 Toggle current input field.

Clicking on a word in the selection list makes it the current selected one. Double clicking is the same as choosing a word and pressing Ok.

1.33 Using multiple buffers

The text you are editing in FrexxEd resides in an object called a "buffer". Each time you open a file, a buffer is created to hold the file's text.

At any time, one and only one buffer is "selected". It is also called the "current buffer". Often we say that a command operates on "the buffer" as if there were only one; but really this means that the command operates on the current buffer (most commands do).

When FrexxEd makes multiple views, each view has a chosen buffer which is displayed there, but at any time only one of the views is selected and its chosen buffer is the selected buffer. Each view's status line displays the name of the buffer that the view is displaying.

Each buffer has a name, which can be of any length, and you can select any buffer by giving its name. Most buffers are made by opening files, and their names are derived from the files' names. But you can also create an empty buffer with any name you want. A newly started FrexxEd has a buffer named "*noname*" which can be used for evaluating FPL expressions in FrexxEd. The distinction between upper and lower case does not matters in buffer names.

Each buffer records individually what file it has opened and whether it is modified. A lot of FrexxEd settings are "local to" a particular buffer, meaning its value in that buffer can be different from the value in other buffers.

Each buffer has an entry, which keeps track of, among other things, the cursor position. All buffers start with one entry and can therefore only have one remembered cursor position (when none are being viewed). By duplicating an entry, FrexxEd offers an unlimited number of entries to the same buffer, and therefore can remember plenty cursor positions even if the buffer isn't viewed in that many views.

Selecting buffers

=====

``alt cursor up'`

Select next buffer on screen (`'NextView'`).

``alt cursor down'`

Select previous buffer on screen (`'PrevView'`).

``alt cursor left'`

Select next buffer that is not visible on screen (`'NextHidden'`).

``alt cursor right'`

Select previous buffer that is not visible on screen (`'PrevHidden'`).

``amiga g'`

Select a buffer (`PromptBuffer()` using script), or simply view all buffers that are currently in memory.

1.34 View concepts

FrexxEd can split the screen into two or many views, which can display parts of different buffers, or different parts of one buffer.

Basic view

=====

When multiple views are being displayed, each view has a FrexxEd buffer designated for display in it. The same buffer may appear in more than one view; if it does, any changes in its text are displayed in all the views where it appears. But the views showing the same buffer can show different parts of it, because each view has its own value of point.

At any time, one of the views is the "selected view"; the buffer this view is displaying is the current buffer. The cursor shows the location of point in this view. Each other view has a location of point as well, but since there is only one cursor there is no way to show where those locations are.

Commands to move point affect the value of point for the selected FrexxEd view only. They do not change the value of point in any other FrexxEd view, even one showing the same buffer. The same is true for commands such as `'NextBuf'` to change the selected buffer in the selected view; they do not affect other views at all.

Each view has its own status line, which displays the buffer name, modification status etc.

Splitting views

=====

``amiga d'`

Split the selected view into two views, one above the other. The two views each get half the height of the view that was split. Future versions of FrexxEd will include ability to split buffers horizontally too.

Using other views
=====

`'alt cursor up'`
Select previous view ('PrevView').

`'alt cursor down'`
Select next view ('NextView').

To select a different view, use `'alt cursor up/down'`. The `'previous'` buffer is the one above the current (if the current is the uppermost view, the previous is the buffer at the bottom) and the `'next'` buffer is the one below the current (if the current is the bottommost view, the next is the buffer at the top).

Deleting and rearranging views
=====

`'amiga delete'`
Get rid of the selected view ('RemoveView').

`'amiga l'`
Get rid of all views except the selected one ('MaximizeView').

`'amiga +'`
Resize the selected view ('ResizeView').

The space occupied by the deleted view is given to the view below. Once a view is deleted, its attributes are forgotten; there is no automatic way to make another view of the same shape or showing the same buffer. But the buffer continues to exist, and you can select it in any view with `'amiga g'`.

To readjust the division of space among existing views, use `'amiga +'` (it enables you to change the size of the currently selected view) or click and drag using the mouse.

1.35 Change font

In a GUI environment, the selection of the right font for the right moment is important for the user to feel comfortable. FrexxEd uses two different font for best look. One is used as `'system font'`, which is the main text edit font used in the editing area as well as in the status lines. That font must be non-proportional. The other font is used everywhere else where fonts are used: in requesters, information windows and in the menus.

The easiest way to change the fonts is to use the menu items:

"Customizing->Font->System font" for changing the first font and
"Customizing->Font->Request font" for changing the second one.

ADVANCED INFORMATION:

* See the info variables `"system_font"` and `"request_font"` in the `'change~settings'` section.

1.36 Long lines

If you add too many characters to one line, without breaking it with a return, the line will grow longer than the window is able to visualize. All characters that do not fit in the width of the screen or window do not appear at all. They remain in the buffer, temporarily invisible. There is no limit of a single line's length.

There is no way to see if a line is longer than the screen but to try to move the cursor to the end of the line, as default. There is although a `FACT` controllable char named `"ContinuedLine"` which is visualized at the right of all lines longer than the width of the window. If the line is longer than the screen is wide, the screen will scroll to the left (the amount of cursor steps `FrexxEd` will scroll is set in `'move_screen'`) when the cursor reach the column specified in `'marg_right'` (actually that number is the number of characters from the right edge).

The screen will scroll back to the right when the cursor hits the `'marg_left'` column of the screen.

1.37 Customizing and Configuration overview

Modern text editors of today's computing societies do often include features for recording a sequence of actions for later replaying. So called macros. They are often even capable of changing fonts and when looking especially at the Amiga market, they all have `ARexx` ports. Text editors of today are made to be altered by the users of them.

`FrexxEd` is a modern editor in a world of change. `FrexxEd` is indeed changeable too, in more ways than any other text editor I've ever seen in my life, including editors in operating systems like `UNIX` (`Xwindows`), `OS/9(000)`, `OS/2`, `MS-DOS` (`Windows`) and `AmigaDOS`.

We have constructed `FrexxEd` with the goal that everyone should be able to make `FrexxEd` to suit their needs and to do exactly whatever it is that you think a text editor should do. Therefore we provide baziljons of different ways to make `FrexxEd` different than default.

If you, after have read through this section, still can't think of a way to solve you problem; get in touch and we'll make it possible. If not by using the current customizing tools, we'll figure out a way to support it in the next `FrexxEd` version!

1.38 Change settings

(Remember that some of the strings mentioned in this text may not be the same using `locale.library` and a non-english language.)

`FrexxEd` provides a bunch of `"settings"` (also refered to as `"info variables"`) that are ment to be altered (or some simply read) by the user to make `FrexxEd` alter its way to do things. Just remember to save the settings to make them take effect every time you start `FrexxEd` in the future.

Available "settings" are dependent of what FPL programs that have been run since such programs can add new ones as well as alter existing ones.

The "settings" are easy controlled using the menu items, under the "Customizing" label. Some of them are global, that affect the entire FrexxEd, and some are local (buffer-specific).

To get a more detailed definition of the variables, which kind, what range they can be altered within, and similar information, refer to the Functions.doc manual and the ReadInfo() function description!

This is the full and verbose FrexxEd internal info variable reference!

| NAME | DESCRIPTION |
|---------------------|--|
| ==== | ===== |
| 'appicon' | Tells FrexxEd that it should bring up an AppIcon on the Workbench screen. AppIcons are the kind of icons that regular disk devices are. Dropping an icon on FrexxEd's AppIcon will invoke the 'IconDrop' exception. By default it will include the dropped file at the current position. |
| 'appwindow' | Tells FrexxEd that is should make its window an AppWindow. AppWindows enables Workbench users to inform FrexxEd when he/she drops icons in the FrexxEd window. Dropping an icon in FrexxEd's window will invoke the 'IconDrop' exception. By default it will include the dropped file at the current position. |
| 'arexx_port' | Each started FrexxEd will get its own unique ARexx port. This variable hold the name of this invoke's port. |
| 'auto_resize' | Tells FrexxEd if it should adjust its size after the selected font's size. If this is enabled, FrexxEd will adjust window size whenever it is resized, to not make any gap within the window. |
| 'autosave' | When enabled, this will make FrexxEd generate the 'AutoSave' exception after 'autosave_interval' number of changes has been done to a buffer. |
| 'autosave_interval' | Controls the number of changes required to generate the 'AutoSave' exception if 'autosave' is enabled. |
| 'autoscroll' | Tells intuition to let the screen scroll when the mouse pointer reaches when edge of the visible area of a screen larger than visible. |
| 'block_begin_x' | Holds the start column of the current block |
| 'block_begin_y' | Holds the start line of the current block |
| 'block_end_x' | Holds the end column of the current block |
| 'block_end_y' | Holds the end line of the current block |

| | |
|------------------|--|
| 'block_exist' | This is 0, 1 or 2 if a block is currently:
0 - not marked
1 - marked at the cursor
2 - marked and released from the cursor |
| 'block_id' | Block ID of the current block. |
| 'block_type' | Type of the currently marked block. 1 = normal, 2 = rectangle |
| 'buffers' | Holds the number of buffers in this FrexxEd. |
| 'byte_position' | The actual byte position of the current line. That means that *all* characters are read as one single byte, independent of what the characters may look like on the screen. |
| 'changes' | Holds the number of changes done to the buffer since it was last saved or loaded. |
| 'colour0' | Holds the 4 bit color value of color 0. |
| 'colour1' | Holds the 4 bit color value of color 1. |
| 'colour2' | Holds the 4 bit color value of color 2. |
| 'colour3' | Holds the 4 bit color value of color 3. |
| 'column' | The column number of the current cursor position. |
| 'comment' | File comment to the current buffer. |
| 'copy_wb' | If enabled at startup, FrexxEd will copy window/screen size, display mode, fonts and colors from the public screen it opens on. |
| 'counter' | A constantly increasing number that increases one step on every internal action of FrexxEd. Read it for whatever purpose you like. |
| 'current_screen' | The name of the screen this FrexxEd is opened on. |
| 'cursor_x' | The cursor position relative the left border of this view. |
| 'cursor_y' | The cursor position relative the upper border of this view. |
| 'default_file' | Which default file that was read initially at startup. |
| 'directory' | Default directory. FrexxEd changes to this directory when it runs. All file operations are done with the specified directory as if FrexxEd was started in that directory. An empty string make FrexxEd use the "real" startup directory. |
| 'display_id' | Which display ID the FrexxEd screen is using. |
| 'ds_Days' | The modification date of the buffer's file. |
| 'ds_Minute' | |
| 'ds_Ticks' | |

| | |
|----------------------|--|
| 'entries' | Number of existing entries in this FrexxEd. |
| 'expand_path' | How to expand the path at load time. "Off" - not at all, "Relative" - expand path names without semicolon and "All" - expand all path names. Using the "Off" version makes it possible to change current directory of FrexxEd and then store all files loaded relative in different dirs than they were loaded from. Version "Relative" enables the user to change the destination of an assign in the middle of an editing session and then saved files will use the new dir. "Off" will disable all chances of storing the file in a different dir that it was brought from. It will use the exact volume name, even changing diskette in df0: will be denied! |
| 'file_name' | The file name of the buffer. |
| 'file_number' | File number of the buffer. If more than one buffer has the same file name, they will get different file numbers. |
| 'file_path' | Path name of the buffer. |
| 'fragmentation' | Number of fragmentations of the buffer. |
| 'fragmentation_size' | Total size of the fragmentations. |
| 'full_file_name' | The full file name as FrexxEd knows it. That is, path and file name joined together the proper way. |
| 'insert_mode' | Characters that are output in the buffer should insert themselves in the text if this is enabled, otherwise they write over the existing data. |
| 'keymap' | Specify the name of your especially designed keymap here and that keymap will be used within FrexxEd. No string means that the system default is used. |
| 'language' | The language the system was using when FrexxEd was started. This string is the same as locale uses. That is, the name of the language is the locale name, the name the speakers of the language use. |
| 'line' | The line number of the current position. |
| 'line_counter' | If enabled, FrexxEd will display each line number to the left of the actual lines in the view. |
| 'line_counter_width' | When using line counter, this is the width of the line counter field that FrexxEd will use. |
| 'line_length' | The length in number of bytes of the current line. |
| 'lines' | The amount of lines in the current buffer. |
| 'macro_key' | If this is a buffer than was created by a macro recording, this variable will hold the key sequence assign to its |

execution.

- 'macro_on' If this is true, macro recording is in progress!
 - 'marg_left' These four "settings" specifies the number of cursors from each
 - 'marg_lower' edge where the screen should move. If the margin is hit,
 - 'marg_right' FrexxEd moves the screen in that direction.
 - 'marg_upper'
 - 'mouse_x' In which column of the view the mouse button was pressed.
 - 'mouse_y' In which line of the view the mouse button was pressed.
 - 'move_screen' The amount of cursor steps that FrexxEd should move the screen
at a time when any margin is hit by the cursor.
 - 'overscan' Overscan type in intuition secret code!
 - 'pack_type' This tells FrexxEd how to pack this file when it is to be
stored on disk. All files that are unpacked when loaded will
get this set to the previously used packing type. "PP20" will
force powerpacker to be used when packing, and all other
types will be up to xpkmaster.library. If the specified
packing type isn't known to xpk, any save action will fail.
Some widely used algorithms include: BLZW, NUKE, IMPL,
HUFF and more. The packing type is always a four-letter
word. Files with a ".pp" suffix is automatically
powerpacked when saved (that is using the "PP20" packing
type).

NOTE: This option does rely on third party library support.
xpkmaster.library is copyrighted by its authors, and Urban
Dominik Müller, umueller@amiga.physik.unizh.ch, is one of
the leading ones. powerpacker.library is copyrighted by
Nico Francois, nico@augfl.be.
 - 'password' If the file previously read and/or should be encrypted when
saved, this is the password that was used/should be used.
This can only be used together with a XPK pack type string
that supports encryption! See the File Handling section.
 - 'pen_info' Pen number of the information on the right half of the status
line.
 - 'popup_view' Tells FrexxEd in which way you would like new views to show
up on the screen. Should they 'replace' the current, should
they 'split' the current or should they take use of the
'full' window?
 - 'protection' The protection bits of the current buffer as one string. Each
bit has its own letter. They can be specified in any order.
The letters used are the ones that the AmigaDOS 'protect'
command uses: "RWEDSPAHH". (R)ead, (W)rite, (E)xecute,
(D)elete, (S)cript, (P)ure, (A)rchive and (H)idden.
 - 'protectionbits' The protection bits of the current buffer as one integer.
The bits are defined exactly as defined in dos/dos.h, see
-

further details under the ReadInfo() description.

- 'public_screen' The name of the public screen to open FrexxEd on. When FrexxEd already is opened, this will show you the name of the current screen FrexxEd is opened on.
 - 'real_screen_height' The "real_#?" variables holds the actual values that their names tell. Those variables without the "real_" prefix are the wished valued.
 - 'real_screen_width'
 - 'real_window_height'
 - 'real_window_width'
 - 'real_window_xpos'
 - 'real_window_ypos'
 - 'replace_buffer' The string in the 'replace buffer', that is the string that will replace the search string when a replace is performed.
 - 'request_font' The font name and size used in all requesters and the menus.
 - 'right_mbutton' If enabled, tells FrexxEd that you would like to have the right mouse button programmable. If disabled, the right mouse button will only bring up the menus.
 - 'rwd_sensitive' If enabled, tells FrexxEd that it should care about the protection bits of the files it reads and writes.
 - 'safe_save' If enabled, FrexxEd will always save buffers by first creating a unique file name and then try to rename that file to the name of the "real" file. If the file name is any kind of link, saving will fail if 'safe_save' is enabled.
 - 'save_icon' 'never' saves any icons
'always' saves icons when anything is written to disk
'parent' saves an icon if the parent directory has an icon.
See the File handling section.
 - 'screen_depth' The number of bitplanes that the FrexxEd screen uses.
 - 'screen_height' The preferred height of the FrexxEd screen.
 - 'screen_width' The preferred width of the FrexxEd screen.
 - 'search_block' The search/replace flag search/replace within block.
 - 'search_buffer' The current search string. Could also be used to read strings from the search history.
 - 'search_case' The search/replace flag search/replace case sensitive.
 - 'search_forward' The search/replace flag search/replace forward.
 - 'search_limit' The search/replace flag search/replace limit wildcards at end of line.
 - 'search_prompt' The replace flag query before replace.
-

'search_wildcard' The search/replace flag search/replace wildcard is used.

'search_word' The search/replace flag search/replace to match only words.

'shared' Number of entries to the same buffer.

'shared_number' Number of this entry among the entries to the same buffer.

'show_path' If enabled, FrexxEd will show as much as possible of the path to the files in the status lines, otherwise simply the file name part of it.

'size' The size in bytes of the current buffer.

'slider' This cycle gadget lets you select slider. You can turn it off or select left/right. When using FrexxEd in a window, this has no effect, but the slider will always be put on the right side.

'startup_file' This is the name of the FPL program to run when FrexxEd is started. Several programs can be specified by separating them with '|'.

'system_font' The font name and size of the screen/text font of FrexxEd.

'tab_size' Width of the TAB character.

'taskpri' The exec priority of this FrexxEd process.

'topline' The number of the topmost line of the view.

'type' Buffer type bitmask:
Bit 0 = standard file buffer
Bit 1 = macro buffer
Bit 2 = block buffer
Bit 3 = hidden buffer

'undo' Undo on/off for this buffer.

'undo_lines' Number of undo lines stored in the undo buffer for this buffer.

'undo_max' The maximum amount of bytes to use for undo buffers in whole FrexxEd.

'undo_memory' Amount in bytes that the undo buffer currently uses.

'undo_steps' The maximum amount of undo steps to store in one undo buffer.

'unpack' If enabled, FrexxEd will unpack loaded files whenever such a file is read and the proper library exist in LIBS:. If 'unpack' is disabled, no unpacking/decryption will be performed.

'version' The version number of the running FrexxEd. The number is built like Version * 10000 + Revision.

'version_id' The version ID string of the running FrexxEd.

'view_columns' Number of columns in the view.

'view_lines' Number of lines in the view.

'views' Number of views in the window.

'window' Select how you want FrexxEd to startup as default. 'screen', 'window' or 'backdrop'.

'window_height' The height which FrexxEd should use when it opens a window.

'window_pos' Whether to use the x and y positions of the FrexxEd opening relative the 0,0 of the host screen, or relative the visible display clip of the screen. 'Absolute' or 'Visible'.

'window_width' The width which FrexxEd should use when it opens a window.

'window_xpos' The x position of the FrexxEd opening. See 'window_pos'.

'window_ypos' The y position of the FrexxEd opening. See 'window_pos'.

ADVANCED INFORMATION:

* See the functions ReadInfo(), SetInfo() and PromptInfo() in the Functions.guide manual.

1.39 Change startup environment

As can be read more about in the "PROGRAMMING~FREXXED,~Startup~functions" chapter, FrexxEd executes none, one or more FPL programs when started. You can easily change which file(s) by selecting the menu item "Customizing->Settings->Globals" and enter the file name in the 'startup_file' field. Several names should be separated with a vertical bar '|'. FrexxEd will search for the file(s) first as specified and then in the 'FrexxEd:FPL/' directory.

1.40 Change the FACT

The FrexxEd ASCII Convert Table (FACT) is responsible for how ASCII codes are represented in the editor window, whether the character is lower or upper case and things like what kind of character it is.

Notice that the End Of File string/character is represented by ASCII character code -1, and the string/character written at the right of a line that continues past the right edge of the window is represented by -2.

There is currently no easy user interface way to change the FACT. There is only the FACT() function way.

See also:

"GENERAL~EDITING~CONCEPTS,~Character~set"
 "PROGRAMMING FREXXED, FPL function descriptions, FACT()"

1.41 Change action on keys

All users using a text editor has different backgrounds. Depending on your background, you have different opinions, likings and habits when it comes to which key-sequence that does what you like. You might want the delete line function to exist on the MS-DOS style "control y", or the amiga style "amiga k" or perhaps the UNIX style "control k".

FrexxEd provides an easy interface to change the placement of different functions, with AssignKey(). Running a small FPL program holding the following line:

```
AssignKey("DeleteLine();", "control y");
```

will make FrexxEd run the "DeleteLine();" program whenever "control y" is pressed! By typing something else instead of "control y", you can decide by yourself which key sequence you want to invoke that "DeleteLine();" program. Likewise, you can change the program to hold any valid FPL program to be ran when the specified key-sequence is pressed!

Multiple key-presses are specified just by adding the keys in one long sequence. E.g, to make the key-sequence "control x" and "control c" (GNU Emacs "quit all" key sequence) close down FrexxEd, specify a line similar to:

```
AssignKey("QuitAll();", "control x control c");
```

Running simple line like:

```
AssignKey();
```

will make FrexxEd to display a requester to be filled out with the proper information (key sequence and FPL program).

A lot of keys have names recognized by FrexxEd when written inside single quotes. Naming such a key is much easier than trying to enter the output of an actual press on that key. These keys are:

| Key | Description |
|-----------|---------------------------|
| === | ===== |
| F1-F10 | The function keys |
| F11-F20 | The shifted function keys |
| Backspace | The backspace key |
| Bspc | As above |
| Del | The delete key |
| Delete | As above |
| Help | The help key |
| Up | The cursor up key |
| Down | The cursor down key |
| Right | The cursor right key |
| Left | The cursor left key |
| Escape | The escape key |
| Esc | As above |

| | |
|--------|----------------------|
| Enter | The return/enter key |
| Return | As above |
| Tab | The tabulator key |
| Space | The space key |
| Spc | As above |

* ADVANCED INFORMATION

See the AssignKey() function in the Functions.doc manual.

1.42 Change icons

When saving files, FrexxEd will examine the state of the 'save_icon' info variable, and if that is enabled/on, FrexxEd will store an icon together with the file. If there already is an icon present, nothing will be done to it.

You control which icon FrexxEd will put there, by copying your favourite icons to the FrexxEd:icons/ directory and name them as:

| | |
|--------------------|---|
| <extension>.info - | for files that have file name extensions |
| .none.info - | for files without file name extensions |
| .default.info - | for files that is saved and none of the above icons was found/matched |

If none of the icon files is found when FrexxEd is supposed to save with an icon, FrexxEd will use its own internal icon.

In the following examples, we think of the 'save_icon' option to be enabled and that no icon is already present.

EXAMPLES

1. You save a file named 'foobar.c':

FrexxEd will first try to use the file named "FrexxEd:icons/c.info" and if that doesn't exist, FrexxEd will use "FrexxEd:icons/.default.info".

2. You save a file named 'startup-sequence':

FrexxEd will first try to use the file named "FrexxEd:icons/.none.info" and if that doesn't exist, FrexxEd will use "FrexxEd:icons/.default.info".

* See also the "Filehandling->Icons" section

1.43 Change the menu setup

One of the most basic parts in the Amiga GUI is the menu and the menu items. FrexxEd includes a default menu setup, which holds some of the more important but yet basic functions of a text editor.

Changing the menu setup is yet again a simple matter of running an FPL program. A menu structure building program can look like:

```
MenuAdd("t", "Project");
    MenuAdd("i", "Clear", "Clear();", "amiga c");
    MenuAdd("i", "Open", "Open();", "amiga o");
    MenuAdd("i", "Save", "Save();", "amiga s");
    MenuAdd("i", "About", "About();", "amiga ?");
    MenuAdd("i", "Kill", "Kill();", "amiga q");
    MenuAdd("i", "Quit all", "QuitAll();", "amiga Q");

MenuAdd("t", "Customizing");
    MenuAdd("i", "Colors");
        MenuAdd("s", "Adjust", "ColorAdjust();"); /* no shortcut */
        MenuAdd("s", "Reset", "ColorReset();"); /* no shortcut */
    MenuAdd("i", "Settings");
        MenuAdd("s", "Locals", "PromptInfo(-1);");
        MenuAdd("s", "All locals", "PromptInfo(-2);");
        MenuAdd("s", "Globals", "PromptInfo(0);");
        MenuAdd("s", "Save", "SetSave();");
    MenuAdd("i", "ScreenMode", "Screenmode();");

MenuBuild();
```

See also:

"Function reference: MenuBuild"
 "Function reference: MenuAdd"

1.44 Change mouse button actions

FrexxEd is controllable with the mouse. You may mark blocks, place the cursor and select current view.

All mouse actions are of course also defineable. A mouse action is in FrexxEd read as a keypress, and therefore is the mouse-modifying done just as you modify keypresses.

The mouse actions are named "MouseLeft", "MouseLeftDrag", "MouseLeftDouble" and the same for "Right" and "Middle". The names can be used with the AssignKey() just like with keys.

This is a small example that opens a file when the left mouse button is pressed(!):

```
AssignKey("Open();", "MouseLeft");
```

See also:

"Function reference: AssignKey()"

1.45 Patch internal functions

FPL programs/scripts in FrexxEd have more than one hundred internal functions that they can call to create the effect they want. They use `Load()` to load a file, `Search()` to search for strings and so on.

There is times when you discover that one or more of those internal functions don't do the exact things that you want them to do. FrexxEd provides a method to do so: hooks!

"Hooks" are functions called before or after an internal function or exception. There can be any amount of hooks, both before and after the internal function/exception. All hooks have the ability to cancel the rest of the hooks, and if it is a "before"-hook, it can cancel the running of the function/exception it has hooked. If a hook returns a non-zero number, the following functions in the hook-chain will be cancelled.

There are also a few other times you can change the way things happen with hooks. FrexxEd generates named exception in some occasions, events that you can hook! One example of such an event is the 'IconDrop' event, which occur when an icon is dropped on FrexxEd's AppIcon or AppWindow. The event will be called, but 'IconDrop' is no function and does not exist as a function!

Hooks can be made to only get executed if a specified info variable is set to a non-zero value (if numerical) or non zero length (if strings) or the other way around, if the variable is preceeded with an exclamation mark.

The hooking is done with the functions `Hook()` or `HookPast()`. For further details of how to use it, see the function reference chapter! Notice that the hook-functions must be declared using the exact parameters as the hooked function does, or an entirely new program that can be executed.

Removing hooks is done with the function 'HookClear', which accepts three parameters: the name of the function, the name of the hook to remove and the name of the dependent variable. This function removes all hooks that matches the parameters. I.e, specifying only the hooked function will clear all hooks on that function, specifying only the name of the hook will clear all hooks in the system using that name and specifying only the name of the dependent variable will remove all hooks in the system dependent on that variable! These parameters can be combined in any way. NOTE: `HookClear()` removes hooks from both `Hook()` and `HookPast()` function calls, all matches will be cleared!

Examples

=====

Example 1. We patch 'Load()' (which can take a string as parameter) to call the function 'Load_hook()' (which we have declared). Our function must accept just a single string as a parameter. The hook is created like:

```
Hook("Load", "Load_hook");
```

Example 2. We patch the same function again, but this time we want our function to be called with the string "foobar" when it is invoked:

```
Hook("Load", "Load_hook(\"foobar\");");
```

Example 3. We look at the examle function "Save()". We decide that we don't

want any program to perform a save of the current buffer if it isn't modified:

```
export int Save2(string Filename)
{
    if(strlen(Filename))
        /* It isn't the default buffer that is about to be saved! */
        return (0);

    if(! ReadInfo("changes"))
        /* No changes has been done to the current buffer! */
        return (1); /* don't save! */
}
Hook("Save", "Save2");
```

Example 4. We want a function to be run right after every call to Output(), using the same parameters as is sent to Output():

```
HookPast("Output", "My_own_func", "");
```

Example 5. We want a function to be run right after every call to Output(), but only if the info variable 'ninja' is non-zero:

```
HookPast("Output", "My_own_func", "ninja");
```

Example 6. We want a function to be run right after every call to Output(), but only if the info variable 'ninja' *IS* zero:

```
HookPast("Output", "My_own_func", "!ninja");
```

Example 7. We want a function to be run right before every call to Output(), but only if the info variable 'ninja' *IS* zero:

```
Hook("Output", "My_own_func", "!ninja");
```

Example 8. We want to remove the hook just added with example 7:

```
HookClear("Output", "My_own_func");
```

Example 9. We want to remove all hooks in FrexxEd that depends on the "c-mode" variable:

```
HookClear("", "", "c-mode");
```

Example 10. We want to remove all hooks on the Output() function:

```
HookClear("Output");
```

Example 11. We want to call our backup function "Backup" whenever the "AutoSave" exception is generated:

```
Hook("AutoSave", "Backup");
```

Things to consider
=====

Hooks is perhaps the most powerful way to change things in FrexxEd. Not even the built-in functions are spared from the user's customizing capabilities. But

think twice before you use this function. In most cases it's not really a patch you want to do, but a clean and nice function defined by yourself. Remember that your hooks should make all kinds of FPL programs to still work if they call the function you have hooked.

Therefore:

- * Don't change the actual behaviour of a function. If you copy a function created by someone else, I think you would like to have that to work as that person coded it to work!
- * Don't patch more than necessary. Always consider coding an entire new function instead!
- * Many hooks make the function run slower. Especially important on machines with non-impressive(!) processors...
- * Make the patches depend on a info variable as often as possible. E.g, if you create a few special patches to make editing text files easier, then make them depend on the 'TextFile' variable! It also makes it lot easier to remove the patches again if wanted.
- * If you don't specify an entire FPL program syntax in the second parameter in the Hook() call, the specified function must accept the **exact** same parameters as the hooked function!

NOTE

====

This is very powerful stuff. You can manipulate a lot with this, but think about compatibility with other scripts. Have in mind that you might (sooner or later) receive other FPL programs from us or from other sources, and that you really would want them to work in your environment without too much problems with your patches, don't you?

See also:

```
"PROGRAMMING~FREXXED:~Exceptions"
"Customizing:~change~setting"
```

1.46 Programming FrexxEd: survey

This manual has earlier stated that FrexxEd is fully programmable, and function driven, but what does that mean? How does that affect us? And most important: how can we take advantage of it?

Every single keystroke invokes a function. Every menu item selection invokes a function, or in some cases a number of functions. Every mouse button press invoke a function. Anything you would like FrexxEd to do is done through a function. Most keypresses invokes the 'Output' function and outputs the string found in the keymap. The menu item "About" invokes the function named 'About'.

Most users of FrexxEd will not have to think about things like that. Most users of FrexxEd will never feel an urge to change a strange behaviour of the editor, never put the functions on other keys than the default settings or add

functions that enhances the FrexxEd environment and makes life easier to the user. But some do. To those who would like to control FrexxEd to the very limit, to those who would like to modify the editor to suit their needs and habits, to those, FrexxEd offers a highly controllable and customizable editor through the interpreting language FPL (Frexx Programming Language).

1.47 Change keymap

The system's global keymap may not always be the best to edit with. Perhaps you would like keyrepeat on the 'return' key (as another famous Amiga editor turns on automatically when you enter that editor), maybe you would like to change the position of some commonly used programming symbols or something else.

Enter the window appearing when the menu item "Customizing->Settings->Globals" is selected and change the 'keymap' field to the name of your desired keymap. If you want FrexxEd to use that on every startup, make sure you save the settings after the change.

ADVANCED INFORMATION:

See also:

"Customizing:~change~setting", PromptInfo(), SetInfo()

1.48 Introduction to FPL

FPL (Copyright © 1992-1994 by FrexxWare) is a shared library based interpreting language *very* similar to the C programming language. The library is named "fpl.library" and must be located in your LIBS: directory to enable FrexxEd to start.

FPL is an entire programming language with lots of syntax rules, variable handing, functions (defining, declaring and calling), expresions and all those other things that define a programming language. To get fully acquainted with all of that, I must refer to the 'FPLuser.guide' documentation.

This manual will describe how to use FPL in FrexxEd. Function names mentioned in here will either be within quotes ('function') or have a pair of open-close parentheses added to the function name (function()), or sometimes the both ways are applied. The mentioning of a function name is only to inform you of the name, not in any way which arguments it accepts or values it returns, unless otherwise is written.

For a complete reference on all functions that FrexxEd provides to the FPL programs, refer to the 'Functions.guide' also included in this package!

If you end up making any kind of FPL program that you think that the FrexxEd community would be able to take advantage from, or simply enjoy, don't hesitate to send~it~to~us for inclusion in future releases or special FPL packages.

1.49 Programming FrexxEd

Programming FrexxEd is very easy. FrexxEd provides a couple of functions that enables execution of specified files, buffers and/or macros:

- * `ExecuteBuffer()`
Executes the current of specified buffer.
- * `ExecuteFile()`
Executes a named file.
- * `ExecuteLater()`
Executes a string later. Later is in this case after all other actions that FrexxEd is "scheduled" to do is done; after the current function.
- * `ExecuteString()`
Executes a string.
- * `AssignKey()`
Assigns an FPL program to a certain key sequence.
- * `Prompt()`
(Placed as default on the short cut "shift escape".) Lets you enter an FPL program in a requester input field. The list view in the requester displays all currently available functions, including function written and exported by you.

You have also received FPL programs that add such functions to the menu item for fast invokes, and there you may also find functions like "ExecuteBlock" which will execute the current block buffer as an FPL program...

1.50 Storing FPL programs

Of course you can store the FPL program wherever you please, on floppies, in ram disks or in the S: directory. FrexxEd supplies by default an FPL directory ("FrexxEd:FPL/") in which all FPL programs is put that is included in the distribution package.

It is recommended that you put your FPL programs there. Whenever the `ExecuteFile()` is invoked, it searches that directory by default if the entered file name wasn't found as given.

1.51 Documenting FPL programs

Since anybody can extend FrexxEd at any time by adding functions through FPL programs, we have set a small standard of how to document the FPL programs to enable users to share any program with anybody.

Remember to separate all FPL programs as much as possible. Do not put too many different functionalities into one single file, its much better to split them (the files) up.

The standard is built up with a few keyword strings that must be included and below those, a free-form documention area. That area is although recommended to feature a few headlines.

Document file format =====

The header must start with a newline character followed by at least 20 pound signs (#). Before that, anything can be written/included. After those pound signs there should be a newline again and then should there should follow keywords and descriptions with one keyword per line in the format like: "keyword: description".

Keywords that must be included:

| | |
|-----------------|--|
| File | - Name of the file that this file documents. |
| Author | - Real and full name of the FPL programmer. |
| Email | - Electronic mail address to the FPL programmer. |
| Short | - Short description of this file. Useful for fast searches. |
| Version | - <Version number>.<Release number> |
| Date | - Common amiga-style date string dd.mm.yy |
| Local settings | - All local settings this file uses |
| Global settings | - All global settings this file uses |
| Keysequence | - All keysequences this file occupies |
| Type | - What sort of functions that this file contains:
Hook/function/key/FACT/Menu. |
| Prereq | - Which file or files that this file requires to be run
before this file can be used. |

End the header with a line like the one that began the header. Following the end of the header is a free-format description of the file.

Example file =====

```
#####
File: FooBar.FPL
Author: Daniel Stenberg
Email: dast@sth.frontec.se, FidoNet 2:201/328
Short: Fools the operator
Version: 1.0
Date: 22.4.94
Local settings: ("_foo" is hidden)
Global settings: "foobar"
Keysequence:
Type: Hook
Prereq:
#####
```

FUNCTION

Descriptive text about the functionality.

BUGS

Known bugs or weaknesses

SEE ALSO

Similar functions

1.52 Global symbols in FrexxEd

In FPL, you can declare global symbols in a program. You can even declare symbols to be of the 'export' kind, which make them accessible from any other FPL program. Ex:

```
int export batch_mode=0; /* TRUE/FALSE if in batch mode or not */
```

When this variable get cached, all FPL programs can read and write this variable. The same goes of course for functions, most effectively declared as:

```
int export MyFunction(int foobar)
{
    /* Program body */
}
```

This function will after caching also be accessible everywhere from all FPL programs.

FPL demands however that files that exports or are using global symbols (variables or functions) must be cached (held in memory) so that FPL can refer to that file if a symbol from it is accessed. FrexxEd allows only FPL programs executed from files to be cached. Programs executed with ExecuteBlock() or Prompt() will loose their global and/or exported variables as fast as that program ends. As long as you don't want to chache anything, or if you want to try out your new functions, they are prefered to be used before ExecuteFile().

1.53 Listing functions

With the function Prompt() (as default on "shift escape") you get a list of all current available functions. You may enter a valid FPL program in the requester to execute small FPL programs. All exported FPL functions will also be included in the list.

See also:

```
"Running~commands~by~name", Prompt()
```

1.54 Startup functions

At startup, FrexxEd are searching the current directory and FrexxEd:FPL/ for an FPL program called "FrexxEd.default". That file is executed and afterwards, the 'startup_file' info variable tells FrexxEd which FPL program(s) to interpret before anything else happens in the editor (after the screen/window has been opened).

The 'startup_file' program(s) is/are to be looked upon as a startup-sequence in which you can assign keys, execute FPL programs, desing menus and much, much more! If you design your own FPL programs and want them to be inserted in the

FrexxEd environment at every startup, such a file execution is perfect. The files are searched for, first in the FrexxEd:FPL/ directory and then in the system path.

When the startup program is executed, only a subset of the FPL functions are present, which greatly limitates the actions available. (The startup functions are marked with a "[*]" symbol in the function reference.)

A startup function that sets all general info variables are generated by the SetSave() call, which can be invoked with the menu item "Customizing->Settings->Save".

See also:

"Starting~FrexxEd"

1.55 ARexx and FPL

The ARexx solution of FrexxEd is as simple as it can be. All input to FrexxEd from the ARexx port is interpreted as a FPL program!

Read more in the Functions.doc manual about the functions:

- * ARexxResult()
This function lets the user set the ARexx return code and "result" string of the FrexxEd ARexx call.
- * ARexxRead()
Returns the contents of an ARexx variable.
- * ARexxSend()
Sends a string to a specified ARexx port and received the returned string.
- * ARexxSet()
Set the contents of an ARexx variable.
- * FindPort()
Find or wait for a system/ARexx port.

1.56 Exceptions

FrexxEd generates exceptions on some specific occations. Those exceptions can be trapped and be programmed to perform actions. A perfect example of this is when a user drops an icon on the FrexxEd AppIcon or AppWindow. The exception generated then is called 'IconDrop', and right after the exception, FrexxEd will do what it thinks should be done when 'IconDrop' is received.

To alter or insert the functioning of an exception, just Hook() or HookPast() the exception just as done with regular internal functions. The exception can be looked upon exactly as if it were a function. The only thing is that you can never call it. Exceptions even get parameters that you can pass to your hook.

Available exceptions are currently:

AllEvents
AutoSave
FPL_Error
GetFile
IconDrop
ResizeWindow
SameName
SliderDrag

1.57 AllEvents

NAME AllEvents

PARAMETERS

int ID - Function ID

DESCRIPTION

Generated when an event is happening. That is when any function is called within FrexxEd. This exception is invoked very often and **should* *not** be extensively hooked since that will cause a *_major_* system slowdown! This exception will get called with an integer parameter that is merely an internal function ID of the function that is to happen right after this exception. NOTE: The number should only be used in purposes that makes **no** connection between the function ID and the actual function since the IDs may very well be altered between versions or even releases of FrexxEd.

SEE ALSO

"Exceptions", "Patching~internal~functions"

1.58 AutoSave

NAME AutoSave

PARAMETERS

int BufferID - Buffer ID of the buffer that sent this exception

DESCRIPTION

Generated when 'autosave_interval' number of changes has been done in a buffer that has the 'autosave' flag enabled. The buffer ID of the buffer that this affects is sent as parameter. Nothing is done outside of a hook when this exception is called. Any actions must be done in the hook.

SEE ALSO

"Exceptions", "Patching~internal~functions"

1.59 FPL_Error

NAME FPL_Error

PARAMETERS

| | |
|----------------------|--------------------------|
| int Line - | Line number |
| string Program - | Program name |
| string Description - | Error description string |

DESCRIPTION

Generated when an FPL program error has occurred. Default action is to bring up a requester saying so!

SEE ALSO

"Exceptions", "Patching~internal~functions"

1.60 GetFile

NAME GetFile

PARAMETERS

| | |
|---------------|------------------------------------|
| string Path - | Path of the file to be loaded |
| string Name - | File name of the file to be loaded |

DESCRIPTION

Generated when a file is to be opened. The parameters sent to this function are the path name and the file name of a file ready for loading. Returning a non-zero value will abort the loading of the file. When using multiple selection or wildcard file loadings, this exception will always occur once for every selected/matching file.

SEE ALSO

"Exceptions", "Patching~internal~functions"

1.61 IconDrop

NAME IconDrop

PARAMETERS

| | |
|-------------------|------------------------------------|
| string FullName - | Full path name of the dropped file |
|-------------------|------------------------------------|

DESCRIPTION

Generated when the user dropped an icon in the FrexxEd AppWindow or AppIcon (we won't see any difference between such two invokes). This exception gets the full path name to the icon's file sent as parameter. If a hook function does not abort the exception actions, the file will be inserted at the current position in the current buffer.

SEE ALSO

"Exceptions", "Patching~internal~functions"

1.62 ResizeWindow

NAME ResizeWindow

PARAMETERS
 none

DESCRIPTION
 Generated when the window size gets changed (resizebutton, zoom gadget
 or changed through settings).

SEE ALSO
 "Exceptions", "Patching~internal~functions"

1.63 SameName

NAME SameName

PARAMETERS
 string NewFile - Full path name of the new file
 int BufferID - Already existing buffer's buffer ID

DESCRIPTION
 Generated with a file is to be opened with the same name as already
 exists as a buffer in FrexxEd. This exception sends the full file name
 as parameter. Returning a non-zero value will abort the loading of
 the file.

SEE ALSO
 "Exceptions", "Patching~internal~functions"

1.64 SliderDrag

NAME SliderDrag

PARAMETERS
 none

DESCRIPTION
 Generated when the vertical slider is dragged.

SEE ALSO
 "Exceptions", "Patching~internal~functions"

1.65 BufferKill

NAME BufferKill

PARAMETERS

int ID - BufferID

DESCRIPTION

Generated when the buffer is removed from memory. It is either killed through the Kill() function, or removed due to that FrexxEd is about to exit.

SEE ALSO

"Patching~internal~functions"
