FrexxEd

Daniel Stenberg

COLLABORATORS

	TITLE : FrexxEd		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Daniel Stenberg	January 18, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

FrexxEd

1.1 nothing

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

          (C) by FrexxWare 1992-1996

 Latest update: 23.7.96, for version 2 beta 4
```

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

1.2 FrexxEd User's Guide

1.3 Customizing & configuratiting

1.4 Programming FrexxEd

1.5 History of changes

This is a brief summary of the most recent changes done to FrexxEd:

```
2.0
===

- GetKey() knows how to read a qualifier-only.

- GetChar() and GetCursor() now take -1 for last position.

- We no longer try to use the same version number of frexxed.library as the
  whole FrexxEd release version. That is plain silly.

- Single character colouring is now supported. The
              FACT
               is now capable of
  definining background and foreground pens of all characters.

-
              Face support.
               (context sensitive colour and styles of text on screen)
  A bunch of functions have been added to support this, and FPL programs
  have been adjusted.

-
              Multi window support.
```

Lots of new functions and info variables have been added and ↩
adjusted to
fully support this new thing.

- Lots of new, improved and updated FPL programs.

1.11
====

- Updated FPL programs

- Changed mailing list address, please forget the old one! Send 'ADD ME'
  to dastmail@sth.frontec.se to subscribe!

- Fixed that keypress bug once more and hopefully better.

- Made FrexxEd unlock the screen properly when iconfied.

  There are more known bugs, yes, but this is only a stress-release to fix
  the two most annoying bugs. We still put all major efforts in making the
  version 2. It will feature color syntaxing as well as multi-windows and
  FPL compiler support.

1.6 Introduction to FrexxEd

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 what-
ever 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 therfore 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 cripples 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.

1.7 Features of FrexxEd

```
        Some of FrexxEd's features are:
        ===============================

 o Any amount of windows or screens (on any public screen)
 o Context-sensitive colouring/styling of text.
 o Any amount of views available in all windows.
 o Ability to edit the same file in all windows and views, or in none
   is selected.
 o 100% recursive undo, changeable undobuffer size and on/off toggle
 o unlimited number of buffers/blocks/macros in memory
 o unlimited line lengths and buffer sizes
 o 100% programmable, everything can be done by a script instead of
   interactive
 o unlimited number of views viewing the same buffer/file
 o *very* C-like programming language (FPL)
 o folds, hide parts of the buffers
 o mounted filehandler volume, access all buffers like with an amiga disk
 o clipboard support, export to/import from any unit
 o uses V39-style memory pools
 o V39 aware, uses V39 functions when available
 o AppIcon/AppWindow support
 o Window/Screen/Backdrop
 o file name extension aware icon usage
 o supports locale library, 7 languages are supported at the moment
 o easy macro record/edit/view/play/save/load
 o customizeable look of any ASCII character (to i.e enable the inverse
   CED-look or the \xxx Emacs-look or...)
 o programmable keyassigns, make any key do anything
 o programmable mouseactions, make any mouse action perform whatever
   you want it to
 o rectangular/column style block marks and block pastes
 o programmable menustrip, make it contain whatever you please!
 o supports XPK and PowerPacker compression/decompression and
   encryption/decryption.
 o startup-script that loads any file into a running editor
 o >500K documentation in amigaguide format
 o works just as good used with gfx hardware like Retina, GVP Spectrum,
   Piccolo, Picasso II, A2024, AGA, ECS or OCS...
 o of course useable on any screenmode
 o ARexx port, capable of reading and writing ARexx variables!
 o A *LOT* of additional scripts that perform a multitude of different
   things. From mousedragging-style blockmarking, archive-flag removal
   when saving, center line, binary editor, snake game, incremental
   search, interactive C indenting and word wrap to generic assembler
   filters, bookmarks, backup, autosave and programs that easy
   interfaces RCS...
 o Supports file comments
 o Custom keymap support
 o optional "safe save", save with temp name and then rename
 o easily exchangeable images for icon creations/usage
 o binary loading/editing/saving
 o unlimited key sequence lengths
 o ability to clone the public screen when opened as a window on such
 o font sensitive GUI
 o line number mode supported, start each line with the line number
 o ability to replace/change the workings of any internal function
   through an advanced hooking procedure
 o mailing list, WWW-site and other eletronic/mail support
 o runs smoothly with virtual memory managers like VMM
 o search/replace history
 o time invoked functions
 o true appicon iconification
```

1.8 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. As FrexxEd
concerns, everything that has to do with the FPL programming is described in
the 'Programming FrexxEd' chapter, but the complete function reference is
found in Functions.guide.

All manuals in the FrexxEd distribution are available in straight ASCII
versions, and the manuals called "FPL - User's Guide" and "FrexxEd - User's
Guide" are available in well-edited, good-looking postscript versions!

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 reqtools.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.

The manuals are written and edited by Daniel Stenberg.

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

1.9 About the FrexxEd 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 500 kilobytes of detailed "online" documentation for a very fair price.

1.10 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.

1.11 Distribution rules

FrexxEd and all files in the FrexxEd package (except reqtools. ↩ library) are
Copyright © 1992-1995 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.

1.12 FrexxEd 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           Description
 ----           -----------

 Fred           FrexxEd executable file
 Freds          FrexxEd frontend program, uses the ARexx program stored as
                Rexx/FrexxEdStart.rx.


 bin/           Binaries to use with or without FrexxEd.
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 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.)
 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.
 startfred      Executable file that uses the Rexx/FrexxEdStart.rx script to
                load files into a running FrexxEd. Useable when used as
                default tool for icons and more.
 startfred.c    Source to the above mentioned program.
 ETags.lha      ETags package. This package contains essential stuff if you
                plan to use the ETags.FPL program. I'd say this package is a
                dream to any programmer using FrexxEd!
 AGgrep         AmigaGuide grep. This program is used to generate the file
                SimpleHelp.FPL uses.
 AGgrep.doc     Short text describing 'AGgrep'.
 FPLdb          FPL debugger. Very useful when trying to get your troubling
                FPL programs to run as you want them to run!
 FPLdb.doc      Short documentation to FPLdb.


 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.
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 #?             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.
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 FPL.README     This text describes shortly what FPL is.
 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.
 FrexxEd.FAQ    Frequently Asked Questions regarding FrexxEd. This file
```

```
                answers to the most frequently asked questions about FrexxEd!
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.
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!
Mailing.doc     Short, detailed text describing how to subscribe to the
                FrexxEd mailing list.
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          Runnable FPL programs.
#?.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)

Startup/        Directory in which you should put all FPL programs that
                should get executed at startup, and all macros that should
                get defined at startup. The FPL programs must end with '.FPL'
                and the macros with '.macro'.



Rexx/           Directory that holds all FrexxEd's ARexx scripts.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FrexxEdStart.rx Script that can read a specified file into an
                already running FrexxEd, and optionally even wait until that
                file is killed.
FrexxEdToFront.rexx Brings FREXXED.1's window/screen to front.
Deiconify.rx    Deiconfies FREXXED.1.
ShowFPL.rx      Tool that simplifies the selection of which FPL programs to
                use in your startup!
```

1.13 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 voice on the following numbers:
        Daniel                        +46-(0)8-4705855
        Kjell                         +46-(0)8-870493

  Email at:
        Daniel                        Daniel.Stenberg@sth.frontec.se
        Kjell                         kjer@netcom.se
        Mailing list                  dastmail@sth.frontec.se

Old style mail address:

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

1.14 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 feel as old as the OS version it was made for... ;(

* 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!)

1.15 Installing FrexxEd

FrexxEd comes with documentation files, registration form, FPL progams, 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 prefered language), and through CLI by entering: "install_<language>.sh" in the FrexxEd directory. That installation procedure uses the Commodore-Amiga standard installtion 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.

1.16 Registrating FrexxEd

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
        Ankdammsgatan 36, 4tr
        S-171 43 SOLNA
        Sweden
```

Danish users have an optional registration address and currency! 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 version 2 and all its forthcoming updates as a registered user. Registered users of version 1 will get version 2 for free when/if it eventually arrives. Following versions will of course be cheaper to already registered users.

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: Daniel.Stenberg@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.

We reserve the right to take precautions if we find out that any of the above stated rules have been broken or ignored.

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

We reserve the right to drop development at any time, without any notice.

NOTE:

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

1.17 Danish Registration information

```
                        FrexxEd registrering i Danmark
                        ==============================
```

Det er fra FrexxEd version 1.5 blevet lidt lettere at registrere FrexxEd i Danmark. Samtidig med den sædvanlige registreringsmetode (direkte ved Daniel eller Kjell i Sverige) kan du nu nøjes med at flytte rundt på penge inden for landets grænser.

En registrering i Danmark er gyldig på lige fod med en registrering i Sverige. Den eneste forskel ligger i metoden, og dermed i, hvor besværligt det er for dig!

Det foregår på følgende måde: Penge og registreringskort holdes i Danmark. Din key-file tilsendes direkte fra Sverige - evt. via Email.

Nedenstående priser gælder kun når den svenske krone ligger i kurs 80 eller derunder! Dette dokument vil blive ændret ved hver FrexxEd version, så det reflekterer den korrekte kurs. Hvis kursen ligger over, eller hvis du er i tvivl, så skal du være velkommen til at ringe til mig (se nedenfor).

Du kan betale på een af følgende måder:

o Ved at sende en crosset check på 180 DKr sammen med registreringskortet.
  Checken skal være udstedt til mig (se nedenfor).

o Ved at indsætte 180 Dkr på min konto i Jyske Bank,
  kontonr. 7253-0004503585.
  Send en kopi af kvitteringen sammen med registreringskortet.
  Hvis du flytter pengene vha. Tele-Info eller lignende, skal du være
  opmærksom på, at du måske ikke automatisk får en kvittering!
  Vær opmærksom på, at din bank måske vil opkræve et gebyr for overførslen.
  Dette kan undgås ved personligt at indsætte pengene i en Jyske Bank.

o Ved at sende 200 DKr i sedler sammen med registreringskortet.
  Pak sedlerne ind i mørkt papir så de ikke kan ses igennem konvolutten.

```
   Bemærk, at det sikkert vil være billigere at få lavet en check!
   Jeg tager *intet* ansvar for breve der forsvinder i posten! - Overvej igen
   en check =)


 Send betaling og udfyldt registreringskort til:

 Jesper Skov
 Sallingsundvej 47, st th
 9220 Aalborg Ø


 Hvis du har spørgsmål, er du velkommen til at ringe til mig:

 98155062 (kl. 18-21)

 Email: jskov@iesd.auc.dk

                                                     /Jesper
```

1.18 Cripples in FrexxEd

```
  An unregistered FrexxEd features a few limitations/actions:

* If a file larger than approximately 60KB is loaded into FrexxEd, the
  'this is an un-register version' requester will appear!

* The export to clipboard function (StringToClip()) is missing.

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

* The window title can't be changed.
```

1.19 FrexxEd requirements

```
  FrexxEd requires
  ~~~~~~~~~~~~~~~~
 * AmigaDOS 2.04 Exec, Intuition, Dos, etc, etc...

 * fpl.library version 13 (For all FPL-programs. Copyright © by FrexxWare.)

 * reqtools.library version 37 (For all requester handling. Do use the V38,
   it's wonderful! Copyright © by Nico Francois.)

  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 only 1MB ram, 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. Around 1.5 MB of disk space is used.

1.20 Starting FrexxEd

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

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

To make start a new editing session using an already started FrexxEd, you can start the program 'startfred' that is in the FrexxEd:bin/ drawer.

FrexxEd accepts a lot of command line/tooltype options.

When used from shell:

```
Fred [BACKDROP] [COPYWB] [DIRECTORY <dir>] [DISKNAME <name>] [INIT <file>]
[OMIT] [PORTNAME] [PRIO <num>] [SCREEN] [STARTUPFILE <file>] [WINDOW] [?]
[ICONIFY] [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.
DISKNAME	Sets the name of the mounted filehandler . Use the name "off" to switch off the filehandler.
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.

ICONIFY	Makes FrexxEd start up in iconified state. As an appicon on the workbench window; doubleclick to deiconify.
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. "frontmost" will make the window pop up on the frontmost screen.
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.

When FrexxEd is started, it will first check for an enviroment variable named "FrexxEd". If such a variable exists, it will be parsed just like a command line, before the actual command line/tool types are parsed.

FrexxEd also support wildcard file openings, which means that if you start FrexxEd with e.g. "Fred #?.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>.

1.21 Loading files into a started FrexxEd

If you already have one copy of FrexxEd of memory running, you don’t have to start another one when you want to edit a file through by kind of automatic editing.

Like when your favourite program wants to use an external program to edit a file, or when you set the tooltypes of your project icons.

How
~~~
’startfred’ is the name of a program in the FrexxEd:bin/ directory which will load the file names specified into a running FrexxEd. ’startfred’ can very well be used and specified as default tool in a project icon as well as external editor in programs like Directory Opus, DiskMaster and similar.
~~~

'startfred' accepts a command line option called 'STICKY' which then makes the program wait until the file(s) loaded has been quit/killed in FrexxEd.

Thanks to Mikael Säker for making this program!

Details
~~~~~~~
'startfred' uses the ARexx program called FrexxEd:Rexx/FrexxEdStart.rx which also can be used stand-alone, but then not as i.e default tool.

'FrexxEdStart.rx' connects and uses only the FrexxEd running with ARexx port called 'FREXXED.1' - in normal cases the first copy started.

Thanks to Nicolas Dade for his extensions of the mentioned ARexx program.

1.22 FrexxEd support

WWW-page
~~~~~~~~
Get the latest FrexxEd and FPL versions, or read the FrexxEd manual online on the WWW-page at HTTP://www.lysator.liu.se/~matax/Main.html. This page is organized and taken care of by Mathias Axelsson.

Bulletin Board System
~~~~~~~~~~~~~~~~~~~~~
The
FrexxEd support BBS
(The Holy Grail) is a 24 hour, dual line, 2 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 ('FREXXED')!
FrexxEd might not always be 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 FrexxEd Mailing list by sending email to >>> dastmail@sth.frontec.se <<<.

The commands understood by the ListSERV program are:

HELP
Sends a help text back to you.

```
    ADD [address]
    SUB [address]
          Adds or deletes the given address to or from the list.
          Mail is sent to the address given to confirm the add or delete
          operation. If you replace the 'address' with ME or MYSELF, the
          command will assume the mailbox that is in the From: line  of
          the message. Note  that SUBSCRIBE is a synonym for ADD and
          UNSUBSCRIBE and DELETE for SUB.
```

A command must be the first line(s) in a message.

If no commands were found on the first line, it is considered as a valid posting to the list! A single message may contain multiple commands; some commands may result in a separate reply (namely HELP, ADD and more) 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: dastmail@sth.frontec.se
     Subject: doesn't matter at all

     ADD myaddress@mysite
     HELP
```

will add myaddress@mysite to the list and send the HELP text 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.

Note that you mail submissions to a mailing list by addressing mail to the same address (e.g., dastmail@sth.frontec.se). If you have any questions or need assistance, you can send email to Daniel.Stenberg@sth.frontec.se for a human answer.

```
    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!

1.23 Thanks to...

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: he made most of the filehandler coding!

...FREXXED and R20_FPL (fidonet message areas) organizer..

Björn Stenberg -	My brother!

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

Michiel Willems -	Dutch believer in the cause!

...our code enhancer...

Jesper Skov -	Also a very frequent bug reporter and an idea source very few can match! Many FPL programs too...

...our contributors...

Mikael Säker -	Brought us 'bin/startfred'!
Björn Reese -	For the 'bin/Barfly2Msg' and Assemble.FPL.
Nicolas S Dade -	For the 'icons/.appicon.info' and AppQuery.FPL fix.
Niklas Angare -	FlipComment.FPL
Carsten Orthbandt -	MHead.FPL, FACT_IBM.FPL and more! Also brought us lots of ideas to implement!
Peter Carlsson -	Devpac.FPL, FileType.FPL, ...
Volker Güth -	For the wordwrap alternative and for taking over the German translations...
Bill Beogelein -	Unbelievable amounts of ideas and suggestions...

...the registered FrexxEd users...

We wouldn't be so keen on continuing to make these updates if it weren't for you!

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

Mathias Axelsson -	FPL user and AmiNet uploader. Holy keeper of the FrexxEd WWW-page. Thanks for the ETags source.
Stefan Boberg -	A4000/A3000, lots of reports with "strange" HW. Also thanks a lot for 'Lha' and moral support!!
Pontus Hagland -	Beta testing, FPL doubter (initially at least)!
Tommy Hallgren -	Protector of the Forth programming language... :) When will we get an etags parser?
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 convertion to Swedish!

 FrexxEd wouldn't be like this without you guys!

 ...the translators...

Michiel Willems -      (Dutch) Thanks for making the Menu.FPL too!
Massimiliano De Otto - (Italian) A really good work!
Rolf Kunisch -         (German) Well, you fixed it in the end, splendid!
Robert Ramiega -       (Polish) Not a standard language, but still...
Marc-Berco Fuhr -      (Danish) A nice piece of translation!
Stephane Zermatter -   (French) Thanx a lot, we needed that!
Jon-Inge Paulsen       (Norwegian) Just awesome!

 ...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!!
                       Drew our beautiful Magic WB icons! Great pal.

Nico Francois -        Author of reqtools.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 contact with and information from
                       the rest of the [Amiga] world!

 ... the magazine reviews...

Datormagazin           5 out of 5 must be considered to be a fairly positive
                       review, even though there were some glitches...
Amiga Magazin          9.6 of 12 in a review that actually understood our
                       thoughts behind FPL...
```

1.24 Things left to do...

```
 FrexxEd Suggestions/ideas/new things
 ====================================

 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 list of what we hope to implement sooner or later. This is only
things from the fantasy of us and other users of FrexxEd. Perhaps nothing of
this will ever become reality, but perhaps a lot of it will...

 Every idea is written alone with the name of at least one of the persons who
have brought the idea to us. They are presented in a very *random* order here,
without any meaning intended about which comes first and which comes last. If
you have any idea or thought that is not presented here, write us a line and
tell us about it!

 New features/functions
 ======================

* Are there any info-variables (or any other way) to get values like:
  1) The name of the function currently being executed.
  2) The filename that #1 is located in.
  3) The current line # within the function.
  4) The current line # within the file.

  5) The last keystroke that was hit (regardless of how long ago).

  Bill

* > Replace(1, "x", "y", "bwf+");

  Shouldn't that only be effecting the current marked-block, and
  nothing else?

  Bill

* Is there anyway to make the buttons that appear in Request() have
  keyboard equivalents?  Underscore-marking gadgets have been a
  standard part of the AmigaOS for ages now.

  Maybe something with like:
  Request("My Text", "My Title", "_Many | Ma_ybe | M_ost | M_ine | _All"

  Bill

* Whenever there's an error in my FPL code that is loaded at startup,
  I get the "Error at Line 186 of Path:FileName.FPL" requester.  Is
  there a way to change that into a "LOAD" "CONTINUE" requester?
  It would load the exact offending file, it would jump down to the
  offending line#, and we could edit it right then and there.

  Bill

* Is there a way for me to use RequestWindow() and create a list-view
  gadget but WITHOUT the string gadget underneath? (Perhaps a
  future release will allow the removal of the string-gadget.
  Perhaps by specifying a default &int instead of a default &string.
  After clicking OK, that int will contain the number of the choice
  picked.)

  Bill

* I considered using 20 checkmark-gadgets, but I can't figure out how
  to mutually exclude.  (Perhaps a future release will allow an
  additional int parameter for checkmark-gadgets.  Whichever bits are
  set, will mutually exclude those similarly marked checkmark-
  gadgets.)

  Bill

* How about making two global (uh, exported) variables containing the
  Customizing->Program position?

  Jesper Skov

* for the status line / window title :
  Add functions like:
  TitleCheck() and StatusCheck().
  They would wait for a mouse-click in the titlebar and status-line.
  (Titlebar or status-line dragging would NOT count as a click.)

  The 2 functions would return the column # that the user clicked at.

  We could set-up our status-line display to show the "cCiIbBmMpP" flags.
  Clicking on them would toggle them.

  Bill Beogelein

* replacing all occurrences start to finish?
  do I have to save the line/column position myself, move to the
  top, do the replacing, and move back to the old line/column
  position?  Which won't be where it was anyway because an unknown #
  of lines/characters have now been added and removed.  So how is
  that done?  A "G" flag would do it all.
  (I.e, do not just search from current position)

  Bill Beogelein

* Is there any way for me to save the size (and position) of a
  zoomed-down window?  Without that feature I always have to move and/or
  resize the window after zooming.

  Maybe 4 new info-variables could be added to a future release:
  zoom_height
  zoom_width
  zoom_xpos
  zoom_ypos

  Bill Beogelein

* Maybe someday you'll consider adding an optional additional 4th parameter:
    AssignKey("MyFuncIfTrue();", "'F4'", "changes", "MyFuncIfFalse();");

  Bill Beogelein

* More than one block marked at the same time. With different colors.

  Bill Beogelein

* "reversed" blocks. Using the same bitplane as the text.

  Bill Beogelein

* Ability to recognize left/right amiga key presses.

  Bill Beogelein

* Amiga symbols in the menu when the key seqyence is *only* amiga and another
  key.

  Mathias Axelsson

* Slider gadgets in request windows

  Carsten Orthbandt

* Independent cursor move 'layout' style, no stop at eol

  Carsten Orthbandt

* ability to Change status line bg pen

  Carsten Orthbandt

* I want the ability to lock the status display. If I display any text in
  the status line, it disappears immediately.

  Carsten Orthbandt

* A option to have the status line at the top of the text.

  Carsten Orthbandt

* Busy-pointer activated by FPL, so that FPL programs that know they will
  take some time can show it, and at the same time block the windows of
  FrexxEd.

  Emil Brink

* Is it possible to have control codes in the text affect the output?
  colour/style control of text areas
  Jesper Skov, Carsten Orthbandt, Emil Brink

* Mouse pointer blanker for FrexxEd only. Without affecting the rest of
  the system.

  Håkan Svensson

* "Save & Quit" in the menu, in the style of CED.

  Lars Magnusson

* Idea to fix problem with big promptinfo windows: Only allow one row of
  infos... Add "prev" and "next" gadgets in the bottom to switch between
  different info vars (different columns as it is now)

  Jesper Skov

* "lock" line numbering. Making insertion/deletion after compiler errors not
  change the line numbers...

  Niklas Larsson

* Reset colors should reset them to the "last saved" state, *NOT* the WB-
  colors

  Mikael Saker

* Add variable description to the CreateInfo() function!

  Magnus Holmgren

* Horizontal slider option.

  Mikael Berglund

* Move rectangular block. Text to the right of the block's left margin should
  be moved, the text on the left side should be left alone!

  Mikael Berglund

* SMART_REFRESH window

  Tommy Hallgren

* more native string routines! upper/lower case string would be nice.
  And right, left. Have a look at the string support in ARexx and make
  me happy

  Jesper Skov

* Images for Amiga, Alt and Control in the menu shortcuts!

  Torsten Stolpmann

* Why not have an own window for local/global user-defined variables?
  (And the system ones have their own window(s))
```

Bjorn Reese

* Free tab positions with _real_ tabs.

  Jesper Skov

* Can I make the scroller not as wide. it is too wide at the moment and it would be nice to be able configure it under 'display'.

  Edd Dumbill

* ELisp to FPL converter

  Jesper Skov

Ideas still left, brought by ourselves or beta testers...

* Enable the function like KeyPress("alt a"); to emulate an actual "alt a" press!
* Mark rectangle should not be limited by the last line's length
* Arrows on the sliders
* Custom buttongadgets on the status lines or perhaps only on the top of the window
* 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
* 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.
* 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.

1.25 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.

1.26 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 rectagular 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 (in the local info variable 'type'), 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!

There are two named block buffers by default that might be of your interest. The first one is called "DefaultBlock" and is the buffer to which all regular block operations will send the text if no special block buffer is specified. "YankBuffer" is the name of the block buffer to which all functions that appends text to the yank buffer add text to.

```
    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 loosing 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...

1.27 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 the 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.

Each entry in FrexxEd can have a different FACT. By using the FACTCreate() and FACTDelete() new FACTs can be made and removed. Change from one FACT to another in an entry by setting the 'fact' info variable to the name of the desired FACT.

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

```
   FACT([name], char, tags...);
```

'name' can be specified and should be if you want to change the contents/ looks of a single FACT.

'char' is the ASCII number of the character you'd like to change the definition of, or one of the 'special' codes:

```
 -1 End Of File string
 -2 ContinuedLine string (if a line continues at the edge of the window)
 -3 Fold start character (displayed to the left of a line with fold)
 -4 Fold area character (displayed to the left of a shown folded area)
 -5 Fold margin fill character (which the fold margin will consist of)
```

```
      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 defualt 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.)

1.28 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()).

1.29 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.

1.30 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'.

Detailed information in the sections:

Encryption

1.31 Loading files

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 view's status line shows the name of the buffer that is being displayed in that view, 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.

1.32 Saving files

```
    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 set to "always" for the saved buffer, FrexxEd
will write an icon for that file. Read further below in the icons section!

 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.

 By default FrexxEd saves all files using a "safe save" method, which first
stores the file on disk in a temporary name and then renames that file into
the destination name. Switch off the global 'safe_save' variable to force
FrexxEd not to use it...
```

1.33 Icons

```
    Icons
    =====

 When FrexxEd saves a buffer with the 'save_icon' option set to "always", 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 previos 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.

 The AppIcon used by FrexxEd is found/used as the file called
'FrexxEd:Icons/.AppIcon.info'.
```

'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! Idea 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!!

Magic WB icons in the distribution package are drawn by Edd Dumbill. *Thanks!*

1.34 Compression

```
    Compresssion
    ============
```

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' info variable 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 propably 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.

1.35 Encryption

```
    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.

1.36 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:Icons/.AppIcon.info' is the one FrexxEd will try to load before using the internal.

* 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.

* FrexxEd allows itself to get iconified. An iconfied FrexxEd appears only as an AppIcon in the Workbench window. Doubleclick the AppIcon to deiconify FrexxEd again.

* FrexxEd brings its screen and/or window to front if its AppIcon is double-clicked.

* The volume mounted by FrexxEd appears just like a regular disk on the Workbench main window. Double-click on its icon to visulized the contents of the disk's FrexxEd. Move files in and out of it just like with regular disks.

1.37 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).

1.38 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.

1.39 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 runable 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 (in fact even 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 for further description of the language FPL.

1.40 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
- Where the cursor will be when that view/buffer is active

Status line
- The revered video line at the bottom of every view

1.41 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.

To make FrexxEd able to remember several cursor positions in the same file, without viewing them, we introduced 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.

1.42 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 information of the status line is user customizable and can be made to contain nearly anything. Below is the default/built-in look described.

```
 The status line is by default 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)
                R = (R)ead-only buffer/file

 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.
```

1.43 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
requester with a list 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 below the search/replace field(s), 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.

* 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!

* 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.

* Wildcards
  Wildcards are since 1.3 true UNIX-style
                regular expressions
                . They are
  extremely powerful and sometimes quite hard to use.

   Replace String
   ==============

 When using regular expression search, you have possibility to use parts of
the matched string in the replace string!

 It can refer to all or part of what is matched by the search string. `\&' in
the replace string stands for the entire text being replaced. `\D', where D is
a digit, stands for whatever matched the Dth parenthesized grouping in the
search string. To include a `\' in the text to replace with, you must give
`\'. FrexxEd also supports the '/' instead of the real '\' in all kinds of
regular expressions!

For example,

 Replace("c[ad]+r", "\&-safe");

replaces (for example) `cadr' with `cadr-safe' and `cddr' with `cddr-safe'.

 Replace("(c[ad]+r)-safe", "\1");

performs the inverse transformation.
```

1.44 Regular expressions

Syntax of Regular Expressions
=============================

NOTE: The characters ‘ and ’ are used in this chapter to limit discussed strings. If the text is discussing the string ‘foobar’, you should not consider the quotes as part of the string. The string is then simply the foobar word!

Regular expressions have a syntax in which a few characters are special constructs and the rest are "ordinary". An ordinary character is a simple regular expression which matches that same character and nothing else. The special characters are $^.*+?[]()|/\#. Any other character appearing in a regular expression is ordinary, unless a ‘\’ precedes it.

For example, ‘f’ is not a special character, so it is ordinary, and therefore ‘f’ is a regular expression that matches the string ‘f’ and no other string. (It does *not* match the string ‘ff’.) Likewise, ‘o’ is a regular expression that matches only ‘o’. (When case distinctions are being ignored, these regexps also match ‘F’ and ‘O’, but we consider this a generalization of "the same string", rather than an exception.)

Any two regular expressions A and B can be concatenated. The result is a regular expression which matches a string if A matches some amount of the beginning of that string and B matches the rest of the string.

As a simple example, we can concatenate the regular expressions ‘f’ and ‘o’ to get the regular expression ‘fo’, which matches only the string ‘fo’. Still trivial. To do something nontrivial, you need to use one of the special characters. Here is a list of them.

‘. (Period)’

is a special character that matches any single character except a newline. Using concatenation, we can make regular expressions like ‘a.b’ which matches any three-character string which begins with ‘a’ and ends with ‘b’.

‘*’

is not a construct by itself; it is a postfix operator, which means to match the preceding regular expression repetitively as many times as possible. Thus, ‘o*’ matches any number of ‘o’s (including no ‘o’s).

‘*’ always applies to the *smallest* possible preceding expression. Thus, ‘fo*’ has a repeating ‘o’, not a repeating ‘fo’. It matches ‘f’, ‘fo’, ‘foo’, and so on.

The matcher processes a ‘*’ construct by matching, immediately, as many repetitions as can be found. Then it continues with the rest of the pattern. If that fails, backtracking occurs, discarding some of the matches of the ‘*’-modified construct in case that makes it possible to match the rest of the pattern. For example, matching ‘ca*ar’ against the string ‘caaar’, the ‘a*’ first tries to match all three ‘a’s; but the rest of the pattern is ‘ar’ and there is only ‘r’ left to match, so this try fails. The next alternative is for ‘a*’ to match only two ‘a’s. With this choice, the rest of the regexp matches successfully.

‘+’

is a postfix character, similar to ‘*’ except that it must match the preceding expression at least once. So, for example, ‘ca+r’ matches the strings ‘car’ and ‘caaaar’ but not the string ‘cr’, whereas ‘ca*r’ matches all three strings.

‘?’

is a postfix character, similar to ‘*’ except that it can match the preceding expression either once or not at all. For example, ‘ca?r’ matches ‘car’ or ‘cr’; nothing else.

’#’

this characters is not part of the original UNIX/Emacs RE, but added by us to enable even better expressions. ‘#’ matches any number of characters until the string following it matches. I.e, searching for ‘foo#bar’ will find the first occurrence of the word ‘foo’ followed by any number of characters and then the word ‘bar’.
To make the REs even more like Amiga-style, we allow a following ‘?’-character after the ‘#’-character without changing the meaning of it. Thus, ‘foo#?bar’ will match just as good.

‘[...]’

is a "character set", which begins with ‘[’ and is terminated by a ‘]’. In the simplest case, the characters between the two brackets are what this set can match.

Thus, ‘[ad]’ matches either one ‘a’ or one ‘d’, and ‘[ad]*’ matches any string composed of just ‘a’s and ‘d’s (including the empty string), from which it follows that ‘c[ad]*r’ matches ‘cr’, ‘car’, ‘cdr’, ‘caddaar’, etc.

You can also include character ranges a character set, by writing two characters with a ‘-’ between them. Thus, ‘[a-z]’ matches any lower-case letter. Ranges may be intermixed freely with individual characters, as in ‘[a-z$%.]’, which matches any lower case letter or ‘$’, ‘%’ or period.

Note that the usual special characters are not special any more inside a character set. A completely different set of special characters exists inside character sets: ‘]’, ‘-’ and ‘^’.

To include a ‘]’ in a character set, you must make it the first character. For example, ‘[]a]’ matches ‘]’ or ‘a’. To include a ‘-’, write ‘-’ at the beginning or end of a range. To include ‘^’, make it other than the first character in the set.

‘[^ ...]’

‘[^’ begins a "complemented character set", which matches any character except the ones specified. Thus, ‘[^a-z0-9A-Z]’ matches all characters *except* letters and digits.

‘^’ is not special in a character set unless it is the first character. The character following the ‘^’ is treated as if it

```
     were first (`-' and `]' are not special there).

     A complemented character set can match a newline, unless newline is
     mentioned as one of the characters not to match.  This is in
     contrast to the handling of regexps in programs such as `grep'.

`^'
     is a special character that matches the empty string, but only at
     the beginning of a line in the text being matched.  Otherwise it
     fails to match anything.  Thus, `^foo' matches a `foo' which
     occurs at the beginning of a line.

`$'
     is similar to `^' but matches only at the end of a line.  Thus,
     `xx*$' matches a string of one `x' or more at the end of a line.

`|'
     specifies an alternative.  Two regular expressions A and B with
     `|' in between form an expression that matches anything that
     either A or B matches.

     Thus, `foo|bar' matches either `foo' or `bar' but no other string.

     `|' applies to the largest possible surrounding expressions.
     Only a surrounding `( ... )' grouping can limit the scope of
     `|'.

     Full backtracking capability exists to handle multiple uses of
     `|'.

`( ... )'
     is a grouping construct that serves three purposes:

       1. To enclose a set of `|' alternatives for other operations.
          Thus, `(foo|bar)x' matches either `foox' or `barx'.

       2. To enclose a complicated expression for the postfix operators
          `*', `+' and `?' to operate on.  Thus, `ba(na)*' matches
          `bananana', etc., with any (zero or more) number of `na'
          strings.

       3. To mark a matched substring for future reference.

     This last application is not a consequence of the idea of a
     parenthetical grouping; it is a separate feature which is assigned
     as a second meaning to the same `( ... )' construct.  In practice
     there is no conflict between the two meanings.  See '\D' for an
     explanation of this feature.

`\' or '/'
     has two functions: it quotes the special characters (including
     `\' and '/'), and it introduces additional special constructs.

     Because `\' quotes special characters, `\$' is a regular
     expression which matches only `$', and `\[' is a regular
     expression which matches only `[', etc.
```

Note: for historical compatibility, special characters are treated as ordinary ones if they are in contexts where their special meanings make no sense. For example, '*foo' treats '*' as ordinary since there is no preceding expression on which the '*' can act. It is poor practice to depend on this behavior; better to quote the special character anyway, regardless of where is appears.

For the most part, '\' followed by any character matches only that character. However, there are several exceptions: two-character sequences starting with '\' which have special meanings. The second character in the sequence is always an ordinary character on their own. Here is a table of '\' constructs.

'\D'

after the end of a '(...)' construct, the matcher remembers the beginning and end of the text matched by that construct. Then, later on in the regular expression, you can use '\' followed by the digit D to mean "match the same text matched the Dth time by the '(...)' construct."

The strings matching the first nine '(...)' constructs appearing in a regular expression are assigned numbers 1 through 9 in order that the open-parentheses appear in the regular expression. '\1' through '\9' refer to the text previously matched by the corresponding '(...)' construct.

For example, '(.*)\1' matches any newline-free string that is composed of two identical halves. The '(.*)' matches the first half, which may be anything, but the '\1' that follows must match the same exact text.

If a particular '(...)' construct matches more than once (which can easily happen if it is followed by '*'), only the last match is recorded.

'\`'

matches the empty string, provided it is at the beginning of the buffer.

'\''

matches the empty string, provided it is at the end of the buffer.

'\b'

matches the empty string, provided it is at the beginning or end of a word. Thus, '\bfoo\b' matches any occurrence of 'foo' as a separate word. '\bballs?\b' matches 'ball' or 'balls' as a separate word.

'\B'

matches the empty string, provided it is *not* at the beginning or end of a word.

'\<'

matches the empty string, provided it is at the beginning of a word.

'\>'

```
     matches the empty string, provided it is at the end of a word.

`\w'
     matches any word-constituent character.  The syntax table
     determines which characters these are.

`\W'
     matches any character that is not a word-constituent.

`\sC'
     matches any character whose syntax is C.  Here C is a character
     which represents a syntax code: thus, `W' for word constituent,
     `(' for open-parenthesis, etc. See the
                FACT description
                 for more
details!

`\SC'
     matches any character whose syntax is not C.

 The constructs that pertain to words and syntax are controlled by the setting
of the FACT.
```

1.45 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’.

1.46 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        First deactivates the current string input field.
                Then it is 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.

1.47 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*". The distinction between upper and lower case does not matter 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 unlimted 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 (starts an FPL program that uses the PromptBuffer() function), or simply view all buffers that are currently in memory.

1.48 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 1'
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.

1.49 Using macros

To record a sequnce of actions and then re-play them, is what we refer to as "macros".

Record a macro
~~~~~~~~~~~~~~
FrexxEd allows the user to at any time invoke the function 'MacroRecord', which by default is done with the key sequence amiga-m (when done, the window and screen title will say 'begin recording'). All actions followed that function invoke will then be recorded. You stop the recording by calling the function again; by pressing amiga-m. You are then supposed the enter the key sequence which you think should replay your recorded sequence of actions. The entered key sequence should be terminated by pressing 'escape'.

Replay a macro
~~~~~~~~~~~~~~
Now, the actions can get replayed by pressing the given key sequence!

Format of the macro
~~~~~~~~~~~~~~~~~~~
The macro is stored and run as a FPL program. Every single action recorded is translated into FPL function invokes, thus enabling the user to view the macro and by that understand what it does, and further more, if the recorded script happens to do something wrong, it is dead easy to make small adjustments.

Edit/view macros
~~~~~~~~~~~~~~~~
Select the menu item brought with the Menu.FPL file called "Special->Edit macros..." and select the macro you want to edit. They're named with a unique name partly built by the key sequnce you assigned it to!

Execute a buffer with a key
~~~~~~~~~~~~~~~~~~~~~~~~~~~
In the menu brought to FrexxEd through the execution of Menu.FPL, the menu item "Special->Fastkey buffer exec..." assigns a key sequence to the execution of the current buffer. Convenient to use when you have a FPL program in a buffer and want to execute it repeatedly.
~~~~~~~~~~~~~~~~~~~~~~~~~~~

* SEE MacroRecord() in the function reference.

1.50 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.

1.51 Filehandler

(Big thanks to Linus Nielsen who brought us the code which made ↩ this feature
possible.)

What's the filehandler?
~~~~~~~~~~~~~~~~~~~~~~~
Each started FrexxEd mounts a virtual disk. By reading or writing to that
disk, you can access the buffers of FrexxEd from outside the editor without
having to mess with temporary files. You can for example compile a source that
never has been saved, straight from within FrexxEd, or you can re- direct the
output from a command straight into a FrexxEd buffer.
By toggling the global 'filehandler' info variable, you can turn it on and
off.

What's it called?
~~~~~~~~~~~~~~~~~
By default, the disk name of the first started FrexxEd will become
'FrexxEd0:', the disk name of the second FrexxEd will be 'FrexxEd1:' and so
on. By using the command line option or tool type called 'DISKNAME' you can
set the name that FrexxEd will use. Setting DISKNAME to "off" on the command
line/tool type will switch off the filehandler!

How can I use it?
~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~

Workbench users will notice it immediately after FrexxEd has been started since the icon of the disk will appear in the Workbench main window (actually, the default disk icon will appear since there won't be any 'Disk.info' file in the FrexxEd disk!). It can then be used just like the standard ram disk. The only difference is that the contents of the FrexxEd disk will also be viewable/editable from within FrexxEd.
Shell users can view all buffers in FrexxEd by typing "dir FrexxEd0:" just like the contents of any other drawer or disk is listed.
Virtually all programs that operate on files can use the FrexxEd0: drive instead of any other drive.

Can I turn it on/off?

Of course, 'filehandler' is the name of the global boolean info variable controlling its existance. NOTE: if you decide to switch off the filehandler while it still is in use, it can't be removed completely. It will then refuse all new uses of it, but allow the old holders to finnish. When all programs have quit using it, it will disappear.
You can of course restart the filehandler at any time.

Identical filenames

Since FrexxEd allows more than one file with the same name loaded, there is a slight problem to visualize all buffer names correctly. If there is more than one occurrence of a file name, the second and the following names will then be displayed as "<name>,<num>". I.e, having two buffers named "foobar" will make them appear as "foobar" and "foobar,2"

What restrictions are there?

The FrexxEd disk doesn't support directories. All files copied to the disk or buffers created in the editor reside in the same 'root' directory/drawer.

Files can't be renamed to a name that already is being used in the same disk/editor (as can be done within the editor).

Files that are modified, that the cursor is currently in and those that are refered to ("in use") by the editor can't be deleted. They will simply return 'object in use'.

Files without names (displayed as "*noname*" within the editor) will be named "noname" when accessed from the filehandler.

FrexxEd can't be quit as long as there is any program or process that has the FrexxEd disk locked or opened in some way. Attempting to do so will make FrexxEd display a requester saying it can't quit.

All packet types are not supported. For a technical and detailed summary, see the list further down.

The volume can still be accessed even though the editor is locked up with things like diplaying a requester or similar.

Things to think of

If you select 'Window->Snapshot->...' in the Workbench menu when the FrexxEd disk is selected, you will get a Disk.info or <filename>.info file created inside FrexxEd...

Filehandler Generated Exceptions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Whenever a file is accessed (created, deleted, opened, closed, read, written) through the filehandler, an exception is generated - called 'HandlerAction'.
By making FPL programs listening to that exception, you can make them react as soon things are done to files, without manually having to invoke them...

Technically speaking, exactly what packets does the handler reply to?
~~~~~~

```
ACTION_COPY_DIR
ACTION_DELETE_OBJECT
ACTION_DISK_INFO
ACTION_END
ACTION_EXAMINE_NEXT
ACTION_EXAMINE_OBJECT
ACTION_FINDINPUT
ACTION_FINDOUTPUT
ACTION_FINDUPDATE
ACTION_FREE_LOCK
ACTION_INFO
ACTION_IS_FILESYSTEM
ACTION_LOCATE_OBJECT
ACTION_READ
ACTION_RENAME_OBJECT
ACTION_SAME_LOCK
ACTION_SEEK
ACTION_SET_DATE
ACTION_SET_COMMENT
ACTION_SET_PROTECT
ACTION_WRITE
```

1.52 Using folds

What is a fold?
~~~~~~~~~~~~~~~
Folding is a way to prevent certain parts of the buffer from being viewed. By making well placed folds, you can make the buffer much more easier overviewed and understood.
When making a fold, the first line will remain visible and the rest will be hidden. Let's say we have a text like the following:

```
 1: This program was cracked on request of Đemax/Delight. I must admit that
 2: it took me some hours to crack it (mainly because i'm lazy and tired),
 3: but now I can even make a FrexxEd auto-registrator for new versions (if
 4: the authors doesn't change protection) but I won't cause i'm too lazy...
 5: This little crack note was written in FrexxEd.
```

and we fold line 1 to 4. Then the text on the screen will look like:

```
 1: This program was cracked on request of Đemax/Delight. I must admit that
 5: little crack note was written in FrexxEd.
```
~~~~~~~~~~~~~~~

You will also notice a little 'fold mark' at the left of the first line of each fold that informs the viewer about the fact that there is a fold there. That fold mark character is changeable through the FACT.

Dealing with folds

You can at any time remove the fold, making it look like the original unfolded text again, or you can unfold the folded area just temporaily and fold it again later. FrexxEd even offers the possibilty to display the fold only, hiding everything else.

Normally, you block mark a few lines and invoke the 'Fold()' function to make the area into a fold.

By calling the function 'FoldShow()' you unfold the fold currently under the cursor, and make it displayed. 'FoldShow()' makes the fold visible, but still remembers all lines that are in the fold. To hide the fold again, just invoke the 'FoldHide()' function.

To remove a fold, invoke the 'FoldDelete()' function standing on a fold, and the text will get unfolded before the fold information is removed.

The most used fold functions are easily invoked by using the menu brought to FrexxEd through the Menu.FPL program, and there, under the Edit title.

Saving/loading folded buffers

To make a loaded buffer get folded just the way it was when it was saved, FrexxEd inserts strings that tell about where the folds start and end when saving a buffer. The strings used will get custom strings written on both sides of them to enable them to appear as comments to programs that use/access/read the saved file...

The info variable 'fold_save' turns this feature on and off. If the variable is set to off, no fold marks will be saved and no marks will be recognized when a folded file is being loaded.

Search/replace in folded buffers

The search and replace functions feature a flag that enables or disables the function from searching inside the folded areas.

Folds displayed on screen

There are a few different things that handles the setup of the display of each fold.

First, you can set the 'fold_width' info variable to set the width of the margin which FrexxEd will use for fold-information only. It will appear to the left of the rest of the lines.

Secondly, you can change the characters that are used within the margin to display the folds. A buffer with a fold being showed can look like:

```
AAA /* nonsense */
1>A /* First line of fold */
1BA /* second line of fold area */
1BA /* this is the last line of the fold */
AAA /* non-folded real-looking line! */
```

Where the characters '>', 'A' and 'B' all are changeable with the FACT() function! The setup like the example above is accomplished by using:

```
 FACT(-3, 's', ">");
 FACT(-4, 's', "B");
 FACT(-5, 's', "A");
```

1.53 Colours and styles within FrexxEd

Colour and styling of text within FrexxEd can be done in two ways ↩
:

1) Change single characters to look different using the FACT

2) Make strings (or start to end strings) get coloured using the Face
system.

FACT
~~~~
The FACT has the possibility to change the look of every single character on
screen. You could easily make all letters 'C' become green if that's what you
want. All FACT details are covered in the FACT's
special chapter.
Face
~~~~
The system of faces works like following. First, a buffer can only have one
Face in use at the same time. The Face is setup to change the colour or style
of certain strings or patterns using a few setup functions (by the face mode
programmer, still no need for the user to touch them).
All strings or patterns in the Face that are coloured, get a named 'Face
style' which decides how the strings of that style should look. Many strings
can get the look of the same style, and lots of different styles can of course
be used within the same Face.

Let's take a look at an example:

A Face is created that is called 'C' for dealing with C-program styling. A
bunch of Face Styles are created too, named like 'c-strings', 'c-comments' and
'c-keywords'. Lots of strings are then made to get recognized and
coloured/styled like 'for', 'while', 'do' and 'else' all get the 'c-keywords'
style and all text within /* to */ get the style 'c-comments' and so on.

At the time of writing, there is not yet any decent built-in user interface
for the end-user to change the Face Style definitions, but all styles as well
as background and foreground pens can be changed on all styles. FaceEdit.FPL
was brought to you until there is a better prefs editor!

For the convenience of Face programmers and users, FrexxEd offers a set of
built-in Style names to be used for appropriate text-patterns.

Programming your own Face
~~~~~~~~~~~~~~~~~~~~~~~~~
Below follows a little example of how it very well could look when you create
your own Face:

```
 {
  int face = FaceGet("c-mode", 1); /* create one if missing */
```
~~~~~~~~~~~~~~~~~~~~~~~~~

```
  if(face) {
    int style;

    /* Get the style named "c-keywords" OR create one that is bold with
       foreground pen 3 and background pen 0. If none matched and none
       could be created, it will return the style of the best match and
       no style will be named like this. */
    style = FaceStyle("c-keywords", "bold", 1, 0);
    FaceAdd(face,  /* add word(s) to this face */
            style, /* use the c-keywords style */
            /* Specify all words we want to add to this face with this
               particular style, we can of course add more words at a
               later time */
            "for|do|while|else|if|int|long|short|char|extern|"
            "static|unsigned|signed|return|continue|break|void|"
            "case|switch|WORD|LONG|struct",
            "word" /* these are word-only matches, that must be surrounded
                      with non-word letters to get recognized */
            );

    style = FaceStyle("c-comments", "italic", 3, 0);
    FaceAdd(face, style, "/*", "anywhere|strong", "*/");
    FaceAdd(face, style, "//", "anywhere|strong|doublechk", "\n");

    style = FaceStyle("c-cpp", "normal", 2, 0);
    FaceAdd(face, style, "#", "backslash|1nonspace|weak", "\n");

    style = FaceStyle("c-string", "italic|bold", 2, 0);
    FaceAdd(face, style, "\"", "backslash", "\"");
  }
  /* else
    face is 0, which means GetFace() failed somehow! */
}

/*
 FLAGS for FaceAdd():
 ====================
 usepipe   - the pipe ('|') letter is a part of the actual string
 word      - the string(s) must match as a word-only
 anywhere  - matches wherever it appears in the text
 strong    - a strong string. Will conquer weak ones.
 weak      - a weak string. Will be conquered by strong ones.
 1nonspace - must be the first non-whitespace on a line to match
 backslash - a letter following a backslash will be ignored
 doublechk - set this on strongs to make it check the weak too when
             reaching the end (allowing them to end with the same string)
*/
```

1.54 Multiple windows with one FrexxEd

```
Multliple windows within one FrexxEd means, that you as a user can:

* Open any amount of windows.
* Display the same buffer in any amount of windows.
* Split any window in any amount of views (like before)
```

* Make all new buffers appear in new windows (like when opened)
* Iconify the windows separately
* Automatically arrange all FrexxEd windows
* Go to/jump to all windows using menuitems or shortkeys.

Multliple windows have been transparantly added to FrexxEd to adjust to all the former functionalities and systems previously used within FrexxEd. On the choise of the user, all can be made to look just like FrexxEd looked like before the multi window support. If on the other hand, the user wants to take advantage of these new features, FrexxEd offers fully integrated functions to make multi-window editing a pleasent time.

To get the best use of the multiple window system, we suggest you use the default delivered Menu.FPL menu strip that sets up a full and extensive menu strip for your FrexxEd. That menu strip does now contain a full setup of window functions and tools.

Take a look at the new way you can set 'popup_view' to show your new buffers.

1.55 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 bazillions 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!

1.56 Change colours

The current version of FrexxEd has a very limited support for ↩ using different colours.

FrexxEd uses the four first system pens for all graphics.

Adjust the colors by selecting the menu item '
Customizing->Colours->Adjust

'.
The colour values will be stored together with all the rest of the info
variables when you save them.

The names of the info variables that hold the 12-bit colour representation
values are called 'colour0', 'colour1', 'colour2' and 'colour3'.

Also selectable is which pen FrexxEd should use to display the information on
the right part of the status line. By setting the info variable 'pen_info' you
tell FrexxEd which pen to use.

1.57 How to 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
fonts for best look. One is used as 'system font', 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 menues.

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.

1.58 How to 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 simlar information, refer to the
Functions.guide 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.
                If this is enabled, iconify will be done _with_ icon, if
                disabled, iconify is done _without_ icon!

 '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 *each* character is read as one single byte, independent of what the character may look like on screen.
'changes'	Holds the number of changes done to the buffer since it was last saved or loaded.
'colour0'	Holds the 12 bit colour value of color 0.
'colour1'	Holds the 12 bit colour value of color 1.
'colour2'	Holds the 12 bit colour value of color 2.
'colour3'	Holds the 12 bit colour value of color 3.
'column'	The column number of the current cursor position.
'comment'	File comment to the current buffer.
'comment_begin'	String that precedes the fold information FrexxEd stores when a folded file is saved.
'comment_end'	String that follows the fold information FrexxEd stores when a folded file is saved.
'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.
'disk_name'	Name of the mounted volume

this FrexxEd has mounted.

'ds_Days'	The modification date of the buffer's file in number of days since the 1st ofJanuary 1978.
'ds_Minute'	The modification date of the buffer's file in number of minutes since 00:00 the day specified in 'ds_Days'.
'ds_Ticks'	The modification date of the buffer's file in number of ticks (seconds * 50) since the full minute in the 'ds_Minute' field.
'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. "All" 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! Using the info variable from an FPL program will give you the following cycle values: 0 - Off 1 - Relative 2 - All
'fact'	The name of the FACT to use in this entry.
'fast_scroll'	Enable usage of the fastgraphics.library (C) Jesper Skov. This improves text and scroll speed a lot when using FrexxEd on a screen with 8-pixel wide fonts.
'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.
'filehandler'	Enables the filehandler. NOTE: if the filehandler is in use when this is turned off, it will remain in memory and in the system lists as long as it still is in use. It will get removed as soon as it can. While it is disabled but still present, no new locks or opens will be allowed.
'fold_save'	Whether fold information should be included when a file is saved, and whether folded information should get interpreted when files are loaded.
'fold_width'	Width of the margin to use for fold information.

```
'fpl_debug'    Tells FrexxEd to run all FPL programs in debug mode or not.

'fpl_path'     Which directories that should be searched through at each FPL
               file execution. Default is first current directory and then
               'FrexxEd:FPL/'.

'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.

'iconify'      Holds information whether FrexxEd is iconified.

'insert_mode'  Characters that are output in the buffer should insert them-
               selves 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. I.e, swedish would make it contain 'svenska,
               german 'deutsch' and so on...

'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" specfies the number of cursors from
'marg_lower'   each edge where the screen should move.
'marg_right'   If the margin is hit,
'marg_upper'   FrexxEd moves the screen in that direction.

'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 the left or right 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 previouly 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?
                Using the info variable from an FPL program will give you
                the following cycle values:
                0 - Replace
                1 - Split
                2 - Full

'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: "RWEDSPAH". (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.

'request_pattern' The pattern used in the filerequester.

'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 menues.

'rwed_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.
                                Using the info variable from an FPL program will ↩
                                   give you
               the following cycle values:
               0 - Never
               1 - Always
               2 - Parent

'screen_depth' The number of bitplanes that the FrexxEd screen uses.

'screen_height' The prefered height of the FrexxEd screen.

'screen_width' The prefered 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_flags' The search/replace flags as one int. This is mainly to be
               used for programs that wants to be able to reset all search
               flags after the program is done.
```

'search_fold'	The search/replace flag whether to search/replace within folded areas of a buffer.
'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, otherwice 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. By default FrexxEd puts the slider on the right side. Using the info variable from an FPL program will give you the following cycle values: 0 - None 1 - Right 2 - Left
'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 '|'.
'status_line'	This variable describes the format of the status line. See the dedicated status line customizing chapter for details.
'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.
'top_offset'	The line of the window that the current view starts at.
'type'	Buffer type bitmask: Bit 0 = standard file buffer Bit 1 = macro buffer Bit 2 = block buffer Bit 3 = invisble buffer. Buffers with this bit set will generally not become visible.

'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.
'visible'	Holds non-zero if the entry is visible.
'window'	Select how you want FrexxEd to startup as default. 'screen', 'window', 'backdrop' or 'winscreen'. 'winscreen' means as a regular window in its own screen. Using the info variable from an FPL program will give you the following cycle values: 0 - Screen 1 - Window 2 - Backdrop window 3 - Window on its own screen
'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'. Using the info variable from an FPL program will give you the following values: 0 - Visible 1 - Absolute
'window_title'	ONLY CHANGEABLE BY REGISTERED USERS!!! Text that appears in the window title! If not specified (a zero length string) it will display the FrexxEd internal string (that is the 'registered to XXXX' text).
'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.

1.59 How to change startup behaviour

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.

Easy n' quick:
~~~~~~~~~~~~~~
Put the FPL programs you want executed at startup in the FrexxEd:Startup/ directory. Remove them from there if you don't want them executed at startup!

Put the macros you always want to use in FrexxEd in the FrexxEd:Startup/ directory. Remove them from there if you don't want those macros at startup!

The FrexxEd:Rexx/ShowFPL.rx (thanks to Edd Dumbill for that one) ARexx program makes it a lot easier for you to select FPL programs to run at startup. It even installs/removes the programs for you!

1.60 How to 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()"
```

1.61 How to 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()
```

1.62 How to change action on keys

```
                 All users using a text editor has different backgrounds.  ←
                    Depending on your
background, you have different opinions, likings and habbits 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 somthing 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).

Hitting the key sequence "control escape" will abort/stop any running FPL program. Don’t try assigning anything to that sequence, you’ll get nothing but trouble!

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 enter key (found on the numeric keyboard)
Return	The regular below-backspace return key.
Tab	The tabulator key
Space	The space key
Spc	As above
num0-9	The number keys of the numeric keyboard
num[	The ’[’ key of the numeric keyboard
num]	The ’]’ key of the numeric keyboard
num/	The ’/’ key of the numeric keyboard
num*	The ’*’ key of the numeric keyboard
num-	The ’-’ key of the numeric keyboard
num+	The ’+’ key of the numeric keyboard
num.	The ’.’ key of the numeric keyboard

* ADVANCED INFORMATION

See ’ change mouse ’ - change the mouse buttons the same way!
See AssignKey() function in the function reference.

1.63 How to change icon images

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 used 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 used 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

1.64 How to 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.

SEE
                Default menu setup
                 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"

1.65 How to change mouse button actions

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

All mouse actions are of course 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 like:

name ~~~~	when ~~~~
Mouse<button>	mouse button pressed down
Mouse<button>Up	mouse button released
Mouse<button>Drag	mouse button held down while mouse moved
Mouse<button>Double	mouse button rapidly pressed twice
Mouse<button>Triple	mouse button rapidly pressed three times

where <button> is "Left", "Right" or "Middle" (without quotes). The names can be used with the AssignKey() just like with keys.

EXAMPLE
 Open a file (!) when the left mouse button is pressed:

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

Note:
 The right mouse button is only able to read if the 'right_mbutton' info variable is enabled. Do also note that if that is enabled, you can only bring up the menus when the pointer is on the window title border.

See also:
 "Function refernce: AssignKey()"

1.66 Save your changes

When you have altered all info variables to what you believe is a reasonable default environment for your editing, you should save them.

'Customizing->Save...' will bring up a file requester to enable you to save them in the same file as they were loaded from. By default that file is called FrexxEd.default and is positioned in FrexxEd:FPL/. FrexxEd features a command line and tool type option that enables it use another file on startup.

The saved file is a generated FPL program. If you save several different environments in different files, you can easily 'transform' into one of the other environments by simply executing that file as an FPL program. (I.e by selecting the menu item 'Special->Execute FPL file...'.)

NOTE:

FrexxEd only saves info variables in the .default file. All other changes and modifications have to be saved in their own proper ways. Also notice that local info variable will get their 'default' value stored in the .default file, not any local value.

1.67 Change the status line

FrexxEd offers since version 1.6 a very customizable status line. ↩ You control the contents of the status line by changing the contents of the global info variable called 'status_line'.

* SEE
    Screen orientation, status line

'status_line' can include a number of different flags which will ↩ make different information to appear. The flags are written in the style:

```
  $<flag>
```

where <flag> is one or more letters that specifies which data to include and possibly even in which format.

Flags available are:

```
  a - The ASCII character on the current cursor postion.

  A - The ASCII character number of the current cursor position.

  b - boolean display. This should be follwed by the TRUE letter, the
      FALSE letter and an info variable name within single quotes (').
      See example 3 below.

  c - Column number of the current cursor position.

  C - The column number string! In English, that is "Col:".
  f - The six standard flags in a bunch. They're shown as 'cIbMpR' which
      means in left-to-right order: buffer is changed, insert mode, block
      is being marked, a macro is being recorded, the buffer is loaded/
      saved using some compression and finally the buffer is read-only.

  l - Line number of the current cursor position.

  L - The line number string! In English, that is "Line:".

  n - Number of changes done to the current buffer.

  P - The page number of the current cursor position. Page is simply line
      number / 66.

  p - The line number of the current page.

  r - Number of lines in the current buffer.

  s - Current buffer size.

  x - Display the following information in lowercase hexadecimal.

  X - Display the following information in uppercase hexadecimal.

  ' - Special! By specifying $'<info variable name>', you can add the
      contents of any info variable to the status line.

EXAMPLES

 1. Make a status line that includes the flags and the buffer size:

    "$f Size: $s"

 2. Make a status line that includes the ASCII character under the cursor
    displayed in lowercase hexadecimal and the contents of the 'comment' info
    variable:

   "Char: $xA Comment: $'comment'"

 3- Make the status line show a 'Y' if we've switched on the 'appicon'
    setting and 'N' if we haven't:

    "AppIcon: $bYN'appicon'"
```

1.68 Patch internal functions

```
                 FPL programs/scripts in FrexxEd have more than one hundred and ↩
                    fifty internal
functions they can call to create the effect they want. Load() loads a file,
Search() searches for strings and so on.

 There are 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!
```

Table of Contents

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 "

1.69 Hook Description

"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 exceptions in some occations, 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), the other way around if the variable is preceeded with an exclamation mark) and even combined into more complex conditions as described below.

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!

Since some functions can be called with an optional number of parameters, all left-out parameters will be replaced with zero length strings or zeroes according to the types of the missing parameters.

HINT

If your past hook would like to know if the "real function" did well or not, call GetErrNo() the first thing in the hook since that will contain the actual error number then!

1.70 Hook Conditions

The conditions available and used with the Hook() functions are the same used with i.e AssignKey() and MenuAdd(). The 'condition' is implemented to make actions dependent on a certain set of info variables.

In the row of examples following, we'll name the info variables like 'name1', 'name2', 'name3' and so on for educational purposes. We will also consider variables to be 'set' or 'on' if it contains a non-zero value or a string text.

In the simplest form, you just specify the info variable you'd like the hook, key assign or whatever to be dependent on. To create more advanced and combined conditions, you have three operators to use:

! is the NOT operator, it makes the variable toggle its meaning. A non-zero variable will look like zero and a zero-length string will look like it contains something. If 'name1' equals TRUE, '!name1' will eual FALSE and vice versa.

& is the AND operator. It requires two variables written with this operator in between them, to equal TRUE to make the condition equal TRUE. 'name1&name2' will equal FALSE is both variables aren't TRUE.

| is the OR operator. It requires one of two variables written with this operator in between them, to equal TRUE to make the condition equal TRUE. 'name1|name2' will equal TRUE if any of the variables equals TRUE.

The NOT-operator has the highest priority, while the rest is read in a left-to-right order. The operators can be combined to any length.

1. A condition that is TRUE if the variable 'name1' is set:

```
        "name1"
```

```
2. A condition that is FALSE if the variable 'name1' is set:

        "!name1"

3. A condition that is TRUE if the variable 'name1' is set *and* 'name2'
   isn't:

        "name1&!name2"

4. A condition that is TRUE if the variable 'name1' is set *or* 'name2'
   isn't:

        "name1|!name2"

5. A condition what equals TRUE if any of the three variables is set:

        "name1|name2|name3"
```

1.71 Hook 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");
```

1.72 Things to Consider about Hooks

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 an 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!

1.73 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).

The FrexxEd distribution package contains hundreds and yet hundreds of FPL programs that bring otherwise non-existant functions or features to FrexxEd. Not all of them are written by Daniel&Kjell, but a huge share has been given to us by friendly users and FPL programmers. A few words about those:
There is no warranty for the programs. All use of them is done at the user’s own risk. The programs have been included and distributed in the hope that you will find them useful and working, but the entire risk as to the quality and performance of the programs is with you. should any of the programs prove defective, you assume the cost of all necessary servicing, repair or correction. Report bugs though. They tend to get fixed really quick.

1.74 Introduction to FPL

FPL (Copyright © 1992-1995 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!

Hitting the key sequence "control escape" will abort/stop any running FPL program.

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." for inclusion in future releases or special FPL packages.

1.75 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 functions 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...

1.76 Regular FrexxEd Return Codes

In more than one place in the 'functions.guide' file, when talking about return codes from different functions, we refer to the "regular FrexxEd return code".

The return FrexxEd return code is simply FrexxEd's way of returning progress information. If has two 'levels':

```
 >= 0 (equal or greater than zero)
        Means that the function went fine.

 <  0 (less than zero)
        Means that the function failed or was cancelled in any way.
```

In addition to that information, the FPL programmer can of course take advantage of the GetErrNo() and ReturnStatus() function.

1.77 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 elsewhere. You may alter the 'fpl_path' info variable to make FrexxEd scan other directories when searching for FPL programs to run.

If you change an FPL program that was included in the FrexxEd original archive, we recommend you to keep the original in FrexxEd:FPL/ and to store your new version in another directory which is put before the FPL/ directory in the 'fpl_path' variable. The original program is likely to get re-stored at its place whenever you install an update!

1.78 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. (See the documentation template file, 'FrexxEd:FPL/xxxx.FPL.README'.)

```
    Document file format
    ====================
```

The header must start with a line with 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
```

1.79 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().

1.80 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()

1.81 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, design menus and much, much more! If you write your own FPL programs and want them to be inserted in the FrexxEd environment at every startup, such a file execution is perfect. 'FrexxEd:FPL/User.FPL' was added for that purpose only!

When the startup program (FrexxEd.default) 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
"

1.82 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 an FPL program!

Read more in the Functions.guide 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.

1.83 Using search and replace from FPL

To use of decent search and replace functions are not smaller ↩
from within FPL
programs than in regular interactive use, and FrexxEd supplies a powerful interface to fast search and replace functions.

* SEE
Using search and replace
Search/replace options
~~~~~~~~~~~~~~~~~~~~~~

When using search and replace interactively, the user can set the options by clicking on the different buttons of the search and replace request windows. FPL programs can of course alter those settings, but can also perform searches and replaces that ignore them.
~~~~~~~~~~~~~~~~~~~~~~

```
 The options are set in a string of the syntax <flags><operator>, where
<flags> are single characters for each flag and <operator> is '+' if the flags
to the left should get set, '-' if they should get cleared and '=' if they
should be the only flags set.

 Available flag-letters are:

   search_block    'b'
   search_case     'c'
   search_forward  'f'
   search_limit    'l'
   search_prompt   'p'
   search_wildcard 'w'
   search_word     'o'

 #1: SearchSet("fc");

 Sets the global search flags to 'search_forward' and 'search_case'.

 #2: Search("a");

 Will search for the first occurence of the string 'a' and use the settings
 already set by the user or previous programs.

 #3: Search("a", "f=");

 Will use the 'search_forward' setting only when searching (this does *not*
 alter the global settings).

 #4: Search("a", "f-");

 Will search as the settings tell, but with the 'search_forward' setting
 switched off (this does *not* alter the global settings).

 Read/Write options
 ~~~~~~~~~~~~~~~~~~
 All the mentioned search and replace options are also readable and writeable
as global, boolean info variables. They are named like above, with that
'search_' prefix to all names.

 In programs that do a lot of search and/or replace, it can be handy to first
set the options the program requires, and then when the program is finished,
restore the old options. It is handy to read from and write to the info
variable 'search_flags' which is all search flags represented as an integer.
Do not assumpt that the bits represent flags identical all the time.

SEE ALSO
  Search()
  Replace()
  SearchSet()
  ReplaceSet()
  ReplaceMatch()
```

1.84 Dealing with info variable

What’s an info variable

Info variables are in plain words numbers or strings associated with a name. The way FrexxEd uses to store and use internal preferences.

FrexxEd has a large amount of internal info variables, that can be read and referenced in the ’ change settings ’ chapter as well as in the function reference manual in the ReadInfo() function.

Any FPL program is also capable of creating, using and saving new custom info variables to be used by that and other FPL programs. This paragraph is intended to shortly describe some things to consider when making and using your own info variables.

How to change a custom info variable

"Custom" variables (that were created by an FPL program) are visible in the general customizing windows (brought up with PromptInfo()) and editable just like all the other are.

How to make one

With the ConstructInfo() function, an FPL program can create info variables. They can be given a lot of different characteristics. An info variable is refered to by its name, so first of all you should select what to call it. Use a plain english word that describes pretty good what it stands for, avoid short nonsense words.

- Name, any word of any length. It must be unique, not previously used in FrexxEd. The info variable name is case sensitive.

You must then decide whether the variable should be local or global:

- Local = Each buffer will have its own value/string in the variable
- Global = There is only one value/string for the entire FrexxEd

The info variable must have a type which is string, integer, cycle or boolean.

- Strings hold any length of characters in any combination
- Integers hold signed 32-bit numbers. An integer info variable can have a specified minimum and maximum number to prevent the user from selecting absurd values.
- Cycles hold a number within the specified range. When an info variable modification window is brought up, the differenct cycle values are represented by different strings.
- Booleans hold a number that is either non-zero or zero, like a digital switch on or off.

You can also set a few different attributes like

- Writeable. This info variable will get its value/string stored on disk together with the rest of the FrexxEd info variables. SetSave() is the function that causes them to get stored.
- Read-only. Such an info variable can only get read after it has been created.
- Hidden. A hidden info variable doesn’t show up in the normal customzing windows (the alter local/global info variable windows).

```
 Each setting can be sorted into a category to help the user filter out
interesting info variables
 - Display
 - IO = file handling stuff
 - Screen
 - System = misc things

 When you create the info variable, you should also consider one of the most
important things
 - Default value/string. The value/string that appears in the info variable
   after it has been created (and no value was stored in and read from the
   FrexxEd.default file).

 For the advanced user, ConstructInfo() features things like
 - FPLmodify = make a specified FPL program to get executed each time the
   created info variable gets changed. (See also chapter 'things to think of'
   below.)
 - FPLreq = makes this info variable get a button in a info variable modify
   window. When that button is pressed, the FPL program known as 'FPLreq'
   will be executed. The returned value/string will become the new contents
   of the info variable.

 How to alter one
 ~~~~~~~~~~~~~~~~
 SetInfo() is the function to use when changing the contents of an info
variable. To think of when dealing with local info variables is that you can
use SetInfo() to set the info variable of a specified buffer, of the current
buffer, of the default buffer or of all buffers.

 Saving info variable
 ~~~~~~~~~~~~~~~~~~~~
 When SetSave() is invoked, FrexxEd will generate an FPL program that will set
all internal info variables to their current value. All custom info variables
that were created as 'Writeable' will get their current value stored too. The
difference is that the custom info variables' values/strings are stored in a
way which makes FrexxEd use that value as default value as soon as the
variable is created.

 Things to think of
 ~~~~~~~~~~~~~~~~~~
 * Local info variable get their "default" value stored, not the value from
   the current or any other buffer (unless, of course, it is identical to the
   default's value/string). The default value is changed with
   SetInfo(0, ...).

 * The 'FPLmodify' program will get called *each* time the variable is
   changed. When using local variables, SetInfo() can set all buffers at one
   call. It will make the 'FPLmodify' program to get called once for every
   buffer in memory (including those hidden ones like 'DefaultBlock' and
   'DefaultYank').
```

1.85 Debugging FPL programs

```
 Bugs
 ~~~~
 All programs sometimes fail to do their work properly. By a typing or
structural mistake, the program simply doesn't do what the author wants it to.
Then, debugging is required.

 FPLdb
 ~~~~~
 FrexxEd is since the 1.5 release distributing the 'FPLdb' debugger program
within the FrexxEd release arhive (in the FrexxEd:bin/ directory) and by using
that little tool, you can get a pretty good idea what the FPL programs do.
 FPLdb outputs all interpreted lines of all programs that are run in debug-
mode.

 Debug-mode
 ~~~~~~~~~~
 By altering the 'fpl_debug' info variable inside FrexxEd, you make all FPL
programs execute in debug-mode. A program can also force that itself by
invoking the debug() function with a non-zero parameter.
 Only programs that run in debug-mode will produce any output in the FPLdb
window. All other programs run just as usual.

 Notes
 ~~~~~
 * Read more in the FPLdb.doc file!

 * Try not to run more than one program at a time in debug-mode with FPLdb
   activated as the output will become pretty confusing!

 * Programs can very well run in debug-mode even if FPLdb isn't started.

 * You are more than welcome to help us develop FPLdb. Things like break-
   points and variable watches can very well be made.
```

1.86 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

                BufferKill

                Exit

                FileChanged

                FPLError

                GetFile

                GotFile

                HandlerAction

                IconClick

                IconDrop

                MenuSelect

                ResizeWindow

                SameName

                SliderDrag

                WindowClose
```

1.87 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 merly 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
                "
```

1.88 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
                "
```

1.89 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
                "
```

1.90 Exit

```
                NAME    Exit

PARAMETERS
```

```
DESCRIPTION
        Generated when FrexxEd is exitting. This exception occurs when there
        is no turning back. FrexxEd is just about to quit, all buffers are
        killed, nothing should be done that deals with FrexxEd "internal
        affairs"!

NOTE
        Since all buffers are killed, you should not make any assumptions
        about the "current buffer" which won't be what you think!

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

1.91 FileChanged

```
                NAME    FileChanged

PARAMETERS

DESCRIPTION
        Generated when a file is to be saved, and the file on disk has been
        changed since it was last saved or loaded into FrexxEd.

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

1.92 FPLError

```
                NAME    FPLError

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
                "
```

1.93 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
                "
```

1.94 GotFile

```
                NAME    GotFile

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 has to been loaded into FrexxEd. The parameters
        sent to this function are the path name and the file name of the file
        that was loaded.
        Returning a non-zero value will *NOT* abort the loading of the file.
        Use the 'GetFile' exception for such matters! When using multiple
        selection or wildcard file loadings, this exception will always occur
        once for every loaded file.

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

1.95 HandlerAction

```
                NAME    HandlerAction

PARAMETERS
        int BufferID -  Buffer ID to the buffer which was effected by the
                        handler's action.
        int Action   -  Action specifier. What was done to the buffer/file.
                        See description for details.

DESCRIPTION
        Generated when a file/buffer is accessed in any way by the
        filehandler. Can be used to i.e starting things when particular files
        are read, created, written, deleted or similar.
        The BufferID received specifies the buffer the action was effecting.
        ("was", since the action has already occured when we get this
        exception) The action can't be prevented whatever the exeption
        function returns.

        Action specifiers are at the moment:
        1 - Read. Data was simply read from the buffer by the filehandler.
        2 - Write. Data was written to the buffer.
        3 - Open. The file was opened. This is a start signal that some
            action(s) may happen within the nearest future.
        4 - Close. This commonly means that the action is past for this
            buffer.
        5 - Create. The buffer has been created by the handler.
        6 - Delete. The buffer was just deleted by the handler! WARNING:
            The BufferID received does not point to any existant buffer. The
            buffer with the specified ID has already been deleted!!

NOTE
        When this exception happens, the current buffer ID WILL NOT be the
        same as the ID sent as parameter. The current ID will be the buffer
        ID of the buffer the cursor is currently in when this exception is
        generated!

SEE ALSO

                Exceptions

                Patching internal functions

                Filehandler
```

1.96 IconClick

```
                NAME    IconClick

PARAMETERS
        int Iconified - Deiconfied state TRUE/FALSE.

DESCRIPTION
        Generated when the appicon has been doubleclicked.

SEE ALSO
```

```
        "
                Exceptions
                ", "
                Patching internal functions
                "
```

1.97 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
                "
```

1.98 MenuSelect

```
                NAME    MenuSelect

PARAMETERS
        int Tnum - Title number
        int Inum - Item number
        int Snum - Subitem number

DESCRIPTION
        Generated when a menu item has been selected. The parameters sent
        with this exception are the same that can be used as parameters to
        i.e MenuRead(). Returning a non-zero value will abort the menu item
        execution.

EXAMPLE
        Make a little function that displays all menu execution programs in
        requesters before they are performed for real:

          export int menu_hook(int title_no, int item_no, int sub_no)
          {
            string type, str, program, key;
            MenuRead(&type, &str, &program, &key, title_no, item_no, sub_no);
            Request( program );
          }
          Hook("MenuSelect", "menu_hook");
```

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

1.99 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
                "
```

1.100 SameName

```
                NAME    SameName

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

DESCRIPTION
        Generated when a file is to be opened, and it uses the same name as
        another buffer already does. This exception sends the full file
        name and buffer id as parameters. Returning a non-zero value will
        abort the loading of the file.

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

1.101 SliderDrag

```
                NAME    SliderDrag

PARAMETERS
        none

DESCRIPTION
        Generated when the vertical slider is dragged.

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

1.102 WindowClose exception

```
                NAME    WindowClose

PARAMETERS

DESCRIPTION
        Generated when the close gadget of the window is pressed.

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

1.103 FrexxEd made easy

```
                           FREXXED FOR DUMMIES or QUICK GUIDE FOR BEGINNERS
           ================================================

 This document
 ~~~~~~~~~~~~~
 This document is aimed to you who haven't previously experienced FrexxEd.
Since this is an editor packed with advanced options and tricky features, it
is a lot to learn. This is a guide for you who have an interest in getting to
know the first basics.
 FrexxEd is kind of based on the programmability, but this chapter is meant to
avoid all talk about that and to concentrate at the first meeting with
FrexxEd.
 I will assume that you haven't done any big changes to the setup
environment.

 Installation
 ~~~~~~~~~~~~
 To be able to use FrexxEd, you must have it installed on your hard disk.
```

Since you read this, I assume that you already have done that.

Start-up
~~~~~~~~
Double-clicking on the FrexxEd icon will start FrexxEd. Shell users can enter "cd frexxed:" followed by "fred".

FrexxEd starts up on a screen. The screen mode and number of colors used by FrexxEd should be the same as your workbench screen uses.

Menu
~~~~
The menu strip attached to the FrexxEd screen has a custom way of specifying key qualifiers: "C" is for 'control', "A" is for 'amiga' and "Alt" is for 'Alt'. That makes a key sequence that looks like "AC-b" equal the key press 'amiga control b'. (Note that both the left and right qualifiers are valid.)

Default menu description
Preferences
~~~~~~~~~~~
The first and easiest things you can do to improve the look and feeling of FrexxEd is to change screenmode, select colors and change fonts. If you would prefer FrexxEd to start up in a window by default, you should alter the 'window' cycle gadget in the Customizing->Globals->Screen window and then resize and position the window as you want it to be at start-up.

Select "Customizing->Save" and accept the default file name.

Macros
~~~~~~
If you would like your FrexxEd to feature any macros at startup, you should

record the macros
as usual, and then save them in the FrexxEd:Startup/ drawer with .macro extensions. All those macros are loaded at startup time.

Requesters
~~~~~~~~~~
FrexxEd is using the reqtools.library for requester handling. If you would like to alter i.e the sizes (and other things) of the font/file requesters, you must use the reqtools prefs utility ((C) by Nico Francois).

Advanced setup
~~~~~~~~~~~~~~
Try the ARexx script FrexxEd:Rexx/ShowFPL.rx. It automatically generates an amigaguide document which you can use to install/deinstall FPL programs that your FrexxEd should execute at startup. FPL programs are run to extend the capabilities of FrexxEd.

1.104 Default menu setup

Default menu description
~~~~~~~~~~~~~~~~~~~~~~~~
The default menu is localized, and exists in several different languages.
~~~~~~~~~~~~~~~~~~~~~~~~

```
This document deals with the ones existing when this is written.

 The menu descrption will be put up like:

 * Menu title name (in english/svenska/nederlands/polski/deutch)
   *I Item name (in english/svenska/nederlands/polski/deutch)
       - English description
     *S Sub item name (in english/svenska/nederlands/polski/deutch)
       - English description

 from left to right, top to bottom...


 Menu titles
 ~~~~~~~~~~~
 *
               Project/Arkiv/Projekt/Projekt/Projekt
                *
               Block/Block/Blok/Blok/Block
                *
               Edit/Editera/Verander/Edycja/Bearbeiten
                *
               Move / Förflytta / Verplaats/ Ruch/Widok / Bewegen
                *
               Search/replace Sök/ersätt Vind Szukanie Suchen
                *
               Customizing/Inställningar/Veranderen/Ustawienia/Einstellungen
                *
               Special/Special/Speciaal/Specjalne/Spezial
```

1.105 Project menu title

```
                *I Clear/Rensa/Schoon/Wyczyôê/Löschen
     - This clear/empty the current buffer. If modified, a requester will
       appear, querying if this really is what you want.

 *I New/Ny/Nieuw/Nowy/Neu
     - A new empty buffer without name is created. (It will be visible as
       "*noname*" in the status line.)

 *I Open/Öppna/Openen/Otwórz/Öffnen
     - Pops up a file requester. The selected file is loaded into a new
       buffer.

     * SEE
               Loading files
                *I Load/Läs in/Laden/Wczytaj/Laden
     - Pops up a file requester. The selected file is loaded and replaces
       the current buffer. If the current buffer is modified, a requester
       will appear to ask if you really want to abandon your changes.
       If more than one file is selected, the first will be loaded into the
       current buffer, but the rest will be loaded into new buffers.

     * SEE loading files.
```

```
*I Insert file/Skjut in/Voeg file in/Doîâcz zbiór/Einfügen
    - Pops up a file requester. The selected files are inserted at the
      current position in the buffer.

*I Rename/Döp om/Hernoem/Zmieï nazwë/Umbenennen
    - Enter a new file name in the file requester. The current buffer will
      then get renamed to the entered name.

*I Save/Spara/Bewaar/Zapisz/Sichern
    - Store the contents of the buffer on disk. If the buffer is unnamed,
      a requester will appear, prompting for a file name to use.

    * SEE
               Saving files
                *I Save changes/Spara ändringar/Bewaar veranderingen/Zapisz  ←
                    zmiany/
   Änderungen sichern
    - Store the contents of the buffer on disk, but only if the buffer has
      been changed since last loaded or saved. If the buffer is unnamed,
      a requester will appear, prompting for a file name to use.

*I Save as/Spara som/Bewaar als/Zapisz jako/Sichern als
    - Store the contents of the buffer on disk in the specified file name.
      A file requester will appear to enable file name to get entered. If
      the specified file name already exists, you will be asked if that file
      should be overwritten.

*I Save All/Spara allt/Bewaar alles/Zapisz wszystkie/Alles sichern
    - Store all buffers in memory to disk.

*I Save all changes/Spara alla ändringar/Bewaar alle veranderingen/
   Zapisz wszystkie zmiany/Alle Änderungen sichern
    - Store all modified buffers in memory to disk.

*I Save project/Spara projekt/Bewaar projekt/Zapisz projekt/Projekt sichern
    - To save a project is to save information about all buffers currently
      in memory. Which buffers, cursor positions and other information is
      stored. This function prompts for a file name to use with a regular
      file requester. The projects do have an especially made drawer where
      they should be put for easy and fast project overview.

*I Load project/Läs in projekt/Laden projekt/Wczytaj projekt/
   Projekt laden
    - Load a project. Enter project name in the file requester and all
      will be set up like saved in the project file.

*I Print/Skriv ut/Print/Drukuj/Drucken
    - Sends the curent buffer to the printer.

*I Iconify/Ikonifiera/Iconify/Ikonifikacja/Iconifizieren
    - Close down the FrexxEd screen/window and make it exist as an AppIcon
      only. Doubleclick on the AppIcon to make the screen/window appear
      again.

*I About/Om FrexxEd/Over/O autorach/Über
    - Displays an information window. Tells about who made FrexxEd, how
      much memory FrexxEd has allocated and how much memory there is free
      in the system.

 *I Kill/Stäng/Sluiten/Zamknij okno/Schliessen
    - Kills and exits the current buffer. If the buffer is modified, you
      will be asked whether you really want to discard your changes.

 *I Quit all/Avsluta/Stoppen Alles/Zamknij wszystkie/Alles Beenden
    - Exists FrexxEd.
```

1.106 Block menu title

```
              *I Mark/Markera/Markeren/Zaznacz/Markieren
    - Starts or stops "regular" block marking.

    * SEE
             Block marking
              *I Mark rectangle/Markera rektangel/Markeer vierkant/Zaznacz  ↩
                 prostokât/
   Kolumne markieren
    - Starts or stops rectangular block marking.

 *I Cut/Klipp ut/Knip/Wytnij/Ausschneiden
    - Cuts the block out of the buffer. The marked block is removed and
      copied into the block buffer.

 *I Cut append/Klipp ut & lägg till/Knip bijvoegen/Wytnij i doîâcz/
   Ausschneiden & Anhängen
    - Cuts the block out of the buffer and appends it to the end of the
      block buffer.

 *I Copy/Kopiera/Kopieer/Skopiuj/Kopieren
    - Copies the marked block to the block buffer.

 *I Copy append/Kopiera & lägg till/Kopieer bijvoegen/Skopiuj i doîâcz/
   Kopieren & Anhängen
    - Copy the marked block to the end of the block buffer.

 *I Delete/Radera/Verwijder/Skasuj/Löschen
    - Deletes the marked block from the buffer. The block buffer is *not*
      affected.

 *I Insert/Skjut in/Invoegen/Wstaw/Einfügen
    - Insert the block buffer at the current position as a "regular" block.

 *I Insert rectangle/Skjut in rektangel/Invoegen vierkant/Wstaw prostokât/
   Kolumne einfügen
    - Insert the block buffer at the current position as a rectangle block.

 *I Load/Läs in/Laden/Wczytaj/Laden
    - Brings up a file requester and loads the specified file into the
      block buffer.

 *I Save/Spara/Bewaar/Zapisz/Sichern
    - Brings up a file requester and stores the current block buffer or
      marked block as a file with the specified name.
```

```
*I Print/Skriv ut/Print/Drukuj/Drucken
    - Sends the marked block to the printer.

*I Edit blocks/Editera block/Verander bloks/Edycja bloku/Blöcke bearbeiten
    - Lets the user select which block buffer to edit. Then makes that
      block buffer visible in the window and enables editing in it. By
      default, there will only exist one block buffer called DefaultBlock.

*I Export to clipboard/Exportera till klippotek/Exporteer naar clipboard/
   Wstaw do clipboard'a/Clipboard exportieren
    - Exports the marked block or the block buffer to the clipboard. Thus
      enabling other programs to access the text.
      NOTE: only available in the
               registered version
               .

*I Import from clipboard/Importera från klippotek/Importeer van clipboard/
   Pobierz z clipboard'a/Clipboard importieren
    - Imports the clipboard contents to the block buffer.

*I Move/Indentera/Verplaats/Przesuï/Verschieben
    - Starts an interactive mode in which you can move the marked block
      horizontally by using the cursor and tab keys. Cancel the operation
      with escape or q, and put the block on the selected position by
      pressing return.

*I Sort/Sortera/Sorteer/Posortuj/Sortieren
    - Brings up a small requester in which you can decide how to sort the
      block buffer or marked block. The field number is which white space
      separated field it should sort according to, forward means in a-z
      order and case sensitive makes difference between 'A' and 'a'.

*I Change case/Byt gemen/versal/Verander hoogte/Zamieï wielkoôê liter
   Groß/Klein wechseln
    - Changes case on the letters in the marked block. Marking a block
      holding the text "Hello" and invoking this function will convert the
      text to "hELLO".

*I Lower case/Till gemener/Verlaag hoogte/Zamieï na maîe litery/
   Block Groß
    - Converts the letters in the marked block to uppercase. If the text
      "Hello" is marked before invoking this function, it will convert the
      text to "HELLO".

*I Upper case/Till versaler/Verhoog hoogte/Zamieï na duûe litery/
   Block Klein
    - Converts the letters in the marked block to lowercase. If the text
      "Hello" is marked before invoking this function, it will convert the
      text to "hello".
```

1.107 Edit menu title

```
*I Delete line/Radera rad/Verwijder regel/Skasuj linië/Zeile ↩
   löschen
```

```
     - Removes the current line from the buffer.

     * SEE
              Erasing text
               *I Delete to EOL/Radera till slut på rad/Verwijder naar EVR/
    Skasuj do koïca linii/Bis Zeilenende löschen
     - Removes all characters to the end of the current line.

 *I Delete word/Radera ord höger/Verwijder woord/Skasuj sîowo/
    Wort löschen
     - Deletes the word immediately under/to the right of the cursor.

 *I Backspace word/Radera ord vänster/Backspace woord/
    Skasuj sîowo z lewej/Wort links löschen
     - Deletes the word immediately under/to the left of the cursor.

 *I Fold/Veck
     - Contains a submenu with all the fold-related menu items
     * See
              Using folds
                 *S Delete/Radera
     - Remove the fold at the current position, and bring back the text
       to the normal state.

   *S Show/Visa
     - Display the contents of the fold at the current line.

   *S Hide/göm
     - Hide the fold at the current line

   *S Show exclusive/Visa enbart
     - Show only the contents of this fold.

   *S Hide exclusive/Göm enbart
     - Abort the 'show exclusive' mode.

   *S Show all/Visa alla
     - Show all folds.

   *S Hide all/Göm alla
     - Bring all folds to the hidden state.

 *I Undo/Ångra/Maak ongedaan/Cofnij zmiany/Rückgängig machen
     - Undoes the latest change. Repeated invokes will undo earlier changes.

     * See
              Undo concepts
               *I Undo restart/Ångra omstart/Herstart ongedaan maken/Undo  ↩
                  Restart/
    Neu Rückgängig
     - Restart the undo session. This enables the user to start undoing
       the changes done by recent 'undo' invokes.

 *I Yank/Skjut in slask/Yank/Yank/Löschpuffer
     - Inserts the last sequence of deleted letters. Character deletes that
       removes more than one character add the deleted string to the yank
       buffer.
```

```
*I Insert buffer/Skjut in buffer/Invoegen buffer/Wstaw bufor/
   Puffer einfügen
    - Pops up a requester asking for which buffer to insert at the current
      position.
```

1.108 Move menu title

```
                *I Maximize view/Maximera vy/Maximale grote scherm/Powiëksz widok ↩
                   /
   Ansicht maximal
    - Make the current view the only view in the window.

    * SEE
            View concepts
                *I Remove view/Stäng vy/Verwijder scherm/Usuï widok/Ansicht  ↩
                   entfernen
    - Hide this view. This only makes the view dissapear. The buffer will
      remain in memory, just not visible.

*I Resize view/Ändra storlek på vy/Verander scherm/Zmieï rozmiar/
   Größe ändern
    - If you by some reason don't want to drag in the status line to resize
      the current buffer. Selecting this menu item lets you enter the resize
      manually.

*I Goto next view/Nästa vy/Ga naar volgende view/Pokaû nastëpny/
   Nächste Ansicht
    - Jumps to the next view in the window. If there is only one view,
      nothing will happen.

    * SEE
            Using multiple buffers
                *I Goto previous view/Föregående vy/Ga naar vorige view/
   Pokaû poprzedni/Vorige Ansicht
    - Jumps to the previous view in the window. If there is only one view,
      nothing will happen.

*I Next buffer/Nästa buffer/Volgende buffer/Nastëpny bufor/
   Nächster Puffer
    - Jumps to the next buffer in the list. If the buffer isn't visible,
      the current view will display the next buffer. If the buffer is
      visible, the cursor will be moved there.

*I Previous buffer/Föregående buffer/Vorige buffer/
   Poprzedni bufor/Voriger Puffer
    - Jumps to the previous buffer in the list. If the buffer isn't visible,
      the current view will display the previous buffer. If the buffer is
      visible, the cursor will be moved there.

*I Split view/Dela vy/Splits scherm/Rozdziel widok/Ansicht teilen
    - Make two views display this buffer. Each view gets its own cursor
      position, block mark, etc.

*I Right word/Nästa ord/Rechts van woord/
```

```
   Przesuï sië o sîowo w prawo/Wortschritt rechts
    - Moves the cursor one word to the right.

 *I Left word/Föregående ord/Links van woord/
   Przesuï sië o sîowo w lewo/Wortschritt links
    - Moves the cursor one word to the left.

 *I Match character/Motsvarande tecken/Vergelijk karakter/
   Znajdú odpowiadajâcy znak/Zeichenpaar suchen
    - Moves the cursor to the character that matches the one currently
      under the cursor. Matching characters are by default (), [], <> and
      {}.

 *I Goto buffer/Gå till buffer/Ga naar buffer/Przejdú do bufora/
   Puffer wählen
    - Moves the cursor to the buffer selected in the requester. If the
      buffer isn't visible, the current view will be made to display it.
      If the buffer is visible, the cursor will be put in that view.

 *I Goto line/Gå till rad/Ga naar regel/Przejdú do linii/
   Zeile wählen
    - Moves the cursor to the line entered in the requester.

 *I Goto latest change/Gå till senaste ändring/
   Ga naar laatste verandering/Przejdú do ostatniej zmiany/
   Letze Änderung
    - Moves the cursor to the position of the last change. Repeated invokes
      move the cursor to earlier changes.
      NOTE: Depending on what the changes actually were, the positions will
      not be exact.
```

1.109 Search menu title

```
                *I Repeat search forward/Repetera sökning framåt/
   Herhaal vinden voorwaards/Powtórz szukanie w przód/
   Vorwärtssuche wiederh.
    - Continue to search forward for the search string. If no previous
      search has been done, the search requester will appear for the user
      to fill in.

    * SEE
               Search and replace
                *I Repeat search backward/Repetera sökning bakåt/
   Herhaal vinden achterwaards/Powtórz szukanie wstecz/
   Rückwärtssuche wiederh.
    - Continue to search backward for the search string. If no previous
      search has been done, the search requester will appear for the user
      to fill in.

 *I Search/Sök/Vind/Szukaj/Suchen
    - Bring up the search requester. When 'OK' has been pressed, it searches
      according to the selections.

 *I Repeat replace/Repetera ersättning/Herhaal vervangen/
   Powtórz zamianë/Ersetzen wiederholen
```

```
    - Continue a previous replace session. If no previous replace has been
      donw, the replace requester will appear for the user to fill in.

⋆I Replace/Ersätt/Vervang/Zamieï/Ersetzen
    - Bring up the replace requester. When 'OK' has been pressed, it
      replaces according to the selections.

⋆I Block to search buffer/Block till söktext/Blok naar vind buffer/
   Blok do bufora poszukiwaï/Block in Suchpuffer kopieren
    - Copies the contents of the marked block or block buffer to the search
      buffer. A following search (or replace) will then search for that
      text.

⋆I Block to replace buffer/Block till ersättningstext/
   Blok naar vervang buffer/Blok do bufora zamiany/
   Block in Ersatzpuffer kopieren
    - Copies the contents of the marked block or block buffer to the replace
      buffer. A following replace will then replace the search string with
      that text.
```

1.110 Customzing menu title

```
                ⋆I Colours/Färger/Kleuren/Kolory/Farben

  ⋆S Adjust/Justera/Verander/Ustaw/Einstellen
    - Brings up a palette requester enabling the user to change the
      palette used in the FrexxEd screen.

  ⋆S Reset/Återställ/Reset/Ustawienia poczâtkowe/Zurücksetzen
    - Resets the colours back to the colors set in the system preferences.

⋆I Locals/Lokala/Lokale/Lokalne/Lokale Einst.
    - Brings up a requester window that hold buffer unique preferences.
      Altered preferences affect the current buffer only.

⋆I Default locals/Initiallokala
    - Brings up a requester window with the buffer unique preferences. When
      'OK' is selected, the preferences is set to the default values. They
      are the ones stored when the preferences are saved and newly created
      buffers will get their values from those same default values.

⋆I Globals/Globala/Globalen/Globalne/Globale Einst.

  ⋆S Screen/Skärm/Scherm/Ekran/Bildschirm
    - Brings up a requester window with the global screen preferences.

  ⋆S IO/IO-hantering/IO/ Operacje we/wy Ein-/Ausgabe
    - Brings up a requester window with the global IO preferences.

  ⋆S Display/System Vyer/System Scherm/Systeem Wyôwietlanie/System
     Bildschirm/System
    - Brings up a requester window with the global display and system
      preferences.

  ⋆S All globals/Alla globala/Alle globale/Wszystkie globalne/
```

```
     Alle globalen Einst.
    - Brings up a requester window with all global preferences. If the
      screen size is small or if the font size is big (or both), this window
      may not fit on your screen. Cancel the window by pressing escape and
      use one of the above three specialized windows in those cases.

*I Program/Program/Program/Program/Program
    - This menu item doesn't do anything! It is here for one reason only:
      When custom FPL programs is run, they should add their own preference
      requesters as sub items to this menu item.

*I Save/Spara/Bewaar/Zapisz/Sichern
    - Brings up a file requester and saves the default setup with that file
      name. That file name is by default called FrexxEd:FPL/FrexxEd.default
      and that will be used on each normal FrexxEd startup.

    * SEE
               Change start-up environment
                *I Font/Teckensnitt/Font/Czcionka/Zeichensatz

  SEE
               Change fonts
                  *S System font/Systemtecken/Systeem font/Systemowa/ ↩
                     Systemzeichensatz
    - This lets the user select which fixed width font FrexxEd should use
      as 'system' font. It is used in the main text screen and status line.

  *S Request font/Requesttecken/Verzoek font/Ûâdaï systemowych/
     Fensterzeichensatz
    - This lets the user select which font FrexxEd should use as 'request'
      font. It is used in the menu and all requesters.

  *S Both fonts/Båda/Allebei de fonts/Obie/Beide Zeichensätze
    - This lets the user select which fixed width font FrexxEd should use.
      This function sets both system and request fonts at the same time.

*I Screenmode/Skärmläge/SchermMode/Tryb ekranu/Bildschirmmodus
    - Brings up a screenmode requester and lets the user select in which
      screenmode the FrexxEd screen should run. When FrexxEd is used as
      window, this function sets the mode to use next time FrexxEd is used
      as a screen!

*I Clone WB/Kopiera WB/Kopieer WB/Skopiuj WB/Kopieren WB
    - Clones the Workbench screen. This makes the FrexxEd screen look as
      similar as the WB screen as posssible. It copies screen size, number
      of colors, the pallette, the fonts, etc. If FrexxEd is used as a
      window on a public screen, this function will clone everything from
      the host screen which FrexxEd is on.

*I Reset FACT/Återställ FACT/Reset FACT/Ustawienie poczâtkowe FACT/
   FACT zurücksetzen
    - Reset the FACT to internal default. The FACT is the internal control
      table which controls how differenct ASCII codes should be represented
      on screen.

    * SEE
               Character set
```

```
                 and
                Change the FACT
```

1.111 Special menu title

```
                 *I Macro record/Makroinspelning/Makro opnemen/Nagrywanie makra/
    Makro aufnehmen
     - Starts or stops macro recording. To record a macro is to make FrexxEd
       remember a sequence of actions and the ability to re-play those
       actions again when the given key sequence is pressed.

     * SEE
                Using macros
                 *I Edit macros/Editera makro/Verander makros/Edycja makra/Makros ↩
                    bearbeiten
     - Lets the user select a macro and displays it on screen. Thus enabling
       editing of the macro.
       NOTE: This requires at least some knowledge of the programming
       language FPL.

     * SEE
                Programming FrexxEd
                 *I Load macro/Load macro/Load macro/Wczytaj makro/Makro laden
     - Loads a previously saved macro. The file name is entered in a regular
       file requester.

 *I Save macro/Save macro/Save macro/Zapisz makro/Makro speichern
     - Saves a previously recorded macro. Which macro to save is selected in
       a list requester and the file name is entered in a regular file
       requester.

 *I Execute buffer as FPL/Kör buffer som FPL/Voer buffer als FPL uit/
    Wykonaj bufor/Puffer als FPL ausführen
     - Runs the contents of the current buffer as an FPL program.
       NOTE: This requires at least some knowledge of the programming
       language FPL.

     * SEE
                Programming FrexxEd
                 *I Execute block as FPL/Kör block som FPL/Voer blok als FPL uit/
    Wykonaj blok/Block als FPL ausführen
     - Runs the contents of the marked block or block buffer as an FPL
       program.
       NOTE: This requires at least some knowledge of the programming
       language FPL.

 *I Execute FPL file/Kör FPL program/Voer FPL bestand uit/
    Wykonaj zbiór/Datei als FPL ausführen
     - Lets the user select a file that is run as an FPL program.
       NOTE: This requires at least some knowledge of the programming
       language FPL.

 *I Prompt/Prompter/Prompt/Prompt/Befehle
     - Brings up a requester holding a list with all FPL functions available
       at the moment. A full FPL program can be entered in the input field,
       or a function that accepts invoke without parameter can be
       doubleclicked.
       NOTE: This requires at least some knowledge of the programming
       language FPL.

     * SEE
                Running commands by name
                 *I Assign key/Tilldela tangent/Ken toets toe/Tastenzuweisung
     - Enables the user to assign a FPL program execution to a key sequence.
       NOTE: This requires at least some knowledge of the programming
       language FPL.

     * SEE
                Change keys
                 *I Fastkey buffer exec/Snabbexekvera buffer/Puffer Taste zuweisen
     - Lets the user assign a key sequence that will execute the current
       buffer as an FPL program.
       NOTE: This requires at least some knowledge of the programming
       language FPL.

 *I Buffer info/Bufferinformation/Buffer informatie/Informacje o buforze/
    Pufferinformation
     - Displays information about the current buffer in a window. The
       information includes size, tab size, changes, file name, file path,
       date and other data.

 *I Help/Hjälp/Hilfe

   *S General/Allmänt/Allgemein
     - Starts 'amigaguide' displaying Frexxed:docs/FrexxEd.guide.

   *S Functions/Funktioner/Funktionen
     - Starts 'amigaguide' displaying Frexxed:docs/Functions.guide.
```

1.112 Index

                Index of database FrexxEd

Documents