

Functions

Daniel Stenberg

COLLABORATORS

	<i>TITLE :</i> Functions		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Daniel Stenberg	July 25, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Functions	1
1.1	Function documentation	1
1.2	FPL Function Reference	1
1.3	Function description syntax	5
1.4	About()	6
1.5	Activate()	7
1.6	ARexxRead()	7
1.7	ARexxResult()	8
1.8	ARexxSend()	8
1.9	ARexxSet()	9
1.10	AssignKey()	10
1.11	Backspace()	12
1.12	BackspaceWord()	13
1.13	BlockChange()	13
1.14	BlockCopy(), BlockCut(), BlockDelete()	14
1.15	BlockCreate()	15
1.16	BlockMark()	16
1.17	BlockMove()	18
1.18	BlockPaste()	18
1.19	BlockSort()	19
1.20	BSearch()	20
1.21	CConvertString()	21
1.22	CenterView()	21
1.23	Check()	22
1.24	Clean()	22
1.25	Clear()	23
1.26	ClipToString()	23
1.27	CloneWB()	24
1.28	ColorAdjust()	24
1.29	ConstructInfo()	25

1.30 CopyInfo()	27
1.31 ColorReset()	28
1.32 CurrentBuffer()	29
1.33 CursorActive()	29
1.34 Cursor movements	29
1.35 CursorStack	30
1.36 Delay()	31
1.37 Deiconify()	31
1.38 Delete()	32
1.39 DeleteEol()	32
1.40 DeleteInfo()	33
1.41 DeleteKey()	33
1.42 DeleteLine()	34
1.43 DisplayBeep()	34
1.44 DuplicateEntry()	35
1.45 Change case	35
1.46 Execution functions	36
1.47 FaceAdd()	37
1.48 FaceGet()	38
1.49 FaceRead()	39
1.50 FaceStyle()	40
1.51 FACT()	41
1.52 FACTClear()	42
1.53 FACTCreate()	43
1.54 FACTDelete()	44
1.55 FACTConvertString()	44
1.56 FACTString()	45
1.57 FindPort()	45
1.58 Fold()	46
1.59 FoldDelete()	46
1.60 FoldHide()	47
1.61 FoldShow()	48
1.62 GetBlock()	48
1.63 GetBufferID()	49
1.64 GetByte()	49
1.65 GetChar()	50
1.66 GetCursor()	51
1.67 GetDate()	51
1.68 GetEnv()	52

1.69 GetErrNo()	53
1.70 GetFileList()	53
1.71 GetKey()	54
1.72 GetLine()	54
1.73 GetList()	55
1.74 GetReturnMsg()	57
1.75 GetWindowID()	57
1.76 GetWord()	57
1.77 GotoChange()	58
1.78 GotoLine()	59
1.79 Hook functions	60
1.80 HookClear()	61
1.81 Iconify()	62
1.82 InsertFile()	62
1.83 InverseLine()	63
1.84 FACT reading functions	64
1.85 Isfold()	64
1.86 KeyPress()	65
1.87 Kill()	66
1.88 Load()	66
1.89 LoadString()	67
1.90 LogSetting()	68
1.91 MacroRecord()	68
1.92 MatchParen()	69
1.93 MaximizeView()	69
1.94 MenuAdd()	70
1.95 MenuBuild()	71
1.96 MenuClear()	72
1.97 MenuDelete()	72
1.98 MenuRead()	73
1.99 New()	74
1.100NextBuffer()	74
1.101NextEntry()	75
1.102NextView()	76
1.103NextWindow()	76
1.104Output()	77
1.105PageDown()	77
1.106PlaceCursor()	78
1.107PrevWindow()	78

1.108PrintLine()	79
1.109Prompt()	79
1.110PromptBuffer()	80
1.111PromptFile()	81
1.112PromptFont()	82
1.113Prompt for integer or string	82
1.114PromptInfo()	83
1.115Random()	85
1.116RedrawScreen()	86
1.117QuitAll()	86
1.118ReadInfo()	87
1.119RemoveView()	93
1.120Rename()	93
1.121Replace()	94
1.122ReplaceMatch()	94
1.123ReplaceSet()	95
1.124Request()	97
1.125RequestWindow()	97
1.126ResizeView()	99
1.127ReturnStatus()	99
1.128Save()	100
1.129SaveString()	100
1.130Screenmode()	101
1.131Scroll up or down	102
1.132Search()	102
1.133SetEnv()	103
1.134SetInfo()	103
1.135SetSave()	104
1.136Sort()	105
1.137Status()	106
1.138Stcgfn() and Stfgfp()	106
1.139Stricmp()	107
1.140StringChangeCase(), StringToLower(), StringToUpper()	108
1.141StringToBlock()	108
1.142StringToClip()	109
1.143Strmfp()	110
1.144System()	110
1.145TimerAdd()	111
1.146TimerDelete()	111

1.147Undo()	112
1.148UndoRestart()	112
1.149Visible()	113
1.150WindowAlloc()	113
1.151WindowToFront()	114
1.152Yank()	115

Chapter 1

Functions

1.1 Function documentation

```
#####

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

Latest update 8.11.96, for version 2.0

Function documentation

Copyright (C) by FrexxWare 1996

1.2 FPL Function Reference

All FPL functions that exist internally in FrexxEd are listed here. You might find functions that are not in this list, but then they are not internal FrexxEd functions, but added by FPL-programs.

First, a list of all functions with a short description:

Function	Page
----------	------

About - Shows information about FrexxEd.....	7
Activate - Visualize a certain buffer.....	8
ARexxRead - Get the contents of an ARexx variable.....	9
ARexxResult - Send back result string to ARexx.....	10
ARexxSend - Send a string to an ARexx port.....	11
ARexxSet - Set the contents of an ARexx variable.....	12
AssignKey - Assigns a function to a key sequence. [*].....	13
Backspace - Kill characters to the left of the cursor.....	16
BackspaceWord - Kill words to the left of the cursor.....	17
BlockChange - Change default block.....	18
BlockCopy - Copy a marked block.....	19
BlockCopyAppend - Copy marked area to the end of a block.....	19
BlockCreate - Create a new block. [*].....	20
BlockCut - Cut a marked block.....	19
BlockCutAppend - Move marked area to the end of a block.....	19
BlockDelete - Delete marked area.....	19
BlockMark - Mark an area in a view.....	21
BlockMarkLine - Mark full width of an area in a view.....	21
BlockMarkRect - Mark a rectangular area in a view.....	21
BlockMove - Moves a marked block area horizontally.....	23
BlockPaste - Insert a block in the buffer.....	24
BlockPasteRect - Insert block as rectangular.....	24
BlockSort - Sort a block.....	25
BSearch - Binary search in a string array.....	26
CConvertString - Convert a raw string to C style. [*].....	27
CenterView - Center the view vertically around the cursor.....	28
Check - Returns file information.....	29
Clean - Execution without hooks. [*].....	30
Clear - Clear a buffer.....	31
ClipToString - Return clipboard.device data as a string. [*]..	32
CloneWB - Clone the current public screen.....	33
ColorAdjust - Adjust colors.....	34
ColorReset - Reset color(s) to workbench default. [*].....	39
ConstructInfo - Create a new info variable. [*].....	35
CopyInfo - Copy info variables.....	37
CurrentBuffer - Change current buffer.....	40
CursorActive - Alter the cursor state.....	41
CursorDown - Moves cursor down.....	42
CursorLeft - Moves cursor left.....	42
CursorLeftWord - Moves cursor entire words left.....	42
CursorRight - Moves cursor right.....	42
CursorRightWord - Moves cursor entire words right.....	42
CursorStack - Stores/gets cursor position.....	43
CursorUp - Moves cursor up.....	42
Deiconify - Pops up FrexxEd from the icon state.....	45
Delay - Sleep for a while. [*].....	44
Delete - Delete char(s) at cursor.....	46
DeleteEol - Delete the rest of the line.....	47
DeleteInfo - Delete an info variable.....	48
DeleteKey - Delete a key assigning.....	49
DeleteLine - Delete a line.....	50
DeleteWord - Delete word(s) at cursor.....	46
DisplayBeep - Flash the screen.....	51
DownCase - Downcase a block.....	53
DuplicateEntry - Make two entries from one.....	52
ExecuteBuffer - Execute a buffer as an FPL program.....	54

ExecuteFile - Execute a file as an FPL program. [*].....	54
ExecuteLater - Execute a string as an FPL program later.....	54
ExecuteString - Execute a string as an FPL program. [*].....	54
FaceAdd - Add words to a face style.....	55
FaceGet - Get a named face.....	57
FaceRead - Read face style characteristics.....	58
FaceStyle - Get a named face style.....	59
FACT - Change the FrexxEd ASCII Convert Table. [*].....	60
FACTClear - Reset the current FACT to defaults. [*].....	62
FACTConvertString - Convert a string with the FACT.....	65
FACTCreate - Create a new FACT.....	63
FACTDelete - Delete a FACT.....	64
FACTString - Returns the FACT string of a character. [*].....	66
FindPort - Find a given system message port. [*].....	67
Fold - Fold an area of a buffer.....	68
FoldDelete - Delete a fold.....	69
FoldHide - Hide a fold.....	70
FoldShow - Show a fold.....	71
GetBlock - Get block as string.....	72
GetBufferID - Get ID of a buffer.....	73
GetByte - Get byte position from a given column.....	74
GetChar - Get character from buffer.....	75
GetCursor - Return column position of a byte position.....	76
GetDate - Return date.....	77
GetEntryID - Get ID of an entry.....	73
GetEnv - Returns the value of an environment variable.....	78
GetErrNo - Return error number. [*].....	79
GetFileList - Get a list of file matching a pattern.....	80
GetKey - Get keypress from user.....	81
GetLine - Return a line from buffer.....	82
GetList - Get listed information from FrexxEd.....	83
GetReturnMsg - Get verbose error description. [*].....	85
GetWindowID - Get WindowID of window or buffer.....	86
GetWord - Get current word.....	87
GotoChange - Go to a previous change.....	88
GotoLine - Go to a certain line and column.....	89
Hook - Patch a FrexxEd function. [*].....	90
HookClear - Clear hooks. [*].....	92
HookPast - Patch a FrexxEd function. [*].....	90
Iconify - Sets FrexxEd in icon state.....	93
InsertFile - Insert a file in buffer.....	94
InverseLine - Inverse graphics on part of line.....	95
Isclose - Check if a character is a close. [*].....	96
Isfold - Check for a fold on a line.....	97
Islower - Check if a character is lower case. [*].....	96
Isnewline - Check if a character is newline. [*].....	96
Isopen - Check if a character is an open. [*].....	96
Isspace - Check if a character is space. [*].....	96
Issymbol - Check if a character is a symbol. [*].....	96
Istab - Check if a character is tab. [*].....	96
Isupper - Check if a character is upper case. [*].....	96
Isword - Check if a character is word type. [*].....	96
KeyPress - Returns FPL program assigned to key.....	98
Kill - Kill a buffer.....	99
Load - Load buffer(s).....	100
LoadString - Load a file and return as string. [*].....	101
LogSetting - Log a variable's default setting. [*].....	102

MacroRecord - Record a macro.....	103
MatchParen - Jump to matching delimiter.....	104
MaximizeView - Make a view alone on screen.....	105
MenuAdd - Add an item to the menu list. [*].....	106
MenuBuild - Builds a menu from the menu list.....	108
MenuClear - Clears the menu list.....	109
MenuDelete - Deletes a menu item.....	110
MenuRead - Read details from the menu.....	111
New - Create a new buffer.....	112
NextBuffer - Move to next buffer.....	113
NextEntry - Move to next entry.....	114
NextHidden - Move to next hidden buffer.....	114
NextView - Move to next view.....	115
NextWindow - Move to next window.....	116
Output - Print text in buffer.....	117
PageDown - Move down a screenful of lines.....	118
PageUp - Move up a screenful of lines.....	118
PlaceCursor - Place cursor on screen.....	119
PrevBuffer - Move to previous buffer.....	113
PrevEntry - Move to previous entry.....	114
PrevHidden - Move to previous hidden buffer.....	114
PrevView - Move to previous view.....	115
PrevWindow - Move to the previous window.....	120
PrintLine - Print a string on the screen.....	121
Prompt - Interactive FPL program execution.....	122
PromptBuffer - Get buffer from user.....	123
PromptEntry - Get entry from user.....	123
PromptFile - Get file name from user.....	124
PromptFont - Get font from user.....	125
PromptInfo - Bring up an info variable window.....	127
PromptInt - Get integer from user.....	126
PromptString - Get string from user.....	126
QuitAll - Quit FrexxEd. [*].....	132
Random - Get a random number.....	130
ReadInfo - Get buffer information. [*].....	133
RedrawScreen - Update the screen image.....	131
RemoveView - Take away a view from screen.....	139
Rename - Change name of the buffer.....	140
Replace - Replaces strings with other strings in buffer.....	141
ReplaceMatch - Match pattern and generate replacement.....	142
ReplaceSet - Set search string, replace string and options....	143
Request - Presents a multi-button requester.....	144
RequestWindow - User defined request window.....	145
ResizeView - Make a view change size.....	147
ReturnStatus - Status line message at return.....	148
Save - Write buffer to disk.....	149
SaveString - Store string on disk. [*].....	150
Screenmode - Change screenmode.....	151
ScrollDown - Scroll down screen.....	152
ScrollUp - Scroll up screen.....	152
Search - Search for string.....	153
SearchSet - Set search string, replace string and options....	143
SetEnv - Set the value of an environment variable.....	154
SetInfo - Set info variables. [*].....	155
SetSave - Save settings. [*].....	156
Sort - Sort - Sort an array.....	157
Status - Write text on the status line.....	158

Stcgfn - Returns the file name part of an entire path string..	159
Stcgfp - Returns the path part of an entire path string.....	159
Stricmp - Compare strings case insensitive. [*].....	160
StringChangeCase - Change case of string.....	161
StringToBlock - Copy string to block. [*].....	162
StringToBlockAppend - Append string to block. [*].....	162
StringToClip - Copy string to clipboard. [*].....	163
StringToLower - Lower case string.....	161
StringToUpper - Upper case string.....	161
Strmfp - Merge a path string with a file string.....	164
SwapCase - Swap case of a block.....	53
System - Perform a system command. [*].....	165
TimerAdd - Start timer execution.....	166
TimerDelete - Remove a timer execution.....	167
Undo - Undos changes.....	168
UndoRestart - Restart an undo session.....	169
UpCase - Uppercase a block.....	53
Visible - Turn visualization on or off.....	170
WindowAlloc - Allocate a window data.....	171
WindowClose - Close a window.....	171
WindowFree - Free window data.....	171
WindowOpen - Open a window.....	171
WindowToFront - Bring FrexxEd to front.....	172
Yank - Paste the yank buffer.....	173

FPL function reference

1.3 Function description syntax

Following here is all functions that FrexxEd includes, with extensive and verbose descriptions, including usage syntax, return values and often also with examples of how to use.

The functions will be described using the following pattern:

NAME

<function name> - one line description. Functions programmed in FPL but included in the default startup configuration will be marked with a "[FPL]" symbol, and functions that are available already in the startup function are marked with a "[*]" symbol.

SYNOPSIS

```
progress = <function name> (parameters);
```

```
int = <function name> (<parameter types>);
```

(The <parameter type> is written in all uppercase letters for parameters that are optional - possible to leave out when calling the described function. Other, required, parameters are written in common lowercase letters.)

FUNCTION

Long description describing exactly what the function does, known drawbacks, if it can be called with a variable number of parameters and other things you need to know when calling the function.

CRIPPLE

If this keyword is present, it means that this function does not work 100% without a keyfile. The text will describe the effects of not having a keyfile when running the function.

INPUTS

Description of the function input parameter(s).

RESULT

What the function returns.

EXAMPLE

If this keyword is present, a short FPL program example is visualized that describes even more how to use this particular function.

SEE ALSO

References to other functions or further documentation which have similar functionalities or provide more information.

BUGS

Any information regarding known bugs, flaws or drawbacks that comes with using the function.

1.4 About()

NAME

About - Shows information about FrexxEd.

SYNOPSIS

```
About();
```

```
void About(void);
```

FUNCTION

This function pops up a window holding information regarding FrexxEd. It tells about who wrote it, how to pay the ShareWare fee, the name of this FrexxEd's ARexxport and how much memory there is available in the system respective has been allocated (used) by FrexxEd (this last figure does only include the amount of memory actually allocated by FrexxEd itself, not including the memory different resources will occupy when FrexxEd initializes such).

This information is localized and will produce a different output in different languages.

INPUTS**RESULT****SEE ALSO**

How to reach us

1.5 Activate()

NAME

Activate - Visualize a certain buffer.

SYNOPSIS

```
BufferID = Activate ( BufferID, Mode, PopupOnto );
```

```
int Activate ( INT, INT, INT );
```

FUNCTION

Forces the specified or current buffer to get visualized in a view or a new window. The view will appear as set in the Mode parameter or as set in 'popup_view'. The third parameter specifies which buffer it should split/replace. This function can be called with none to three parameters.

INPUTS

BufferID - Buffer ID number of the buffer you want to visualize. 0 or illegal IDs activate current buffer.

Mode - How to bring up the view.

0 = Replace the current view.

1 = Split the current view.

2 = Make this the only view on screen.

3 = Create a new window to show it in

-1 = As the 'popup_view' info variable tells!

PopupOnto - Buffer ID of the entry that should be replaced/split when the 'BufferID' buffer gets viewed.

RESULT

Returns the BufferID.

SEE ALSO

GetBufferID(), CurrentBuffer()

1.6 ARexxRead()

NAME

ARexxRead - Get the contents of an ARexx variable.

SYNOPSIS

```
Contents = ARexxRead ( Variable );
```

```
string ARexxRead ( string );
```

FUNCTION

When FrexxEd has received a message from an ARexx program, this function returns the contents of a named ARexx variable.

INPUTS

Variable - The name of the ARexx variable. It must be specified in all uppercase letters.

RESULT

The string the variable holds, or an empty string ("") if the variable didn't exist or anything went wrong. Use GetErrNo() to figure out if an empty string was an error or a real empty string!

SEE ALSO

A proper ARexx manual
ARexxResult(), ARexxSend(), ARexxSet(), FindPort()

1.7 ARexxResult()

NAME

ARexxResult - Send back result string to ARexx.

SYNOPSIS

```
ARexxResult ( Error, ResultString );
```

```
void ARexxResult ( int, STRING );
```

FUNCTION

This function is useful when the program using it is called from ARexx, otherwise it will not do anything.

It sets the return code and optionally the regular ARexx result string (only if the return code is zero (0)).

INPUTS

Error - True/false if error should be reported to ARexx.
Non-zero means error, zero means OK.

ResultString - The string which the result variable will hold after returning to ARexx.

RESULT

SEE ALSO

A proper ARexx manual
ARexxSend(), ARexxSet(), ARexxRead(), FindPort()

1.8 ARexxSend()

NAME

ARexxSend - Send a string to an ARexx port.

SYNOPSIS

```
Result = ARexxSend ( Port, String, Timeout );
```

```
string ARexxSend ( string, string, INT );
```

FUNCTION

Tries to send the 'String' to the specified ARexx 'Port'. If 'Timeout' is specified and non-zero, FrexxEd will wait the specified amount of seconds for ARexx to return a result string to us.

This function can be called with two or three parameters!

INPUTS

Port - Name of the ARexx port to send to.

String - Message to send to the port.

Timeout - Number of seconds to wait for an answer.

RESULT

If 'Timeout' is specified, the result string from the called program will be returned, or an empty string if anything went wrong.

If 'Timeout' isn't specified, FrexxEd skips the waiting and an empty string is always returned from this function.

SEE ALSO

A proper ARexx manual

ARexxResult(), ARexxSet(), ARexxRead(), FindPort()

1.9 ARexxSet()

NAME

ARexxSet - Set the contents of an ARexx variable.

SYNOPSIS

```
Progress = ARexxSet ( Variable, String );
```

```
int ARexxSet(string);
```

FUNCTION

When FrexxEd has received a message from an ARexx program, this function can return set the contents of a named ARexx variable.

INPUTS

Variable - Name of the variable.

String - New contents of the variable.

RESULT

Zero means progress, non-zero means that the port didn't exist or that any other error occurred.

SEE ALSO

A proper ARexx manual

ARexxResult(), ARexxSend(), ARexxRead(), FindPort()

1.10 AssignKey()

NAME

AssignKey - Assigns a function to a key sequence.

SYNOPSIS

```
ret = AssignKey ( FPLprogram, KeySequence, Condition );

int AssignKey ( STRING, STRING, STRING );
```

FUNCTION

This function makes FrexxEd recognize the given key sequence and when it is hit, the FPL program will be executed.

If this function is invoked with no parameter or with the first parameter specified as an empty string (""), a requester will appear asking for the right input.

If it is invoked with only one parameter or with 'KeySequence' set to an empty string (""), FrexxEd will wait for a key sequence to be pressed interactively. The key sequence is ended by pressing the 'escape' key (which can't be part of a key sequence).

The 'Condition' parameter controls whether the FPL program will get executed when the 'KeySequence' is pressed, or if it should continue searching for another action bound to that key sequence!

INPUTS

FPLprogram - This must be a complete program using FPL syntax.

KeySequence - The key sequence string. It uses straight left-to-right order plain key description text in the format:

```
[qualifier] <key>
```

AssignKey can recognize the following qualifier names:

"Amiga", "Shift", "Control", "Alt"

and these key names:

"F1" - "F20", "Del", "Delete", "Help", "Up", "Down",
 "Right", "Left", "Esc", "Escape", "Enter", "Return",
 "Tab", "Bspc", "Backspace", "Space", "Spc", "num0" -
 "num9", "num[", "num]", "num/", "num*", "num-",
 "num+", and "num.".

These latter strings should be written within single quotes (ie, 'delete'). (See example 3)

All of those is read case insensitive. Specifying multiple key presses is done by simply writing several key presses next to each other. (See example 4)

Characters are also recognized when written with a "\" prefix in regular C-style.
 (See example #5)

Rawkeys can be assigned by specifying the rawkey code (hexadecimal, octal, decimal or binary) within single quotes. (See example #6)

Spaces can only be written as a part of a key sequence like "\x20" or "\040". Normal spaces indicates the separation between words. (See example #7, #8)

Condition - The name of an info variable that has to hold a non-zero number or a non-empty string to make the FPL program get run when this key is pressed. If the name is preceded with a '!' character, the condition will be reversed (that is, the FPL program will run if the variable holds a zero or an empty string). Combine several dependencies with the logical expressions OR and AND by separating the variables with | resp. &. That is, to assign a key to execute if 'foo' or 'bar' is non-zero, use 'foo|bar' as 'Condition'. SEE the 'hook condition' chapter of the FrexxEd User's Guide.

RESULT

Standard FrexxEd error code. Returns zero if everything is OK, otherwise non-zero.

NOTE

The key string is matched case sensitive. That means that there is a difference the keysequences "amiga b" and "amiga B".

EXAMPLES

#1: To make 'amiga s' save the current buffer using the current name:

```
AssignKey("Save();", "Amiga s");
```

#2: To make 'control alt p' prompt the user for a text and then write it:

```
AssignKey("Output(PromptString());", "control ALT p");
```

#3: Make the cursor go up when pressing "alt delete" (!):

```
AssignKey("CursorUp();", "alt 'delete'");
```

#4: Assign the block sort to the sequence "Amiga B Control s"

```
AssignKey("BlockSort();", "Amiga B Control s");
```

#5: Assign the key producing the ASCII code 18 to Load():

```
AssignKey("Load();", "\x12");
```

#6: Assign rawkey code 61 (the "8" on the numerical keyboard) to move cursor up:

```
AssignKey("CursorUp();", "'61'");
```

or written hexadecimal:

```
AssignKey("CursorUp();", "'0x3e'");
```

or written binary:

```
AssignKey("CursorUp();", "'0b111101'");
```

or written octal:

```
AssignKey("CursorUp();", "'075'");
```

or use the existing easy-way:

```
AssignKey("CursorUp();", "num8");
```

#7: Assign the string "Foo Bar" to Save():

```
AssignKey("Save();", "Foo\x20Bar");
```

#8: Assign the string "WRONGSPACE();" to shift space:

```
AssignKey("WRONGSPACE();", "shift 'space'");
```

#9: To make 'amiga s' save the current buffer using the current name, but only if the 'changes' variable is non-zero:

```
AssignKey("Save();", "Amiga s", "changes");
```

#10: To make 'amiga s' save the current buffer using the current name, but only if the 'changes' variable is non-zero and the 'prevent_save' variable is zero.

```
AssignKey("Save();", "Amiga s", "changes&!prevent_save");
```

#11: To make the screen beep if any change is done when 'F4' is pressed:

```
AssignKey("DisplayBeep()", "'F4'", "changes");
```

NOTE: if 'changes' isn't non-zero, the string bound to the key 'F4' will get printed in the buffer (which is "3~" in my keymap)!

SEE ALSO

DeleteKey(), Change action on keys

1.11 Backspace()

NAME

Backspace - Kill characters to the left of the cursor.

SYNOPSIS

```
Actual = Backspace ( Number );
```

```
int Backspace ( INT );
```

FUNCTION

Performs a specified number of backspaces. If no parameter is specified, it defaults to 1.

INPUTS

Number - Amount of backspaces you want to perform.

RESULT

The number of backspaces that actually were done. If this function is invoked with a parameter making the backspacing hit the left edge of the first line, the backspacing is aborted even though the requested number of backspaces couldn't be done.

SEE ALSO

BackspaceWord(), Delete(), DeleteLine()

1.12 BackspaceWord()

NAME

BackspaceWord - Kill words to the left of the cursor.

SYNOPSIS

```
Actual = BackspaceWord ( Number );
```

```
int BackspaceWord ( INT );
```

FUNCTION

Performs a specified number of backspace words. For each number, one "word" is deleted and the cursor is moved, A "word" is a number of characters of the same character class written next to each other.

INPUTS

Number - Amount of words you want to backspace.

RESULT

The actual amount of deleted characters. If the function run out of words to backspace, it will be aborted before it could perform all the requested backspace words.

SEE ALSO

Backspace(), Delete(), DeleteLine(), DeleteEol(), DeleteWord()

1.13 BlockChange()

NAME

BlockChange - Change default block.

SYNOPSIS

```
Previous = BlockChange ( NewBlock );
```

```
int BlockChange ( int );
```

FUNCTION

This function changes the default block to the one specified. Block functions are always able to use the default block when performing operations on blocks, and this function changes the current default block to a new, already created, block.

NOTE

The startup internal default block is named "DefaultBlock".

INPUTS

NewBlock - BufferID of the new defaultblock. 0 or an illegal ID is equal to the internal default block ID.

RESULT

Previous block ID or zero if anything went wrong.

EXAMPLE

Change default block to the buffer named "Ninja.c":

```
BlockChange(GetBufferID("Ninja.c"));
```

SEE ALSO

All the other BlockXXXXXX() functions.
The 'Block concepts' chapter.

1.14 BlockCopy(), BlockCut(), BlockDelete()

NAME

BlockCopy - Copy a marked block.
BlockCopyAppend - Copy marked area to the end of a block.
BlockCut - Cut a marked block.
BlockCutAppend - Move marked area to the end of a block.
BlockDelete - Delete a marked area.

SYNOPSIS

```
ret = BlockXXXX ( BlockID, Column1, Line1, Column2, Line2 );  
int BlockXXXX ( INT, INT, INT, INT, INT );
```

(All these functions takes the same parameters of the same types, and return the same return codes.)

FUNCTION

This is a set of functions operating on a marked block area, with similar behaviour. They all use a default or specified destination block, and they all have the possibility to receive four coordinates to simulate a marked block area. These functions must be called with either none, one or five parameters.

BlockCopy() - Copies the contents of the currently marked area to the destination block.

BlockCopyAppend() - Appends the contents of the currently marked area to the end of the destination block.

BlockCut() - Cuts the contents of the marked area and copies it to the destination block.

BlockCutAppend() - Cuts the contents of the marked area and appends it to the end of the destination block.

BlockDelete() - Deletes the marked area.

NOTE

These functions can't work if the current buffer is the destination block!

INPUTS

BlockID - ID of the destination block. By specifying 0 (zero), the default block will be used.

Column1 - The column which the marked block is to start at.

Line1 - The line which the marked block is to start at.

Column2 - The column which the marked block is to end at.

Line2 - The line which the marked block is to end at.

RESULT

Standard FrexxEd error code. If zero is returned everything is OK, otherwise the function failed.

SEE ALSO

All the other BlockXXXXXX() functions.
The 'Block concepts' chapter.

1.15 BlockCreate()

NAME

BlockCreate - Create a new block. [*]

SYNOPSIS

```
ret = BlockCreate ( BlockName );
```

```
int BlockCreate ( string );
```

FUNCTION

This function creates/initializes a new block. It is pretty much the same as New(), but this sets the "block bits".

INPUTS

BlockName - The name of the new block/buffer.

RESULT

The ID of the newly created buffer/block.

SEE ALSO

BlockChange(), Kill()
The 'Block concepts' chapter.

1.16 BlockMark()

NAME

BlockMark - Mark an area in a view.
BlockMarkLine - Mark full width of an area in a view.
BlockMarkRect - Mark a rectangular area in a view.

SYNOPSIS

```
BlockMarkXXX ( Status, Column1, Line1, Column2, Line2 );
```

```
void BlockMarkXXX ( INT, INT, INT, INT, INT );
```

(All functions accept the same arguments of the same types,
and they return the same return codes.)

FUNCTION

The functions work very similar. They set the block mark
status and possibly the coordinates of the marked area.

BlockMark() - Performs "column style" block marking at specified
(or current) coordinates.

BlockMarkLine() - Performs "line-mode" block marking at specified
(or current) coordinates.

BlockMarkRect() - Performs rectangular block marking at
specified (or current) coordinates.

"column style" blocks are the "usual" kind of blocks. They
follow the orientation of the text lines, which means that
when the cursor is moved one line down, all columns to the
right on the previous line and all columns to the left on
the current line will automatically be included in the block.

"line-mode" block marking makes the block always mark the whole
current line, from the leftmost to the rightmost column.

Rectangular blocks are always square-shaped. They exist to
make it possible to cut out non line-oriented parts of text
documents, such as parts of figures or tables.

These functions should be called with none, one, three or five
parameters.

The main usage for BlockMark() is of course the interactive toggle
of the block existence. The toggling has three different 'positions'.
The marked block that is attached to the cursor (moving the cursor
makes the block-edge follow it), "released" block (a block is marked
and remain at the same place even though you run around with the

cursor and finally: "off" - no block is marked. On the status line, there will be a 'b' visible when there is an attached block mark and 'B' when it is released.

The 'Status' parameter can set the toggle state of the block. Set status 0 to unmark the block, set 1 to make it mark in attached-style and use 2 for the released block mark style. -1 means "ignore", it is used when all you want to do is specify new coordinates without affecting the "current" block mark status.

INPUTS

Status - Set state of the block:
 -1 = Ignore. Just set the coordinates.
 0 = Off. Quit block marking.
 1 = On. Start block marking.
 2 = Release. Let the block go.
 Column1 - Start column of the marked block area.
 Line1 - Start line of the marked block area.
 Column2 - End column of the marked block area.
 Line2 - End line of the marked block area.

RESULT

EXAMPLE

Toggle column style block marking:

```
BlockMark();
```

Activate block marking:

```
BlockMark(1);
```

Mark a rectangular block, released from the cursor, from line 5 column 7 to line 10 column 12:

```
BlockMarkRect(2, 7, 5, 12, 10);
```

Make a block mark from the start of the current line to the cursor. This also starts "block marking mode":

```
BlockMark(1, 1, ReadInfo("line") );
```

Mark/highlight from 1,10 to 5,20. How does the BlockMark() call look like?

```
BlockMark(2, /* released block style */  
          10, 1, /* from column 10 line 1 */  
          20, 5); /* to column 20 line 5 */
```

The user have already started a block marking (like with amiga b) and your program should change the start column of that marked block:


```
BlockMark(1, /* end-of-block is attached to cursor */  
          1,1); /* marks from the start of buffer */
```

SEE ALSO

The other BlockXXXX() functions.
The 'Block concepts' chapter.

1.17 BlockMove()

NAME

BlockMove - Moves a marked block area horizontally.

SYNOPSIS

```
ret = BlockMove ( Steps );  
  
int BlockMove ( INT );
```

FUNCTION

This function moves the contents of the marked block area a specified number of steps horizontally. If this function is invoked without parameter, the block can be moved interactively with the cursor keys and tab/shift tab to move the block. Cancel the movings and get back to previous state by pressing q as in quit, place the block at the current position by pressing return or y as in yes. The maximum number of steps possible to move, is the same as the width of the screen.

INPUTS

Steps - Number of steps to move the block. Positive numbers to right, and negative to left.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

The other BlockXXXX() functions.
The 'Block concepts' chapter.

1.18 BlockPaste()

NAME

BlockPaste - Insert a block in the buffer.
BlockPasteRect - Insert block as rectangular.

SYNOPSIS

```
ret = BlockPaste ( BlockID );  
ret = BlockPasteRect ( BlockID );  
  
int BlockPaste ( INT );  
int BlockPasteRect ( INT );
```

FUNCTION

BlockPaste() and BlockPasteRect() insert the marked, current or specified block at the current position.
The first one inserts the block as a regular line oriented block, and the second one as a rectangular block.

They can also be forced to use the default block prior to the currently marked!

These functions must be invoked with either none or one parameter.

NOTE

The block can't be pasted into it's own buffer.

INPUTS

BlockID - ID of a valid buffer/block. Specify 0 (zero) to use the default block as source. -1 makes the default block to get pasted even if there is a currently marked one.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

The other BlockXXXX() functions.
The 'Block concepts' chapter.

1.19 BlockSort()

NAME

BlockSort - Sort a block.

SYNOPSIS

```
ret = BlockSort ( BlockID, Field, Flags );
```

```
int BlockSort ( INT, INT, INT );
```

FUNCTION

This function lexicographically sorts the lines of a marked, default or specified block according to a field. Fields are separated with whitespace. If no Field is specified, it defaults to 1.

Invoked with none or with one parameter, this will pop up a requester.

This function can be invoked with none to three parameters.

INPUTS

BlockID - ID of a valid buffer/block. Specify 0 (zero) to sort the default block.

Field - Field number. 0 means from the absolute left (column 1), all other numbers means from the beginning of that field. Fields are separated by characters marked with

the space class in the FACT.

Flags - bit 0 - case sensitive if unset (insensitive if set)
bit 1 - forward if unset (backwards if set)

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

EXAMPLE

Sort the block backwards, case sensitive according to field 2:

```
BlockSort(0, 2, 2);
```

Sort the block forwards, case sensitive according to the leftmost position of the lines:

```
BBlockSort(0, 0, 0);
```

SEE ALSO

The other BlockXXXX() functions.
The 'Block concepts' chapter.

1.20 BSearch()

NAME

BSearch - Binary search in a string array.

SYNOPSIS

```
Found = BSearch ( Array, Search, Items, Method );
```

```
int BSearch ( &string[], string, INT, INT );
```

FUNCTION

Scans a sorted string array pointed to by the 'Array' parameter for a string that equals the 'Search' parameter. The string array must be sorted forwards and case sensitive if 'Method' isn't specified (with the same bit flags as Sort() accepts).

INPUTS

Search - String to search for.

Array - Reference to a string array.

Items - If not the entire array should be scanned, this will tell the function how many items it should scan for the match.

Method - Defines sorting details. Set the following bits to specify how the array is sorted:
0 - case insensitive
1 - backwards sort

RESULT

Array position, or -1 if not found.

NOTE

If the array isn't sorted the right way, or if 'Items' isn't set properly, this function may very well not find the string.

SEE ALSO

Sort()

1.21 CConvertString()

NAME

CConvertString - Convert a raw string to C style. [*]

SYNOPSIS

```
Cstring = CConvertString ( RawString );  
  
string = CConvertString ( string );
```

FUNCTION

This function makes a C style string out of a raw binary string. E.g, ASCII code 9 will be converted to \t, 10 will become a \n and so on. Very practical if you would like to visualize a string that might include non printable characters.

INPUTS

RawString - The string to convert from.

RESULT

The converted string. If anything went wrong the resulting string will be empty ("").

SEE ALSO

FACTConvertString()

1.22 CenterView()

NAME

CenterView - Center the view vertically around the cursor.

SYNOPSIS

```
ret = CenterView ( ViewID );  
  
int = CConvertString ( INT );
```

FUNCTION

Makes the current line, of the current or the specified view, the one in the middle of the view - if possible. This function can be called with or without parameter.

INPUTS

ViewID - The view to center.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

ScrollDown(), ScrollUp()

1.23 Check()

NAME

Check - Returns file information.

SYNOPSIS

```
ret = Check ( Filename, Options );

int Check ( string, STRING );
```

FUNCTION

Returns information regarding the specified file, according to the 'Options'. Without any options, Check() returns non-zero if the specified file exists.

No options has been implemented yet!

INPUTS

Filename - Full path name to the file.

Options - Not currently in use!

RESULT

If no option is specified:
* if zero is returned, the file doesn't exist, otherwise the file is there!

SEE ALSO

System()

1.24 Clean()

NAME

Clean - Execution without hooks. [*]

SYNOPSIS

```
Clean ( Program );

void Clean ( string );
```

FUNCTION

This function executes the specified string as an FPL program. When that program runs (including all called functions), no hooks will be called.

INPUTS

Program - A complete FPL program.

RESULT

FPL error code. Zero if everything went OK, otherwise non-zero.

SEE ALSO

ExecuteString(), ExecuteBuffer(), Hook(), HookClear()
Error code defines in <libraries/FPL.h>

1.25 Clear()

NAME

Clear - Clear a buffer.

SYNOPSIS

```
ret = Clear ( BufferID );
```

```
int Clear ( INT );
```

FUNCTION

Clear removes all contents of the current or specified buffer.
This function can be called with or without argument.

INPUTS

BufferID - A valid buffer ID.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

New(), BlockCreate(), Kill()

1.26 ClipToString()

NAME

ClipToString - Return clipboard.device data as a string.

SYNOPSIS

```
String = ClipToString ( Unit );
```

```
string = ClipToString ( INT );
```

FUNCTION

This function returns the contents of the specified or default (0) unit from clipboard.device. Using this function you can interface virutally any other software running on your Amiga using the clipboard. This function can be called with or without parameter.

INPUTS

Unit - Which clipboard unit you want to read data from.

RESULT

The data read is returned as a string.

EXAMPLE

Output the contents of clipboard unit 0 into the current buffer:

```
Output(ClipToString(0));
```

SEE ALSO

StringToClip()

1.27 CloneWB()

NAME

CloneWB - Clone the current public screen.

SYNOPSIS

```
ret = CloneWB ( WindowID );
```

```
int = CloneWB ( INT );
```

FUNCTION

This function copies as much as possible from the current public screen to FrexxEd's setup settings for the specified or current window. Set WindowID to -1 to change the default settings. Can be invoked with or without parameter.

INPUTS

WindowID - A valid Window ID.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

1.28 ColorAdjust()

NAME

ColorAdjust - Adjust colors.

SYNOPSIS

```
ColorAdjust ( ColorNo, Red, Green, Blue );
```

```
void ColorAdjust ( INT, INT, INT, INT );
```

FUNCTION

This function enables you to change the RGB-settings of a specified color. Without any parameter a palette requester will appear to enable interactive changing of colors.

INPUTS

ColorNo - Color number. Which color do you want to change.
Ranging from zero to maximum colors - 1.

Red - The amount of red. 0 - 15

Green - The amount of green. 0 - 15

Blue - The amount of blue. 0 - 15

RESULT

SEE ALSO

ColorReset(), graphics.library/SetRGB4()

1.29 ConstructInfo()

NAME

ConstructInfo - Create a new info variable.

SYNOPSIS

```
ConstructInfo( Name, FPLmodify, FPLreq, Type, CycleString, Min, Max, Default );
```

```
int ConstructInfo( string, string, string, string, string, int, int, STRING/INT ↵  
);
```

FUNCTION

This function creates a new info variable which is added to the internal list of info variables. The new info variable will be accessible/readable/modifyable just like other info variables through ReadInfo() and SetInfo().

This function can be invoked with 7 or 8 parameters.

INPUTS

Name - The name of the new info variable. (If the name is already used, everything is ignored)

FPLmodify - FPL string to run when this variable gets modified.
Full FPL syntax required.

FPLreq - FPL program to run when its button in the settings window is pressed. Full FPL syntax required.
The return value of the program will become the new value of the setting. If the program doesn't return anything, nothing is changed.

Type - A string holding any number of the following characters in any combination:

(existence information, specify one of these:)

G - Global variable

L - Local variable

(variable type, specify one of these:)
 B - A boolean variable (just TRUE or FALSE)
 I - An integer variable
 C - A cycle variable, able to alter between a specified number of values.
 S - A string variable

(extras:)
 W - Write this variable in the default file
 R - Read only variable
 H - Hidden. Does not appear in a settings window.

(group)
 Specify which groups the variable is to be sorted in.
 Write the names of the groups within parentheses next to each other. Available groups are:

Display - Display matters (eg. 'cursor_x')
 IO - OS interactions (eg. 'expand_path')
 Screen - Screen oriented variable (eg. 'window')
 System - Misc (eg. 'autosave', 'language')

CycleString - If the variable is a cycle type, this string specifies the names associated with each cycle step. It should be specified as "string0|string1|string2|...".
 The variable will contain the number of the currently selected cycle (0 for the first, 1 for the second and so on).

Min - If this is an integer variable, this is its minimum value.

Max - If this is an integer variable, this is its maximum value. To make an integer variable without any limits, neither min nor max, just make sure both equal zero.

Default - The default value/string of this setting. If this isn't specified, it defaults to the minimum value or an empty string ("").

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

EXAMPLE

Create a local boolean variable with the name 'tab_space' that should be saved and is unset by default:

```
ConstructInfo("tab_space", "", "", "LBW(display)", "", 0, 0);
```

Create a global cycle variable named 'weather' that can alter between "rainy", "sunny" and "cloudy". It is sorted under the 'system' group and is by default set to "sunny":

```
ConstructInfo("weather", "", "", "GC(system)",
```

```
"rainy|sunny|cloudy", 0, 0, 1);
```

SEE ALSO

```
ReadInfo(), SetInfo(), PromptInfo(), DeleteInfo()
```

1.30 CopyInfo()

NAME

CopyInfo - Copy info variables.

SYNOPSIS

```
CopyInfo ( SourceID, DestID, IncludeMask, ExcludeMask, Variable, ... );
```

```
void CopyInfo( int, int, INT/STRING, INT/STRING, STRING, ... );
```

FUNCTION

Copies the specified set of info variables to another buffer (or default).

By altering the 'IncludeMask', you set the bitmask of which setting types you want copied. If this is set to 0, no masks are functional (but the default which depends on the 'SourceID' parameter), -1 selects all.

By altering the 'ExcludeMask' you can exclude setting types from the selected ones. Use 0 to not exclude anything.

Both masks can negate the masks. That will give you all types but the negated one. You do that by preceeding the mask name with a '!' character within the parentheses.

'DestID' is the destination bufferID, entryID or windowID.

If 'Variable' is specified with a list of info variables that should be visible in the window. They are added to the one already matched by the masks.

Specify settings to include by simply writing their name (as in "changes"), and exclude settings by preceding the name with a '!' character (as in "!changes"). When excluding, you can specify a '*' as a kind-of wildcard. All settings that match the string before that '*' will be excluded (as in "!colour*").

This function can be called with two or more parameters.

This is the same system as PromptInfo is using.

INPUTS

SourceID - Source ID of settings to read.

```
-1 current locals (default IncludeMask "(local)" and default
ExcludeMask "(global) (read) (hidden)")
```

DestID - Destination ID to copy variables to.

```
0 Means the default buffer.
```

IncludeMask - Which setting types that should be included. Available types are:

- Display - Display matters (eg. 'cursor_x')
- IO - OS interactions (eg. 'expand_path')
- Screen - Screen oriented variable (eg. 'window')
- System - Misc (eg. 'autosave', 'language')
- User - User created info variable
- Hidden - Hidden variable. By default not visualized.
- Global - Global variable.
- Local - Local variable.
- Read - Read-only variable. Cannot get modified.

They should be written within parentheses right next to each other.

ExcludeMask - Which setting types that should be excluded from the ones that matched the include mask. See 'IncludeMask' for available masks.

Variable - A list of settings to include in the copy.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

PromptInfo

1.31 ColorReset()

NAME

ColorReset - Reset color(s) to workbench default.

SYNOPSIS

```
ColorReset ( Color );
```

```
void ColorReset ( INT );
```

FUNCTION

Resets all colors or just the specified one (of the current window) to the workbench/ preferences default. If invoked without argument, all colors is affected.

INPUTS

Color - Color bits. Bit 0 is color 0, bit 1 color 1 and so on. -1 resets all colors.

RESULT

SEE ALSO

ColorAdjust(), graphics.library/SetRGB4()

1.32 CurrentBuffer()

NAME

CurrentBuffer - Change current buffer.

SYNOPSIS

```
Previous = CurrentBuffer ( BufferID );  
  
int Currentbuffer ( int );
```

FUNCTION

This function changes the current buffer to the specified one. If the current buffer isn't visible when the FPL program ends, it will pop up visible.

INPUTS

BufferID - A regular, valid buffer ID.

RESULT

Previous buffer ID or zero (0) if anything went wrong.

SEE ALSO

GetBufferID(), Activate()

1.33 CursorActive()

NAME

CursorActive - Alter the cursor state.

SYNOPSIS

```
Previous = CursorActive ( Switch );  
  
int CursorActive ( INT );
```

FUNCTION

Turns the cursor on or off. If called without parameter, simply returns the current state.

INPUTS

Switch - New state: 1 = on, 0 = off

RESULT

Previous cursor state.

SEE ALSO

Visible()

1.34 Cursor movements

NAME

CursorDown - Moves cursor down.
CursorLeft - Moves cursor left.

CursorLeftWord - Moves cursor entire words left.
CursorRight - Moves cursor right.
CursorRightWord - Moves cursor entire words right.
CursorUp - Moves cursor up.

SYNOPSIS

```
Actual = CursorXXXXX ( Steps );
```

```
int CursorXXXXX ( INT );
```

(All the six functions accept the same amount of parameters of the same types, and they use the same return codes.)

FUNCTION

CursorUp() and CursorDown() move the cursor one or a specified amount of lines up or down.

CursorRight() and CursorLeft() move the cursor one or a specified amount of columns right or left.

CursorRightWord() and CursorLeftWord() move the cursor one or a specified amount of words right or left.

INPUTS

Steps - The amount of steps to take. Negative numbers mean opposite direction.

RESULT

The actual number of steps taken. FrexxEd may abort the function before being able to perform all requested steps due to the end or beginning of file, and the return value will tell how many steps that were performed.

SEE ALSO

ScrollDown(), PageDown()

1.35 CursorStack

NAME

CursorStack - Stores/gets cursor position.

SYNOPSIS

```
CursorStack ( Flag );
```

```
void CursorStack ( INT );
```

FUNCTION

This function stores the current cursor position on a stack, or gets a cursor position from it. If called without parameter or -1, it stores, and if called with 1 as parameter it sets the cursor on the position that is on the top of the cursor stack.

NOTE

The stack is simply an 8 level circular stack. You can't push more than eight positions to it and expect to get them back. There's no

harm done if you don't pull the same amount you push.

INPUTS

Flag - (-1) for storing (push), 1 for getting (pull)

RESULT

none

SEE ALSO

GotoLine()

1.36 Delay()

NAME

Delay - Sleep for a while.

SYNOPSIS

```
Delay ( Pause );
```

```
void Delay ( int );
```

FUNCTION

The parameter 'Pause' specifies how many ticks (50 per second) to wait before returning control.

QUOTE

"Stay a while... stay forever!!!" - Impossible Mission

INPUTS

Pause - Sleeping time specified in 50th of seconds.

RESULT

SEE ALSO

1.37 Deiconify()

NAME

Deiconify - Pops up FrexxEd from the icon state.

SYNOPSIS

```
ret = Deiconify ( WindowID );
```

```
int Deiconify ( INT );
```

FUNCTION

The reversed action of an Iconify() call. The FrexxEd screen/window pops up again.

INPUTS

WindowID - Specifies which window that is to be deiconified. No ID

specified, or 0, makes all windows get deiconified. -1 pops up the current window.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

1.38 Delete()

NAME

Delete - Delete char(s) at cursor.
DeleteWord - Delete word(s) at cursor.

SYNOPSIS

```
Actual = DeleteXXXX ( Steps );  
  
int DeleteXXXX ( INT );
```

FUNCTION

Delete() - Removes one or a specified number of characters at the cursor.

DeleteWord() - Removes one or a specified number of words at the cursor.

INPUTS

Steps - The amount of deletions to perform.

RESULT

The actual number of deletions that were performed before end of file (EOF) was reached.

SEE ALSO

Backspace(), BackspaceWord(), DeleteLine(), DeleteEol()

1.39 DeleteEol()

NAME

DeleteEol - Delete the rest of the line.

SYNOPSIS

```
Chars = DeleteEol();  
  
int = DeleteEol ();
```

FUNCTION

This function deletes the line from the current position to the rightmost column.

INPUTS

RESULT

The number of characters that were deleted.

SEE ALSO

DeleteLine(), Delete()

1.40 DeleteInfo()

NAME

DeleteInfo - Delete an info variable.

SYNOPSIS

```
ret = DeleteInfo ( Variable );
```

```
int = DeleteInfo ( string );
```

FUNCTION

This function deletes the specified info variable.

INPUTS

Variable - The name of the variable to delete.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

ConstructInfo(), ReadInfo(), SetInfo()

1.41 DeleteKey()

NAME

DeleteKey - Delete a key assigning.

SYNOPSIS

```
DeleteKey ( KeySequence );
```

```
void DeleteKey ( STRING );
```

FUNCTION

When you want to remove a previous AssignKey() to a certain key sequence, this is the function to call. When deleting a key sequence, the function that was earlier bound to that will yet again become the present action for that key sequence.

If invoked without parameter, interactive mode is entered which enables the user to type the sequence to be removed, ended by hitting escape. Interactive key sequences entered, will always strip the caps lock and shift qualifiers.

INPUTS

KeySequence - A valid key sequence string. See AssignKey()
for how to write such a string.

RESULT

SEE ALSO

AssignKey()

1.42 DeleteLine()

NAME

DeleteLine - Delete a line.

SYNOPSIS

```
Actual = DeleteLine ( Lines );
```

```
int DeleteLine ( INT );
```

FUNCTION

This function removes one or a specified number of lines.

INPUTS

Lines - The number of lines to delete.

RESULT

The actual amount of lines that were deleted. When end of file (EOF) is reached before all requested lines have been deleted, this function is aborted.

SEE ALSO

DeleteEol(), Delete()

1.43 DisplayBeep()

NAME

DisplayBeep - Flash the screen.

SYNOPSIS

```
DisplayBeep ();
```

```
void DisplayBeep ();
```

FUNCTION

This function makes the screen flash. It calls the intuition.library function with the same name.

INPUTS

RESULT

SEE ALSO

`Intuition.library/DisplayBeep()`

1.44 DuplicateEntry()

NAME

`DuplicateEntry` - Make two entries from one.

SYNOPSIS

```
NewID = DuplicateEntry ( EntryID );
```

```
int = DuplicateEntry ( INT );
```

FUNCTION

This function makes a copy of the current or specified entry and adds it to the list of entries.

INPUTS

`EntryID` - A valid entry ID.

RESULT

The ID of the newly created entry.

SEE ALSO

`GetEntryID()`, `Kill()`

1.45 Change case

NAME

`DownCase` - Downcase a block.

`SwapCase` - Swap case of a block.

`UpCase` - Upcase a block.

SYNOPSIS

```
ret = XXXCase ( BlockID );
```

```
int XXXCase ( INT );
```

FUNCTION

These functions work on the marked, default or the specified block.

`DownCase()` - Change all letters to downcase.

`SwapCase()` - Change all letter to the opposite case.

`UpCase()` - Change all letters to upcase.

INPUTS

`BlockID` - ID of a valid buffer/block. Specify 0 (zero) to change

the default block.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

BlockMark()

1.46 Execution functions

NAME

ExecuteBuffer - Execute a buffer as an FPL program.
ExecuteFile - Execute a file as an FPL program.
ExecuteLater - Execute a string as an FPL program later.
ExecuteString - Execute a string as an FPL program.

SYNOPSIS

```
ret = ExecuteBuffer ( BufferID );  
ret = ExecuteFile ( FileName, Program );  
ret = ExecuteLater ( String );  
ret = ExecuteString ( String, BufferID );
```

```
int ExecuteBuffer ( INT );  
int ExecuteFile ( string, STRING );  
int ExecuteLater ( string );  
int ExecuteString ( string, INT );
```

FUNCTION

These are functions for invoking FPL programs and FPL program executions.

ExecuteBuffer executes the current or specified buffer.

ExecuteFile executes the specified file or the files that match the pattern. If the optional 'Program' parameter is used, `_that_` string is executed instead of the files actual "main" program (the program outside all functions).

ExecuteLater executes the specified string later on.

ExecuteString executes the specified string. It handles an optional BufferID which specifies in which buffer the execution is to take effect in.

To read more about what FPL programs are, how to code them and how to control them, read the "Programming FrexxEd" chapter of this manual.

INPUTS

BufferID - A valid buffer ID.

FileName - Full path name/pattern to the file(s). Files will be search for through the path specified in 'fpl_path'.

String - The FPL program string.

RESULT

FPL error code. Zero if everything went OK, otherwise non-zero.

SEE ALSO

Introduktion to FPL

Clean()

Error code defines in <libraries/FPL.h>

1.47 FaceAdd()

NAME

FaceAdd - Add words to a face style.

SYNOPSIS

```
ret = FaceAdd ( FaceHandle, StyleHandle, Strings, Flags );
```

```
int FaceAdd ( int, int, string, string );
```

FUNCTION

Adds strings to a style that should be coloured/styled accordingly.

'FaceHandle' and 'StyleHandle' must be picked up with FaceGet() and FaceStyle() prior the use of this function.

'Strings' is all strings that should be added, separated with vertical bars (|). If a vertical bar is meant as part of a string, it must be added separately and the "usepipe" flag should be set in 'Flags'.

'Flags' describes how the string(s) is or are supposed to be parsed.

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. See below.
weak	- a weak string. Will be conquered by strong ones.
lnonspace	- 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)

Weak/Strong styles are introduced to enable some styles to appear more important than others. STRONG types interfere with WEAK ones so that if a STRONG style-string appears within the limits of a WEAK one, the STRONG one's style will be used. STRONG _only_ interfere on WEAK styles, and only WEAK styles can be interfered. Styles marked as neither STRONG nor WEAK will not be dominant, nor get run over.

INPUTS

FaceHandle - Which face to add these strings to. Return code from FaceGet()

StyleHandle - The style to use. Return code from FaceStyle()

Strings - The string(s) to add to this style.

Flags - Options to control the parsing/colouring/styling.

EXAMPLE

```
/* Create a C programming face mode */
{
    int face = FaceGet("C", 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|lnonospace|weak", "\n");

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

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

Faces and how to use them
 FaceGet
 FaceStyle
 FaceRead

1.48 FaceGet()

NAME

FaceGet - Get a named face.

SYNOPSIS

```
ret = FaceGet ( Name, Flags );

int FaceGet ( string, INT );
```

FUNCTION

Get and optionally creates a new Face.

INPUTS

Name - Name of the Face to check or create.

Flags - 0 just check whether the Face exists.

1 create the Face if it doesn't exist.

RESULT

FaceHandle or 0 if non-existent/error. The FaceHandle should be used in the FaceAdd() call.

SEE ALSO

Faces and how to use them

FaceAdd

FaceRead

FaceStyle

1.49 FaceRead()

NAME

FaceRead - Read face style characteristics.

SYNOPSIS

```
ret = FaceRead ( Name, &Style, &fgpen, &bgpen );

int FaceRead ( string, &string, &int, &int );
```

FUNCTION

Returns the characteristics of the named style in the three variables to which references are sent to.

INPUTS

Name - Name of the style to read.

Style - A string with all the different styling written in. If all kinds of styling was used, it would contain:

'reverse|bold|italic|underline'. No style at all would make the string contain just 'normal'.

fgpen - The number of the pen used as foreground.

bgpen - The number of the pen used as background.

EXAMPLE

```
string array[1];
string name;
int number;
```

```
number=GetList("facestyle", &array);
if(number) {
    string style;
    int fg, bg;
    Request(sprintf("%d faces", number));
    RequestWindow("Defined face styles:",
                  "Style:", "a", &array, &name, number);
    FaceRead(name, &style, &fg, &bg);
    Request(sprintf("%s = '%s' fg: %d bg: %d", name, style, fg, bg));
}
else
    Request("none");
```

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

Faces and how to use them
FaceAdd
FaceGet
FaceStyle

1.50 FaceStyle()

NAME

FaceStyle - Get a named face style.

SYNOPSIS

```
ret = FaceStyle ( Name, Style, Fgpen, Bgpen, Flags );

int FaceStyle ( string, string, int, int, INT );
```

FUNCTION

Set a new named style or reset an already existing.

INPUTS

Name - Face style name

Style - Style string. Enter which kind of style you want this to use. 'reverse|bold|italic|underline' sets all available styles.

Fgpen - Sets the foreground pen to use.

Bgpen - Sets the background pen to use.

Flags - Set to 1 to FORCE writing a new style. Usually the one that writes the style first decides the look of it to enable startup programs to defined the looks of faces added later on by custom programs. This FORCE flags let's you change the first stored definition.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned,

everything is OK, otherwise something went wrong.

SEE ALSO

1.51 FACT()

NAME

FACT - Change the FrexxEd ASCII Convert Table

SYNOPSIS

```
ret = FACT ( [Name,] ASCIIcode, Tag, [Value, [Tag]], ...);
```

```
int FACT ( [STRING,] int, INT or STRING, ... );
```

FUNCTION

The FACT customizes the screen output of a certain ASCII code, among other things. For closer documentation on why and other stuff, refer to the "Character set" chapter.

FrexxEd supports local FACTs, and to specify another one than the current one, the first parameter of the FACT() call should be the name of that special FACT.

The 'fact' info variable holds the name of the current FACT.

TAGS

The tags are all integers, some of them requires nothing else but the tags alone, but some requires a following argument:

Tag Arg Description

--- ---

'E' flag Clear a flag. The argument can be any one of 'W', ' ', '!', 'N', 'T' or '-' for all flags.

'W' Word class attribute. When set, that means that this character can be a part of a regular word.

' ' White space attribute. This character is a blank.

'!' Symbol attribute. Such characters are e.g: !#\$%&.

'N' Newline attribute. This character causes a new line to start! This attribute can only be set when no buffers, blocks or undo buffers exist in FrexxEd. If you think of modifying this, do it at the absolute startup.

'T' Tab attribute. This character causes a jump to the next tab position stop.

(' char This character is an open delimiter. The specified 'char' argument is the ASCII code of the matching opposite.

'**)**' char This character is a close delimiter. The specified 'char' argument is the ASCII code of the matching opposite.

'**U**' char This character is an upper case character. The specified 'char' argument is the ASCII code of the lower case version of the same character.

'**L**' char This character is a lower case character. The specified 'char' argument is the ASCII code of the upper case version of the same character.

'**S**' string This is the string which will be visualized when this ASCII code appears in the buffer.
Read that character set chapter for details of how to customize the strings.

INPUTS

Name - If any other FACT than the current is to be changed.

ASCIIcode - The ASCII code which to modify the FACT entry for, or

- 1 = end of file character
- 2 = 'ContinuedLine' character.
- 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)
- 6 = The character used to the side of all text in a view. Where the contents of the file doesn't fill up.

Tag - See TAGS description

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

Character set
FACT reading functions

1.52 FACTClear()

NAME

FACTClear - Reset the FACT to defaults.

SYNOPSIS

```
ret = FACTClear ( ASCII, Name );
```

```
int FACTClear ( INT, STRING );
```

FUNCTION

Resets the specified ASCII code FACT entry to default. If no ASCII

code is specified (or 256), all FACT entries will be reset. Use this to remove not so very good FACT experiments.

Can be called with none, one or two parameters.

INPUTS

ASCII - The character code to reset. Valid numbers are 0-255 for all those ASCII codes, or
-1 = end of file character
-2 = 'ContinuedLine' character.
-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)

Name - Name of the FACT to clear.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

FACT(), FACTConvertString(), FACTString()
FACT reading functions
Character set

1.53 FACTCreate()

NAME

FACTCreate - Create a new FACT.

SYNOPSIS

```
ret = FACTCreate ( Name );  
  
int FACTClear ( string );
```

FUNCTION

Creates a new FACT. After it has been created, it has the built-in look, and it can be altered and used.

INPUTS

Name - Name of the new FACT.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

FACT(), FACTConvertString(), FACTString()
FACT reading functions
Character set

1.54 FACTDelete()

NAME

FACTDelete - Delete a FACT.

SYNOPSIS

```
ret = FACTDelete ( Name );

int FACTDelete ( string );
```

FUNCTION

Deletes a named FACT.

INPUTS

Name - Name of the FACT to delete.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

FACT(), FACTConvertString(), FACTString()
FACT reading functions
Character set

1.55 FACTConvertString()

NAME

FACTConvertString - Convert a string with the FACT.

SYNOPSIS

```
Converted = FACTConvertString ( Text );

string FACTConvertString ( string );
```

FUNCTION

This function uses the current FACT to convert the input string, and returns the conversion. The resulting string will look just as the specified string would look like on screen (font styles and colours are of course stripped).

INPUTS

Text - The string to convert.

RESULT

The converted text.

EXAMPLE

To obtain the string that is output when a tab character appears in a file:

```
string output = FACTConvertString("\t");
```

SEE ALSO

FACT(), FACTString(), CConvertString()
FACT reading functions
Character set

1.56 FACTString()

NAME

FACTString - Returns the FACT string of a character.

SYNOPSIS

```
Text = FACTString ( ASCII );  
  
string FACTString ( int );
```

FUNCTION

Returns the string that is displayed when the specified ASCII code is displayed on screen.

EXAMPLE

To obtain the string that is output when a tab character appears in a file:

```
string output = FACTString('\t');
```

INPUTS

ASCII - The character code to view. Valid numbers are 0-255 for all those ASCII codes, -1 for the "end of file" character and -2 for the "continued line" character. See the FACT chapter for a complete list of IDs.

RESULT

The string that represent the specified ASCII code.

SEE ALSO

FACT(), FACTClear(), FACTConvertString()
FACT reading functions
Character set

1.57 FindPort()

NAME

FindPort - Find a given system message port. [*]

SYNOPSIS

```
Exist = FindPort ( Portname, Timeout );  
  
int FindPort ( string, INT );
```

FUNCTION

This function returns whether the specified 'Portname' exist. If the option 'Timeout' field is specified with a non-zero number, FrexxEd will wait at the most that number of seconds for the port to appear.

INPUTS

Portname - The name of the port.

Timeout - Number of seconds to wait for it.

RESULT

Non-zero if the port exist, otherwise zero.

SEE ALSO

exec.library/FindPort()
A proper ARexx manual
ARexxRead(), ARexxResult(), ARexxSend(), ARexxSet()

1.58 Fold()

NAME

Fold - Fold an area of a buffer.

SYNOPSIS

```
Ret = Fold ( From, To, BufferID );
```

```
int Fold ( INT, INT, INT );
```

FUNCTION

Makes a fold of the lines between 'From' and 'To'. 'BufferID' may specify which buffer to affect.

If this function is called without parameters, the currently marked block's first and last line will be used.

INPUTS

From - First line of the new fold.

To - Last line of the new fold.

BufferID - A valid buffer ID.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

The Using folds chapter.
FoldDelete(), FoldHide(), FoldShow()

1.59 FoldDelete()

NAME

FoldDelete - Delete a fold.

SYNOPSIS

```
Ret = FoldDelete ( Line, BufferID );
```

```
int FoldDelete ( INT, INT );
```

FUNCTION

Removes the fold (not the text) on the specified or current line.

Can get called with none, one or two parameters. Without parameters, the current line will be used.

INPUTS

Line - Line on which the fold is to be removed from. -1 will make all folds in the buffer to get deleted!

BufferID - A valid buffer ID.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

The Using folds chapter.

Fold(), FoldHide(), FoldShow()

1.60 FoldHide()

NAME

FoldHide - Hide a fold.

SYNOPSIS

```
Ret = FoldHide ( Line, BufferID );
```

```
int FoldDelete ( INT, INT );
```

FUNCTION

Hides the fold on the specified or current line.

Can get called with none, one or two parameters. Without parameters, the current line will be used.

INPUTS

Line - Line on which the fold is. 0 means hide all folds

BufferID - A valid buffer ID.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

The Using folds chapter.

Fold(), FoldHide(), FoldShow()

1.61 FoldShow()

NAME

FoldShow - Show a fold.

SYNOPSIS

```
Ret = FoldShow ( Line, BufferID );
```

```
int FoldShow ( INT, INT );
```

FUNCTION

Shows the fold on the specified or current line.

Can get called with none, one or two parameters. Without parameters, the current line will be used.

INPUTS

Line - Line on which the fold is.
0 means show this fold *only*
-1 means show all folds

BufferID - A valid buffer ID.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

The Using folds chapter.
Fold(), FoldHide(), FoldShow()

1.62 GetBlock()

NAME

GetBlock - Get block as string.

SYNOPSIS

```
Block = GetBlock ( BlockID );
```

```
string GetBlock ( INT );
```

FUNCTION

This function returns the marked, default or named block. Can be called with or without parameter.

INPUTS

BlockID - ID of a valid buffer/block. Specify 0 (zero) to get the default block.

RESULT

This function returns the block as a string. If the block didn't exist, a zero length string ("") is returned (and use GetErrNo() to get the error code).

SEE ALSO

The Block concepts chapter.

1.63 GetBufferID()

NAME

GetBufferID - Get ID of a buffer.

GetEntryID - Get ID of a buffer.

SYNOPSIS

```
ID = GetBufferID ( BufferName, FileNumber, ViewNumber );
```

```
ID = GetEntryID ( BufferName, FileNumber, ViewNumber );
```

```
int GetBufferID ( STRING, INT, INT );
```

```
int GetEntryID ( STRING, INT, INT );
```

FUNCTION

These functions return the buffer/entry ID of the current or specified buffer. The buffer is specified by entering information about the name (which can be the entire path or simply the filename), file number (if there are more than one buffers with the identical name) and view number. The functions can be called with none to three parameters.

INPUTS

BufferName - The name of the buffer.

FileNumber - The number of this file. Only interesting if the buffer name is used in more than one buffer.

ViewNumber - The view number. Only interesting if a DuplicateEntry() has been done to the buffer.

RESULT

The buffer/entry ID, or zero if anything went wrong.

SEE ALSO

CurrentBuffer(), Activate()

The Buffer concepts chapter.

1.64 GetByte()

NAME

GetByte - Get byte position from a given column.

SYNOPSIS

```
BytePos = GetByte ( Column, Line );
```

```
int GetByte ( INT, INT );
```

FUNCTION

Converts from column to byte position of the current or specified

position.

The problem with characters of different widths (like the tab character) is that the column number doesn't tell which character it is on the line. What we refer to as 'Byte position' is the exact number of the position, when each character is one byte.

If no parameter is passed to this function, the current byte position is returned!

This function can be called with none, one or two parameters.

INPUTS

Column - Number of the column to convert into byte position. If this is beyond the end of line, the last column of the line will be read.

Line - Number of the line to read the column from. If it is specified beyond end of file, the last line will be read.

RESULT

The byte position of the given or current position.

SEE ALSO

GetCursor()

1.65 GetChar()

NAME

GetChar - Get character from buffer.

SYNOPSIS

```
Character = GetChar ( Byte, Line );
```

```
int Getchar ( INT, INT );
```

FUNCTION

This function reads a character from the buffer at the current or specified position. This function can be called with none, one or two parameters. Left out parameters defaults to the current position.

INPUTS

Byte - Number of the byte position to read the character from. If this is beyond the end of line or -1, the last character of the line will be read.

Line - Number of the line to read the character from. If it is specified beyond end of file or -1, the last line will be read.

RESULT

The character at given position or -1 if anything failed.

SEE ALSO

GetLine(), GetWord()

1.66 GetCursor()

NAME

GetCursor - Return column position of a byte position

SYNOPSIS

```
Column = GetCursor ( BytePos, Line );
```

```
int GetCursor ( INT, INT );
```

FUNCTION

This function returns in which column the current or specified position is. Can be called with none, one or two parameters.

Useful when checking which cursor position a certain place in a text would get.

INPUTS

BytePos - Number of bytes from the leftmost position of the specified or current line. -1 makes it check the last position of the line.

Line - Line to check. -1 makes it check the last line of the buffer.

RESULT

The column in which that position is on screen.

SEE ALSO

GetByte, GetChar(), GetWord(), GetKey(), GetLine()

1.67 GetDate()

NAME

GetDate - Return date.

SYNOPSIS

```
Date = GetDate ( BufferID, Format );
```

```
string GetDate ( INT, INT );
```

FUNCTION

This function returns the date and time of the current or specified buffer, or even the system time, in the format specified.

When invoked with none or one parameter, it returns the time and date in the format "DD-MMM-YY TT:MM:SS".

INPUTS

BufferID - A valid buffer ID. If set to 0, the current buffer time is returned, if set to -1, the system time is.

Format - Specifies how the time and date should be constructed:
(OR together one value from each group)

value	meaning
~~~~~	~~~~~
(0x000)	return date and time
(0x001)	return date
(0x002)	return time
(0x000)	FORMAT_DOS dd-mmm-yy
(0x010)	FORMAT_INT yy-mmm-dd
(0x020)	FORMAT_USA mm-dd-yy
(0x030)	FORMAT_CDN dd-mm-yy
(0x040)	<locale> yyyy-mm-dd (varies)
(0x000)	DTF_NORMAL
(0x100)	DTF_SUBST substitute Today, Tomorrow, etc.
(0x200)	DTF_FUTURE day of the week is in future

#### RESULT

A string holding the date/time or "".

#### EXAMPLE

Get the system time only, specified internationally with the future flag set:

```
GetDate ( -1, 0x424);
```

SEE ALSO

## 1.68 GetEnv()

#### NAME

GetEnv - Return the value of an environment variable.

#### SYNOPSIS

```
Contents = GetEnv( Name );
```

```
string GetEnv( string );
```

#### FUNCTION

Gets the value of an environment variable.

#### INPUTS

Name - Name of the variable to get.

#### RESULT

The contents of the variable or "" if the variable wasn't found.  
The error number will be set if an empty string is caused because of the variable being absent.

SEE ALSO

`SetEnv(), GetErrNo()`

## 1.69 GetErrNo()

### NAME

`GetErrNo` - Return error number.

### SYNOPSIS

```
ErrorNumber = GetErrNo();
```

```
int GetErrNo();
```

### FUNCTION

When FrexxEd functions fail, most of them set the global error variable. This function returns the value of that variable. This is used to output verbose descriptions to faults.

### INPUTS

### NOTE

To get the correct error number, it should be read immediately after the failing function.

### RESULT

Returns the latest error number set by a FrexxEd function.

### SEE ALSO

`GetReturnMsg(), ReturnStatus()`

## 1.70 GetFileList()

### NAME

`GetFileList` - Get a list of file matching a pattern.

### SYNOPSIS

```
Matches = GetFileList( Pattern, ArrayRef );
```

```
int GetFileList( string, string *[] );
```

### FUNCTION

`GetFileList()` returns the number of directory entries that match the 'Pattern'. All matches are returned in the 'ArrayRef' string array. Entries are files, directories and links to such.

### INPUTS

`Pattern` - A valid AmigaDOS wild card pattern.

`ArrayRef` - A regular FPL string array reference.

### RESULT

The number of matches.

---

## EXAMPLE

A little routine that displays all entries in the FrexxEd: catalog, or a if no entries are present, a requester holding the text "none":

```
string args[1]; /* will be resized to the proper size */
int number; /* holds number of matches */
number=GetFileList("FrexxEd:#?", &args);
if (number) {
    string result;
    RequestWindow("Filer",
                  "", "A", &args, &result, -1);
} else
    Request("None");
```

## SEE ALSO

An AmigaDOS manual for wildcard description!

## 1.71 GetKey()

## NAME

GetKey - Get keypress from user.

## SYNOPSIS

```
Output= GetKey ( Flags );
```

```
string GetKey ( INT );
```

## FUNCTION

This function waits for the user to press a key. As soon as a key press is detected, the keymap string for that press will be returned. If 'Flags' is set to 1, the function will return whatever is pressed right now without waiting. If no key is pressed an empty string will be returned.

## INPUTS

Flags - If bit 0 is set, GetKey() will return without waiting for the keypress.  
if bit 1 is set, the qualifiers of the keypress will be included in the returned string,  
If bit 2 is set, it'll return on the first keypress even if that is a qualifier only.

## RESULT

The string in the keymap that represents the pressed key sequence.

## SEE ALSO

PromptString()

## 1.72 GetLine()

## NAME

GetLine - Return a line from buffer.

---

## SYNOPSIS

```
Line = GetLine ( LineNumber, BufferID );
```

```
int GetLine ( INT, INT );
```

## FUNCTION

This function returns the current or specified line as a string. Can be called with none to two parameters. Specifying a line number out of range will make GetLine() return the last line available.

## INPUTS

LineNumber - line number

BufferID - A valid buffer ID.

## RESULT

The requested line as a string.

## SEE ALSO

GetChar(), GetWord(), GetKey()

## 1.73 GetList()

## NAME

GetList - Get listed information from FrexxEd.

## SYNOPSIS

```
Entries = GetList ( ListName, &Array, [Mask, NonMask, &TypeArray] );
```

```
int GetList ( string, &string/int[], STRING, STRING, &STRING[] );
```

## FUNCTION

GetList() resizes and fills in a string or int array with the contents of an internal list.

These are the current lists available for reading:

extra        returns extra information from a function just invoked.  
             Which information that is returned in the integer array is  
             documented in the specific function.

FACT        lists all FACTs in memory

search      all items in the search/replace history

setting     all info variables. This list is special since it  
             requires three extra parameters to be specified.

symbols     all exported identifiers available at the moment

face        all currently defined faces

facestyles all currently defined face styles

## INPUTS

ListName - Name of the list to get.

Array - Reference to a string or int array. The array will be  
         resized to fit all the items returned. The "extra"  
         list requires an int array, the rest string arrays!

---

These following is used only with the "setting" list:

Mask - Which settings to include. The mask is specified as the 'IncludeMask' for PromptInfo().

NonMask - Which settings to exclude. The mask is specified as the 'ExcludeMask' for PromptInfo().

ToArray - Reference to a string array. This array will be resized to fit a string for each 'Array' entry. Each string will hold the info variable type in the following format (one or more of the following letters):

- 'b' - boolean
- 's' - string
- 'c' - cycle
- 'i' - integer
- 'g' - global
- 'l' - local
- 'w' - writeable (stored in the .default file)
- 'r' - read only
- 'h' - hidden
- 'u' - userdefined

#### RESULT

Returns the amount of items in the array.

#### EXAMPLES

Display all info variables currently present in FrexxEd in a requester to let the user select from the list:

```
string arr[2];
string type[2];
int antal;

string res;

antal=GetList("setting", &arr, "", "", &type);
while (antal) {
    antal--;
    arr[antal]=arr[antal]+" "+type[antal];
}
RequestWindow("Pick a variable", "", "a", &arr, &res, -1);
```

Display all exported identifiers currently present in FrexxEd in a requester to let the user select from the list:

```
{
    string array[1];
    string str;
    int antal;

    antal=GetList("symbols", &array);
    Sort(&array);
    RequestWindow("Title", "", "a", &array, &str, antal);
}
```

SEE ALSO

## 1.74 GetReturnMsg()

NAME

GetReturnMsg - Get verbose error description.

SYNOPSIS

```
Message = GetReturnMsg ( Number );
```

```
string GetReturnMsg ( int );
```

FUNCTION

This function returns a verbose text that explains the error code that was sent as a parameter.

INPUTS

Number - A regular FrexxEd error code.

RESULT

A string holding a verbose description.

SEE ALSO

GetErrNo(), ReturnStatus()

## 1.75 GetWindowID()

NAME

GetWindowID - Get WindowID of window or buffer.

SYNOPSIS

```
WinID = GetWindowID ( Check );
```

```
int GetWindowID ( int );
```

FUNCTION

Returns the Window ID of the current window or of the window that the specified buffer resides.

INPUTS

Check - ID of the buffer or entry you want the Window ID of.

RESULT

Window ID or 0 if anything went wrong.

SEE ALSO

GetBufferID(), CurrentBuffer()

## 1.76 GetWord()

---



## NAME

GetWord - Get current word.

## SYNOPSIS

```
Word = GetWord ( Line, Pos );
```

```
string GetWord ( INT, INT );
```

## FUNCTION

This function returns the word currently under the cursor or at the specified position. Can be called with none, one or two parameters.

If the specified line is not within reach, the current line will be used. If the column isn't within reach, a zero-length string will be returned.

If 'Line' is the only specified parameter, the current pos will be used on the specified line.

## INPUTS

Line - Line number of the word. Use -1 to read current line.

Pos - Byte position of the line. -1 means current pos.

## RESULT

A string holding the word.

## SEE ALSO

GetLine(), GetChar()

## 1.77 GotoChange()

## NAME

GotoChange - Go to a previous change.

## SYNOPSIS

```
Actual = GotoChange ( Number );
```

```
int GotoChange ( INT );
```

## FUNCTION

This function consults the undo buffer, and moves the cursor to the position of the Xth last change, where X is the number specified as parameter to this function. No parameter means that FrexxEd defaults to the very latest change.

## INPUTS

Number - The change number. 1 is the latest change. 2 is the second most recent change and so on...

## NOTE

This function uses the undo buffer to get the information, which gives that earlier changes than the very latest may not have happened exactly where FrexxEd puts the cursor. The position it sets is the position it

---

would get if you would keep undoing until that change.

#### RESULT

The actual change number that it moved the cursor to. Trying to move to a change not existing in the buffer, will make GotoChange() return 0 (zero). If no changes is done to this buffer, 0 will be returned likewise.

#### SEE ALSO

GotoLine()

## 1.78 GotoLine()

#### NAME

GotoLine - Go to a certain line and column.

#### SYNOPSIS

```
fail = GotoLine ( Line, Byte );
```

```
int GotoLine ( INT, INT );
```

#### FUNCTION

This function moves the cursor to the specified line and byte position. If 'Byte' isn't specified, the cursor will be put on the beginning of the line. Specifying no parameter will bring up a requester to be filled up by the user. The requester field accepts a line number and a column number separated with whitespace.

If the specified position is out of range, GotoLine() will put the cursor on the position closest to the request.

#### INPUTS

Line - The line number to jump to. -1 is the same as the last line of the buffer.

Byte - The byte position of the line to jump to. -1 is the same as the last (rightmost) position of the line.

#### RESULT

This function returns zero if the placement of the cursor was put on the exact spot as specified, or non-zero if the cursor couldn't be put on the exact specified spot.

#### EXAMPLE

A function call that move the cursor to the beginning of the current line:

```
GotoLine(ReadInfo("line"));
```

Make the key control cursor up move to the uppermost line of the current buffer:

```
AssignKey("GotoLine(1)", "control 'up'");
```

---

SEE ALSO

## 1.79 Hook functions

### NAME

Hook - Patch a FrexxEd function.  
HookPast - Patch a FrexxEd function.

### SYNOPSIS

```
ret = Hook ( HookName, FPLprogram, Condition );  
ret = HookPast ( HookName, FPLprogram, Condition );
```

```
int Hook ( string, string, STRING);  
int HookPast ( string, string, STRING);
```

### FUNCTION

FrexxEd provides, with these functions, ways for any FrexxEd programmer to change the behaviour of a builtin function. When hooking onto a function, you make your own function get called before the real function. If your function returns a non-zero value, the real function (and following hooks) won't be started. If using HookPast(), the hook function will be called *after* the real function, and the return code simply stop following hooks from being run.

If the optionally specified info variable is set to a non-zero value, the hook will be executed . If no variable is specified (or ""), it will be invoked as described.

It is indeed possible to hook more than one program onto the very same function. They will be interpreted in the hook order (as long as no program return a non-zero value).

These functions can be called with two or three parameters.

See further information in the 'Patching internal functions' chapter of the FrexxEd.guide manual.

Remember *not* to change the original function of a hooked function. That won't do any good, but only confuse.

### INPUTS

HookName - Name of the hook. Valid hooks are all internal FrexxEd function names and an additional amount of 'exceptions'.

FPLprogram - The FPL program that is interpreted when the hook is activated. It can be specified in two ways:

- * One way is to write only the name of the function, no parentheses, no semicolons and no other stuff. Only the name of the function. It is very important that the function accepts the exact same number and kind of parameters as the hooked function does, otherwise it won't be a successful hooking (no error code will be returned since this can't be checked when this function is called, but the hook won't be

run properly later)!

- * The other way to specify the hook is to write a full FPL program to be run.

Condition - Name of the conditional info variable expression. See further details in the description of AssignKey()!

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### SEE ALSO

HookClear()  
Patching internal functions

## 1.80 HookClear()

#### NAME

HookClear - Clear hooks.

#### SYNOPSIS

```
Number = HookClear ( Hook, FPLprogram, Info);
```

```
int HookClear ( STRING, STRING, STRING );
```

#### FUNCTION

This function removes the functions that the parameters match.

If 'Hook' is specified, all hooks on that function will be removed.  
If it isn't specified, it matches all functions.

If 'FPLprogram' is specified, all hooks that calls this function will match.  
If it isn't specified, it matches all hooks.

If 'Info' is specified, all hooks that is dependent on this variable will match. NOTE: This matches *all* hooks dependent on this variable, including those dependent on "!"<variable>".  
If it isn't specified, it matches all hooks.

This function can be called with none to three parameters. Left out parameters will act as if specified as a zero-length string.

#### INPUTS

Hook - Hook name, usually a FrexxEd function name. Specify "" to match all hooks.

FPLprogram - Program to be called when the hook is activated. This can also be a full fledged FPL program. Specify "" to match all functions on this hook.

Info - Variable that the hook is dependent on. Specify "" to match all hooks.

#### RESULT

---

The actual amount of hooks that was cleared.

SEE ALSO

Hook(), HookPast()  
Patching internal functions

## 1.81 Iconify()

NAME

Iconify - Sets FrexxEd in icon state.

SYNOPSIS

```
ret = Iconify ( WinID );  
  
int Iconify ( int );
```

FUNCTION

Closes down a screen/window, and if the 'appicon' info variable is switched on, it creates an appicon. Double-click on the icon, call Deiconify() or send CONTROL-E to FrexxEd to make it normal (deiconified) again. The input 'WinID' can make FrexxEd to iconify just a single window or all windows at once.

INPUTS

WinID - Window ID to iconify, or 0 to iconify all windows.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

Deiconify  
HotKey.FPL

## 1.82 InsertFile()

NAME

InsertFile - Insert a file in buffer.

SYNOPSIS

```
ret = InsertFile ( File, Title, PromptName );  
  
int InsertFile ( STRING, STRING, STRING );
```

FUNCTION

Inserts a file (or a number of files) in the current buffer at the current position. Using no parameter will bring up a requester prompting for which file(s) to include.

If the 'PromptName' field is used, the 'File' parameter is ignored, a filerequester will appear prompting for a file name, and the 'PromptName' string will appear as default choice of the requester.

---

This function uses its own requester to support multiple file selections.

#### CRIPPLE

Unregistered versions of FrexxEd cannot load buffers larger than around 60KB.

#### INPUTS

File - The name of the file to insert. Wildcard is accepted, more than one file can then be inserted at one function call.

Title - Requester window title.

PromptName - Default string choice of the file requester.

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### SEE ALSO

Load()

The IconDrop exception

## 1.83 InverseLine()

#### NAME

InverseLine - Inverse graphics on part of line.

#### SYNOPSIS

```
InverseLine ( Line, Length, Column);
```

```
void InverseLine ( INT, INT, INT );
```

#### FUNCTION

This function inverse the whole or specified part of the line at the specified or current position. It can be called with none to three parameters.

The inverse length is limited to the end of the line which it acts on.

#### INPUTS

Line - Line number to inverse. If this isn't displayed in the view, nothing will happen! Default is current line.

Length - Number of characters to inverse. Default is the entire line.

Column - Start column to inverse. Default is the leftmost, 1.

#### RESULT

---

SEE ALSO

`PrintLine()`, `RedrawScreen()`

## 1.84 FACT reading functions

NAME

`Isclose` - Check if a character is a close. [*]  
`Islower` - Check if a character is lower case. [*]  
`Isnewline` - Check if a character is newline. [*]  
`Isopen` - Check if a character is an open. [*]  
`Isspace` - Check if a character is space. [*]  
`Issymbol` - Check if a character is a symbol. [*]  
`Istab` - Check if a character is tab. [*]  
`Isupper` - Check if a character is upper case. [*]  
`Isword` - Check if a character is word type. [*]

SYNOPSIS

```
ret = IsXXXXXX ( ASCII, FACTname );  
  
int IsXXXXXX ( int, STRING );
```

FUNCTION

These functions return information about the FACT character characteristics.

`Isclose` returns the corresponding open character or -1.  
`Islower` returns the corresponding upper case character or -1.  
`Isnewline` returns non-zero if the character is a newline.  
`Isopen` returns the corresponding close character or -1.  
`Isspace` returns non-zero if the character is a space.  
`Issymbol` returns non-zero if the character is a symbol.  
`Istab` returns non-zero if the character is a tab character.  
`Isupper` returns the corresponding lower case character or -1.  
`Isword` returns non-zero if the character is a word type.

The functions can be called with one or two parameters.

INPUTS

`ASCII` - The character code to check. Valid numbers are 0-255.  
Other input values give undefined results.

`FACTname` - Name of the FACT to check!

RESULT

See description.

SEE ALSO

`FACTString()`, `FACTConvertString()`, `FACT()`

## 1.85 Isfold()

## NAME

Isfold - Check for a fold on a line.

## SYNOPSIS

```
ret = Isfold ( Line, BufferID );
```

```
int Isfold ( INT, INT );
```

## FUNCTION

Returns the fold level of the current or specified line. If the line is a visible fold, the returned value will be negative.

Can get called with none, one or two parameters. Without parameters, the current line will be used.

## INPUTS

Line - Line to check fold level on. 0 means current line.

BufferID - A valid buffer ID.

## RESULT

The fold level. If the line is a viewed fold, the level value will be negative and if there is no fold at all on the line, 0 (zero) will be returned.

## SEE ALSO

The Using folds chapter.

Fold(), FoldHide(), FoldShow()

## 1.86 KeyPress()

## NAME

KeyPress - Returns FPL program assigned to key.

## SYNOPSIS

```
Program = KeyPress ( KeySequence );
```

```
string KeyPress ( STRING );
```

## FUNCTION

Returns the FPL program assigned to the specified key sequence. If no parameter is specified, the key is to be entered interactively.

When the interactive mode is chosen, a key which has no actual function bound to it, will although return a program like 'Output(key):'.

## INPUTS

KeySequence - The key sequence you want the program from.

## RESULT

The FPL program assigned to the key.

## SEE ALSO



`AssignKey(), MenuAdd()`

## 1.87 Kill()

### NAME

Kill - Kill a buffer.

### SYNOPSIS

```
ID = Kill ( BufferID );
```

```
int Kill ( INT );
```

### FUNCTION

This function removes current or specified entry from memory. This function is **not** undoable. It can be called with or without arugment.

If the current entry is killed, the next entry in memory will become the new current.

### INPUTS

BufferID - A valid buffer or entry ID. If an entry ID is specified, only the entry is killed.

### RESULT

Returns the buffer ID of the next buffer in memory.

### SEE ALSO

`New(), CreateBlock(), QuitAll()`

## 1.88 Load()

### NAME

Load - Load buffer(s).

### SYNOPSIS

```
ret = Load ( Filename, Title, PromptName );
```

```
int Load ( STRING, STRING, STRING );
```

### FUNCTION

This function loads the contents of a file into the buffer instead of the current contents. If the input file name is a wildcard matching several files, the first will replace the current buffer and the rest will be loaded into new buffers. When called without parameter, a file requester will be presented which enables multi file selections. If the 'PromptName' field is used, the 'Filename' parameter is ignored, a filerequester will appear prompting for a file name, and the 'PromptName' string will appear as default choice of the reqeuster.

Each matching file (and each file specified without wildcard) will generate a 'GetFile' exception.

---

The pattern used in the requester that appears if no file is specified, can both be set and read with the 'request_pattern' info variable.

**CRIPPLE**

Unregistered versions of FrexxEd cannot load buffers larger than around 60KB.

**INPUTS**

Filename - A standard AmigaDOS search pattern.

Title - Requester window title.

PromptName - Default string choice of the file requester.

**RESULT**

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

**SEE ALSO**

New(), LoadString()  
The GetFile exception

## 1.89 LoadString()

**NAME**

LoadString - Load a file and return as string. [*]

**SYNOPSIS**

```
String = LoadString ( Filename );  
  
string LoadString ( string );
```

**FUNCTION**

This function reads the contents of a specified file and returns it as a string.

**CRIPPLE**

Unregistered versions of FrexxEd cannot load buffers larger than around 60KB.

**INPUTS**

Filename - Name of the file to load.

**RESULT**

The file returned as a string. If something went wrong, an empty string ("" ) is returned.

**SEE ALSO**

Load(), SaveString()

---

## 1.90 LogSetting()

### NAME

LogSetting - Log a variable's default setting. [*]

### SYNOPSIS

```
LogSetting ( Variable, Type, Default );  
  
string LoadString ( string, string, int/string );
```

### FUNCTION

This function logs the default contents of a variable. When that variable is later created, it will get this default value. It is not meant for user programs, but for the initial program generated by FrexxEd.

### INPUTS

Variable - Name of the info variable  
  
Type - Variable type  
  
Default - Default integer or string for the variable

### RESULT

### SEE ALSO

## 1.91 MacroRecord()

### NAME

MacroRecord - Record a macro.

### SYNOPSIS

```
ret = MacroRecord ( InterActive, Name, KeySequence );  
  
int MacroRecord ( INT, STRING, STRING );
```

### FUNCTION

This function toggles between recording and stop recording a macro. Invoke this function to start the recording, and invoke it again to stop recording.

if 'InterActive' is set to 0, a requester will appear, if set to 1, the user have to enter the key sequence interactively, key by key ended with escape. If 'InterActive' is set to 1, and 'Name' isn't set, MacroRecord() will create a unique name for the macro.

When invoked without any parameters, a requester asking for the parameters will appear after the macro has been recorded.

### INPUTS

InterActive - Interactive key sequence mode on/off. Set to 1 to enable interactive mode, 0 to pop up requester.

---

Name - Name of the macro. If not set, MacroRecord() will create a unique name for the macro.

KeySequence - Key sequence as described in AssignKey().

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### SEE ALSO

The block/buffer/macro concepts

## 1.92 MatchParen()

#### NAME

MatchParen - Jump to matching delimiter.

#### SYNOPSIS

```
NoJump = MatchParen ();

int MatchParen ();
```

#### FUNCTION

This function jumps to the matching delimiter as specified in the FACT.

#### INPUTS

#### RESULT

If it performed a jump, a zero is returned, otherwise non-zero.

#### SEE ALSO

## 1.93 MaximizeView()

#### NAME

MaximizeView - Make a view alone on screen.

#### SYNOPSIS

```
MaximizeView ( ViewNumber );

void MaximizeView ( INT );
```

#### FUNCTION

This function makes the current or the specified view number (0 is the uppermost, followed by 1, 2, 3 and so on all the way down to the view visible at the bottom which will have the number NumberOfViews-1). Can be called with or without parameter.

#### INPUTS

ViewNumber - Which view to enlarge.

---

RESULT

SEE ALSO

`RemoveView()`, `ResizeView()`

## 1.94 MenuAdd()

NAME

`MenuAdd` - Add an item to the menu list. [*]

SYNOPSIS

```
ret = MenuAdd ( Type, String, FPLprogram, Keysequence, T, I, S );
```

```
int MenuAdd ( string, string, STRING, STRING, INT, INT, INT );
```

FUNCTION

This function adds a menu item to the current menu list. After a menulist has been completed, `MenuBuild()` creates a menu out of the list, and `MenuClear()` restarts the list from scratch.

Do note the special handling of the 'FPLprogram' parameter to create a boolean item!

`MenuAdd()` is capable of adding an item to a given coordinate of the already existing menu strip. By using up to three integers, you specify which tiel/item/subitem the new item should appear as. Coordinates given as -1 means the last of the kind.

NOTE that separation lines are also counted as an item row.

This function can be called with from two up to seven parameters.

INPUTS

**Type** - Type of the item to add. Any case insensitive starting substring of "Title" will make it a title, of "item" will make it an item, and of "subitem" will make it a subitem. Thus, the words "ti", "S", "ite" and likewise are valid type names.

If the added item should become a boolean type, the type strings to use are the first unique parts of "setting" and "subsetting".

**String** - The string which will appear in the visible menu. Any string holding just a small number of "-" characters, will be converted to a standard menu separator.

**FPLprogram** - A full fledged FPL program to be invoked when this item is selected. Titles and items with subitems can't be selected and should not get FPL programs assigned to them!

If this is a boolean info variable name only, the menu item will become a boolean item which will toggle the specified info variable.

---

Keysequence - A string describing the key sequence that equals selection of this menu item. Shortcut, fastkey. The key sequence will be visible to the right in the menu item.

T - Title number to put the item after/below.

I - Item number to put the item after/below.

S - Sub item number to put the item after/below.

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### EXAMPLES

Add an item that will appear uppermost under the leftmost title in the menu (without key short cut):

```
MenuAdd("i", "FIRST", "First();", "", 1, 1);
```

Create a sub item to the second item of the second title:

```
MenuAdd("s", "SECOND-SUB", "Sub();", "", 2, 2, 1);
```

Add a new title to the right end of the titles:

```
MenuAdd("t", "RIGHTMOST", "", "", -1);
```

Add an item to the right most title, make appear last among the items:

```
MenuAdd("i", "LASTRIGHT", "", "", -1, -1);
```

Add a boolean menu item that toggles the 'insert_mode' variable last among the menu items:

```
MenuAdd("setting", "insert", "insert_mode");
```

#### SEE ALSO

MenuBuild(), MenuClear()

## 1.95 MenuBuild()

#### NAME

MenuBuild - Builds a menu from the menu list.

#### SYNOPSIS

```
ret = MenuBuild ();
```

```
int MenuBuild ();
```

#### FUNCTION

Makes a new menu from the menu list created by calls to MenuAdd(), and then attach it to FrexxEd.

## INPUTS

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

## SEE ALSO

MenuAdd(), MenuClear()

## 1.96 MenuClear()

## NAME

MenuClear - Clears the menu list.

## SYNOPSIS

```
ret = MenuClear ();
```

```
int MenuClear ();
```

## FUNCTION

Clears the menu list and makes following MenuAdd() restart a new one.

## INPUTS

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

## SEE ALSO

MenuAdd(), MenuBuild()

## 1.97 MenuDelete()

## NAME

MenuDelete - Deletes a menu item.

## SYNOPSIS

```
ret = MenuDelete ( Tnum, Inum, Snum);
```

```
int MenuDelete ( int, INT, INT);
```

## FUNCTION

Deletes the specified item from the current menu strip. The specified coordinates (one to three parameters) are the same as used with MenuAdd(). -1 CAN NOT be used to specify anything.

Deleting a title removes all items below it, and deleting an item removes all subitems below that.

## INPUTS

Tnum - Title number you want to read.

Inum - Item number you want to read.

Snum - Sub item number you want to read.

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### SEE ALSO

MenuAdd(), MenuBuild()

## 1.98 MenuRead()

#### NAME

MenuRead - Read details from the menu.

#### SYNOPSIS

```
ret = MenuRead ( TitleRef, NameRef, FPLref, KeyRef, Tnum, Inum, Snum);
```

```
int MenuRead ( string *, string *, string *, string *, int, INT, INT);
```

#### FUNCTION

Reads the specified item from the current menu strip. The specified coordinates (last three parameters) are the same as used with MenuAdd().

#### INPUTS

TitleRef - Reference to a string which will hold the title name.

NameRef - Reference to a string which will hold the item name.

FPLref - Reference to a string which will hold the FPL program.

KeyRef - Reference to a string which will hold the key sequence.

Tnum - Title number you want to read.

Inum - Item number you want to read.

Snum - Sub item number you want to read.

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### EXAMPLE

Using the Menu.FPL menu from the 1.5 release package, the piece:

```
string type, name, fpl, key;  
MenuRead(&type, &name, &fpl, &key, 3, 4);
```

returns information about the "Backspace word" item.

---



SEE ALSO

MenuAdd(), MenuBuild()

## 1.99 New()

NAME

New - Create a new buffer.

SYNOPSIS

```
EntryID = New ();
```

```
int New ();
```

FUNCTION

This function creates a new initialized buffer.

INPUTS

RESULT

Returns the new entry ID, or zero if something went wrong.

SEE ALSO

Clear(), CreateBlock(), Kill()

## 1.100 NextBuffer()

NAME

NextBuffer - Move to next buffer

PrevBuffer - Move to previous buffer

SYNOPSIS

```
BufferID = NextBuffer ( FromBufferID, Mask );
```

```
BufferID = PrevBuffer ( FromBufferID, Mask );
```

```
int NextBuffer ( INT, INT );
```

```
int PrevBuffer ( INT, INT );
```

FUNCTION

These functions return ID of the next/previous buffer in the FrexxEd internal list. If 'Mask' is specified, it will choose the next buffer that matches that bit mask! The functions can be called with none, one or two parameters. Without parameter they default to change from the current buffer.

INPUTS

FromBufferID - Buffer ID of the buffer to change from. 0 is the current buffer. -1 or an illegal value will change from the start of the internal list.

Mask - Buffer selection mask

bit 0 - Regular file buffer

---

bit 1 - macro buffer  
bit 2 - block buffer  
bit 3 - invisible buffer

**RESULT**

Returns the new buffer ID, or zero if something went wrong.

**SEE ALSO**

NextEntry(), PrevEntry()  
The 'type' info variable

## 1.101 NextEntry()

**NAME**

NextEntry - Move to next entry.  
PrevEntry - Move to previous entry.  
NextHidden - Move to next hidden buffer.  
PrevHidden - Move to previous hidden buffer.

**SYNOPSIS**

```
EntryID = NextEntry ( FromEntryID, Mask );  
EntryID = PrevEntry ( FromEntryID, Mask );  
EntryID = NextHidden ( FromEntryID, Mask );  
EntryID = PrevHidden ( FromEntryID, Mask );
```

```
int NextEntry ( INT, INT );  
int PrevEntry ( INT, INT );  
int NextHidden ( INT, INT );  
int PrevHidden ( INT, INT );
```

**FUNCTION**

These functions changes the current entry to the next/previous entry in the FrexxEd internal list. If 'Mask' is specifed, it will chose the next buffer that machthes that bit mask! The functions can be called with none, one or two parameters. Without parameter they default to change from the current entry.

**INPUTS**

FromBufferID - Entry ID of the buffer to change from. 0 is the current entry. -1 or an illegal value will change from the start of the internal list.

Mask - Buffer selection mask  
bit 0 - Regular file buffer  
bit 1 - macro buffer  
bit 2 - block buffer  
bit 3 - invisible buffer

**RESULT**

Returns the new buffer ID, or zero if something went wrong.

**SEE ALSO**

NextBuffer(), PrevBuffer(), NextView(), PrevView()

---

The 'type' info variable

## 1.102 NextView()

### NAME

NextView - Move to next view.  
PrevView - Move to previous view.

### SYNOPSIS

```
View = NextView ( FromView, Mask );  
View = PrevView ( FromView, Mask );
```

```
int NextView ( INT, INT );  
int PrevView ( INT, INT );
```

### FUNCTION

These functions returns the entry ID of the next (lower) or previous (upper) view. The functions can be called one, two or none parameter.

Without parameter, they default to read the one from the current view to any type.

### INPUTS

FromView - Entry ID of the view to move from. 0 means current.  
-1 to NextView() returns entry ID of the bottommost  
-1 to PrevView() returns entry ID of the uppermost

Mask - Buffer selection mask  
bit 0 - Regular file buffer  
bit 1 - macro buffer  
bit 2 - block buffer  
bit 3 - invisible buffer

### RESULT

Returns the new entry ID, or 0 (zero) if the entry wasn't a view.

### SEE ALSO

NextBuffer(), PrevBuffer(), NextEntry(), PrevEntry()  
The 'type' info variable

## 1.103 NextWindow()

### NAME

NextWindow - Move to next window.

### SYNOPSIS

```
Window = NextWindow ( FromWindow );  
  
int NextWindow ( INT );
```

### FUNCTION

Returns the window ID of the next window.

---

Without parameter, it defaults to read the one from the current window.

#### INPUTS

FromWindow - Window ID of the window to move from. 0 means current.

#### RESULT

Returns the new current window ID.

#### SEE ALSO

PrevWindow()

## 1.104 Output()

#### NAME

Output - Print text in buffer.

#### SYNOPSIS

```
Num = Output ( Text );
```

```
int Output ( string );
```

#### FUNCTION

This function outputs the given string at the current position.

#### INPUTS

Text - String to insert in the buffer.

#### RESULT

The number of bytes actually written in the buffer.

#### SEE ALSO

Delete()

## 1.105 PageDown()

#### NAME

PageDown - Move down a screenful of lines.

PageUp - Move up a screenful of lines.

#### SYNOPSIS

```
Actual = PageDown ( Number );
```

```
Actual = PageUp ( Number );
```

```
int PageDown ( INT );
```

```
int PageUp ( INT );
```

#### FUNCTION

These functions move the current location a screenful of lines up or down, one or specified number of times. If the functions are called without parameter it defaults to 1. If the end or beginning of buffer

---

is reach before the screenful lines could be reached, it will put the cursor on the first or last line.

**INPUTS**

Number - The number of PageUps or PageDowns that is to be done.

**RESULT**

The actual number of rounds that were performed before it ended. If the beginning or end of buffer is reached before the 'Number' is reached, it will end.

**SEE ALSO**

ScrollUp(), ScrollDown()

## 1.106 PlaceCursor()

**NAME**

PlaceCursor - Place cursor on screen.

**SYNOPSIS**

```
PlaceCursor ( X, Y );
```

```
void PlaceCursor ( int, int );
```

**FUNCTION**

Put cursor on the specified coordinates in the current view. 'X' is the column and 'Y' is the line counted from the upper left corner.

**NOTE**

If called alone, without parameters, when a mousebutton is pressed, FrexxEd will fill in the parameters before calling this function. This can be used in a line like:

```
AssignKey("PlaceCursor();", "mouseleft");
```

**INPUTS**

X - Horizontal position

Y - Vertical position

**RESULT****SEE ALSO**

GotoLine()

## 1.107 PrevWindow()

**NAME**

PrevWindow - Move to previous window.

**SYNOPSIS**

```
Window = PrevWindow ( FromWindow );
```

```
int PrevWindow ( INT );
```

---

**FUNCTION**

This functions returns the window ID of the previous window.

Without parameter, this default to read the one previous from the current window to any type.

**INPUTS**

FromWindow - Window ID of the window to move from. 0 means current.

**RESULT**

Returns the new current window ID.

**SEE ALSO**

NextWindow()

## 1.108 PrintLine()

**NAME**

PrintLine - Print a string on the screen.

**SYNOPSIS**

```
PrintLine ( Text, Viewline, BufferID );
```

```
void PrintLine ( string, int, INT );
```

**FUNCTION**

The specified text is written in the view. The output does not affect the contents of the buffer but is purely visual.

If you PrintLine() some text and then scroll the screen until that text is no longer visible, there is no way to yet again see that text. The text information is stored on screen only.

**INPUTS**

Text - The text you want in the view.

Viewline - The line number of the view in which you would like to output your text.

BufferID - The ID of the view where you want the text.

**RESULT**

None

**SEE ALSO**

Output(), RedrawScreen()

## 1.109 Prompt()

**NAME**

Prompt - Interactive FPL program execution.

---

## SYNOPSIS

```
Prompt ();

void Prompt ();
```

## FUNCTION

Prompt presents a requester with a list of all in memory existing FPL functions, even functions that has been declared and exported as common FPL code. In the input field of the requester, a FPL program may be entered and when pressing 'OK', the program will be interpreted by the FPL interpreter.

## INPUTS

## RESULT

## SEE ALSO

ExecuteString(), ExecuteFile()  
Introduktion to FPL

## 1.110 PromptBuffer()

## NAME

PromptBuffer - Get buffer from user.  
PromptEntry - Get entry from user.

## SYNOPSIS

```
BufferID = PromptBuffer ( Header, Mask );
EntryID = PromptEntry ( Header, Mask );

int PromptBuffer ( STRING, INT );
int PromptEntry ( STRING, INT );
```

## FUNCTION

These function presents a requester with the 'Header' parameter as window title, holding a list with all buffers/entries currently in memory that fits the type mask.

(The actual difference of buffers and entries are appearant when using these functions. There can be more than one entry to each buffer.)

These functions can be called with none, one or two parameters.

## INPUTS

Header - String to write in the window title.

Mask - Buffer selection mask  
bit 0 - Regular file buffer  
bit 1 - macro buffer  
bit 2 - block buffer  
bit 3 - invisible buffer

---

**RESULT**

Returns the buffer/entry ID of the selected entry, or zero if the requester was cancelled or a non-matching string was entered.

**SEE ALSO**

Activate(), CurrentBuffer(), GetBufferID(), GetEntryID()

## 1.111 PromptFile()

**NAME**

PromptFile - Get file name from user.

**SYNOPSIS**

```
File = PromptFile ( Defaultfile, Header, Pattern, Flags, Array );
```

```
string PromptFile ( STRING, STRING, STRING, STRING, REFERENCE );
```

**FUNCTION**

This function presents a file requester with the 'Header' parameter as window title and the 'Defaultfile' as the name already present in the string gadget, viewing all files matching 'Pattern'. The 'Flags' parameter enables you to make the requester a 'save' requester (thus enabling a 'make dir' button) and/or a 'directory' requester (which only allows a directory to get selected).

If no 'Header' is specified, 'Pick a file' will be used.

This function can be called with none to four parameters.

The pattern used in the requester can both be set and read with the 'request_pattern' info variable.

**INPUTS**

Defaultfile - The default file selection.

Header - String to write in the window title.

Pattern - AmigaDOS pattern to use when viewing files.

Flags - A 's' in this string makes it a 'save' requester.

A 'd' in this string makes it a 'directory' requester.

A 'm' in this string makes it a 'multi' requester. Use it with an string array sent in as the fifth function parameter, and it'll be filled up with all selections. When used, this function will return number of selected files.

Array - String array reference for multi requester returns.

**NOTE**

Only one file/directory name can be selected.

**RESULT**

---



The file/directory name that the user selected/entered, or if anything went wrong or the user didn't select a file, a zero length string ("") is returned.

SEE ALSO

Load(), InsertFile()

## 1.112 PromptFont()

NAME

PromptFont - Get font from user.

SYNOPSIS

```
Font = PromptFont ( Header, Type );

string PromptFont ( STRING, INT );
```

FUNCTION

This function presents a font requester with the 'Header' parameter as window title. The 'Type' parameter specifies whether proportional fonts are allowed selections or not. If no header parameter is specified 'Select a font' will be used, no type parameter will make all fonts selectable.

INPUTS

Header - String to write in the window title.

Type - Font type. 1 - not proportional, 2 - either type.

RESULT

The font will be returned as a string in the format "<font> <size>" since that happens to be how the info variables 'system_font' and 'request_font' want their string to look like! If no font was selected, a zero length string ("") is returned.

SEE ALSO

The 'system_font' and 'request_font' info variables

## 1.113 Prompt for integer or string

NAME

PromptInt - Get integer from user.  
PromptString - Get string from user.

SYNOPSIS

```
Integer = PromptInt ( Title, DefaultInt, Text );
String = PromptString ( Default, Title, Text );

int PromptInt ( STRING, INT, STRING );
string PromptString ( STRING, STRING, STRING );
```

FUNCTION

---

Present a requester to the user and receive the input. The 'Text' will be presented in the body of the requester, or if no such is specified, no text will appear.

These functions both can be called with any number of parameters.

PromptInt() - Requires the user to write an integer number.

PromptString() - Requires the user to write a string.

#### INPUTS

Default - The default string to appear in the requester.

DefaultInt - The default value of the input field.

Text - Text string that appears in the requester, above the input field, below the window title.

Title - The string to appear in the window title.

#### RESULT

PromptInt() returns an integer. If zero is returned, GetErrNo() must be called to see if the return value was due to an error or if the user actually entered '0'.

PromptString() returns string. If a zero length string ("") is returned, GetErrNo() must be called to see if the return string was due to an error or if the user actually didn't enter a string.

#### SEE ALSO

PromptFile(), PromptFont()

## 1.114 PromptInfo()

#### NAME

PromptInfo - Bring up an info variable window.

#### SYNOPSIS

```
PromptInfo( BufferID, Title, IncludeMask, ExcludeMask, Variable, ... );
```

```
void PromptInfo( int, STRING, INT/STRING, INT/STRING, STRING, ... );
```

#### FUNCTION

Brings up a window with the default or specified settings. By altering the 'BufferID' parameter, you can select from Local, All local or Global settings.

By altering the 'IncludeMask', you set the bitmask of which setting types you want in the window. If this is set to 0, no masks are functional (but the default which depends on the 'BufferID' parameter), -1 selects all.

By altering the 'ExcludeMask' you can exclude setting types from the selected ones. Use 0 to not exclude anything.

---

Both masks can negate the masks. That will give you all types but the negated one. You do that by preceeding the mask name with a '!' character within the parentheses.

'Title' is the window title of the setting window.

If 'Variable' is specified with a list of info variables that should be visible in the window. They are added to the one already matched by the masks.

Specify settings to include by simply writing their name (as in "changes"), and exclude settings by preceding the name with a '!' character (as in "!changes"). When excluding, you can specify a '*' as a kind-of wildcard. All settings that match the string before that '*' will be excluded (as in "!colour*").

If no settings match, no requester will appear!

This function can be called with one or more parameters.

#### INPUTS

BufferID - BufferID of the settings.

- 0 Globals (default IncludeMask "(global)" and default ExcludeMask "(local)(read)(hidden)")
- 1 current locals (default IncludeMask "(local)" and default ExcludeMask "(global)(read)(hidden)")
- 2 all locals (default IncludeMask "(local)" and default ExcludeMask "(global)(read)(hidden)")

Title - Window title of the settings requester.

IncludeMask - Which setting types that should be included. Available types are:

- Display - Display matters (eg. 'cursor_x')
- IO - OS interactions (eg. 'expand_path')
- Screen - Screen oriented variable (eg. 'window')
- System - Misc (eg. 'autosave', 'language')
- User - User created info variable
- Hidden - Hidden variable. By default not visualized.
- Global - Global variable.
- Local - Local variable.
- Read - Read-only variable. Cannot get modified.

They should be written within parentheses right next to each other. (see example)

ExcludeMask - Which setting types that should be excluded from the ones that matched the include mask. See 'IncludeMask' for available masks.

Variable - A list of settings to include in the requester window.

#### NOTE

Both masks can negate the masks. That will give you all types but the negated one. You do that by preceeding the mask name with a '!' character within the parentheses. (see example)

**CAUTION**

If too many selections are done, the window will grow very large, and might even become too big to get visualized on the screen of the user!

**RESULT**

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

**EXAMPLE**

Bring up a requester holding all local info variables:

```
PromptInfo(-1);
```

Bring up a requester with all globals including hidden, but excluding "system" variables:

```
PromptInfo(0, "Globals", "(global) (hidden)", "(!system)");
```

Bring up a requester holding three specified variables:

```
PromptInfo(-1, "mixed", -1, -1, "comment", "keymap",  
    "save_icon");
```

**SEE ALSO**

ReadInfo(), SetInfo()  
The settings chapter

## 1.115 Random()

**NAME**

Random - Get a random number.

**SYNOPSIS**

```
value = Random ( );
```

```
int Random ( );
```

**FUNCTION**

Returns a 32-bit random number. This value is to be looked upon as a true random value. It will hardly be the same between two identical FrexxEd, or even system, startups.

**INPUTS****RESULT**

See description

**EXAMPLE**

Display a random line in a requester to the user:

```
{  
    string lines[4] = {  
        "You're a fool!",  
        "Ding, ding, ding",  
        "Press OK quick or your computer will fry!",  
        "Another interesting requester"  
    }  
    int display = Random()%4; /* get number from 0 to 3 */
```

```
        Request( lines[ display ] ); /* pop up requester */  
    }
```

SEE ALSO

## 1.116 RedrawScreen()

NAME

RedrawScreen - Update the screen image

SYNOPSIS

```
RedrawScreen ( Switch );  
  
void RedrawScreen ( INT );
```

FUNCTION

Forces FrexxEd to update the screen. The optional parameter is for future use, nothing but 0 should be entered.

INPUTS

Switch - What to update. Only 0 makes sence today.

RESULT

SEE ALSO

Visible()

## 1.117 QuitAll()

NAME

QuitAll - Quit FrexxEd. [*]

SYNOPSIS

```
QuitAll ();  
  
void QuitAll ();
```

FUNCTION

This is the last function invoke of a running FrexxEd. After a call to this function, FrexxEd frees all buffers and used resources and exits.

INPUTS

RESULT

FrexxEd never returns from this function.

SEE ALSO

Kill()

---

## 1.118 ReadInfo()

### NAME

ReadInfo - Get buffer information. [*]

### SYNOPSIS

```
HoldS = ReadInfo ( Setting, BufferID, [Extra] [, ...] );
```

```
int / string ReadInfo ( string, INT, ... );
```

### FUNCTION

Returns the contents of a specified info variable. It can be of either string, integer, boolean, cyclic kind, and the returning variable type will become that type (cycle and boolean kinds will be returned as integers).

### INPUTS

BufferID - A valid buffer Id number or -1 for current.

Setting - The specified setting to read. Available variables are as follows.

Extra - A possible extra parameter.

### INFO VARIABLES

K is for Kind of info variable:

S - String  
C - Cycle  
I - Integer  
B - Boolean

E is for Existence:

G - Global  
W - Window specific  
L - Local

S is for Specials:

R - Read-only  
W - Write in default file  
H - Hidden

T is for Type:

D - Display  
I - IO  
S - Screen  
Y - System

Name	K	E	S	T	Description
appicon	B	G	W	I	Whether to create an AppIcon.
appwindow	B	W	W	I	Whether to make the window an AppWindow.
arexx_port	S	G	RH	I	FrexxEd's ARexx port name.
auto_resize	B	W	W	S	Whether to ajust the window so that there is no extra pixels below the bottom status line.

autosave B L W I Whether to generate the 'AutoSave' exception.  
 See the "exceptions" chapter.

autosave_interval IGW I The number of changes that is required before  
 the 'AutoSave' exception is generated.

autoscroll B G WH S Whether a screen larger than the display  
 should scroll automatically when the pointer  
 reaches the display edge.

block_begin_x I L R Y Start column of the current block.

block_begin_y I L R Y Start line of the current block.

block_end_x I L R Y End column of the current block.

block_end_y I L R Y End line of the current block.

block_exist C L R Y If a marked block exists.  
 0 - No block exist.  
 1 - A block is marked at the cursor.  
 2 - A block is marked and released from the  
 cursor.

block_id I G RH Y Block ID of the current block.

block_type I L RH Y 1=normal, 2=rectangle

buffers I G R D Number of buffers in memory.

byte_position I L H D The actual byte position of the current line.  
 That means that *all* characters are read as  
 one single byte, independent of what the  
 characters may look like in the editor.

changes I L H Y Number of changes done in the buffer since  
 it was loaded or last saved.

colour I G H Use this like:  
 ReadInfo("colour", bufid, pen_no);  
 Returns the colour-value of the pen specified  
 of the screen the buffer 'bufid' resides on.

colour0 I G WH S Color 0 (4 bits/color RGB) Range: 0 - 15

colour1 I G WH S Color 1

colour2 I G WH S Color 2

colour3 I G WH S Color 3

NOTES about the colour values:

The colour values are in the form 0xrgbRGB where RGB is the upper part  
 and rgb the lower. Thus 0-15 is still valid for least to max, but  
 by using the upper bits we can enhance the resolution.

column I L H D The column number of the current position.

comment S L I File comment to the current buffer.

comment_begin S L W I The string preceding fold information when  
 a file is saved.

comment_end S L W I The string following fold information when  
 a file is saved.

counter I G RH Y This is a continuously increasing counter  
 variable that increases one step on every  
 internal action of FrexxEd. Can be used in  
 functions that prevents invocations twice  
 without nothing in between, and other things.

current_screen S G R S Name of the screen which FrexxEd is opened on.

cursor_x I L RH D The cursor position relative the left border  
 of the view.

cursor_y I L RH D The cursor position relative the upper border

```

    of the view.
default_file  S G R   I Name of the default file that was read when
                  FrexxEd was started.
directory S G W   I Default directory.
disk_name S G HR I Name of the volume mounted by this FrexxEd.
display_id  I G WH S Intuition display ID.
ds_Days     I L H   I Number of days since 1 Jan 1978.
ds_Minute   I L H   I Number of minutes.
ds_Tick     I L H   I Number of ticks.
entries     I G R   S Number of entries that exist in FrexxEd.
expand_path C G W   I How to treat the path name when reading files
                  from disk. 0 = "Off", don't do anything with
                  the path, 1 - "Relative", expand all file
                  names that does not contain any colon ':' and
                  leave assign as they are written, 2 - "All",
                  always expand the entire path to get the
                  *real* name of the file entered. No assigns
                  will ever stay part of the file path! Using
                  the "Off" version makes it possible to change
                  current directory of FrexxEd and then store
                  all files loaded relative in different dirs
                  than they were loaded from. Version "Relative"
                  enables the user to change the destination of
                  an assign in the middle of an editing session
                  and then saved files will use the new dir.
                  "Off" will disable all chances of storing the
                  file in a different dir that it was brought
                  from. It will use the exact volume name, even
                  changing diskette in df0: will be denied!
face         S L W   D Name of the face to use in this buffer.
fact         S L W   S Which FACT to use in the current entry.
file_name    S L R   I File name of the buffer.
file_number  I L R   I Number of the file if there are more than one
                  buffer using the same names.
file_path    S L R   I Directory path to the file.
fold_save    B G W   I Should folds be saved or read when loaded.
fold_width   I G W   D Width of the fold margin. Range 0-8.
fpl_debug    B G     Y FPL execution debug mode. If enabled, all FPL
                  programs executed will be run in debug mode.
                  Read more in the debugging FPL chapter!
fpl_path     S G W   I Directory search path for FPL file
                  executions. Separate each directory with a
                  vertical bar '|'. End all directory names
                  with a colon ':' or slash '/'.
fragmentation I L RH Y Number of fragmentations of the buffer,
fragmentation_size ILRHY Total size of all the fragmentations.
full_file_name S L     I The entire file name including path.
iconify      B G RH D Holds 1 if FrexxEd is iconified!
insert_mode  B L W   D Insert mode on/off. Should written characters
                  insert themselves at the current position or
                  should they overwrite the existing text.
keymap       S G W   I Should be the name of a valid keymap to use
                  within FrexxEd, or not set at all (a zero
                  length string) to use the system default
                  keymap.
language     S G R   Y Which language this system uses! "english"
                  is the built-in.

```

---



line I L R H D Which line number is the current position.  
 line_counter B L W D Line numbering on/off. Line numbering means  
 that the left side of the view will show a  
 line number on every line.  
 line_counter_width I G W D The width of the line numbering field.  
 Default is 5. Range 2 - 8.  
 line_length I L R D The length in number of bytes of the current  
 line.  
 lines I L R D Number of lines in the buffer.  
 macro_key S L Y If this is a buffer created as a macro, this  
 variable contains the key sequence that will  
 execute it.  
 macro_on B G R Y This tells whether macro recording is on.  
 marg_left I G W D Number of character that is left margin.  
 marg_lower I G W D Number of character that is lower margin.  
 marg_right I G W D Number of character that is right margin.  
 marg_upper I G W D Number of character that is upper margin.  
 mouse_x I G R Y In which column of the view the mouse button  
 was pressed. -1 if a button wasn't pressed.  
 mouse_y I G R Y In which line of the view the mouse button  
 was pressed. -1 if a button wasn't pressed.  
 move_screen I G W D Number of characters to move the screen at a  
 time when scrolling horizontally. Min 1.  
 overscan I G W H S Overscan type according to Intuition. Only  
 used when opening FrexxEd as a screen.  
 pack_type S L W I Pack type string. This string should hold a  
 four-letter valid XPK packing type. "PP20"  
 will make FrexxEd use powerpacker when  
 saving. Read more in the 'File handling->  
 Compression' chapter.  
 page_length I G W D Length of a regular page in lines. This is  
 used to calculate page number (optionally  
 displayed on the status line).  
 page_overhead I G W D Number of lines that should overlap when  
 using page up/down functions. Default is 0.  
 password S L I Password used when this file was loaded and  
 also password to use then this file is to be  
 saved. See the 'File handling->Encryption'  
 chapter.  
 pen_block I W W S Pen number to use for block marks.  
 pen_cursor I W W S Pen number to use for the cursor.  
 pen_info I W W S The pen number that should be used for the  
 information text on right half of the status  
 line. -1 gives the same as the left part.  
 popup_view C G W D Which way a new view should be presented on  
 screen. 0 - Replace the current view, 1 -  
 Split the current view or 2 - Open as the  
 only view on screen. Default is 1.  
 protection S L W I The protection bits of the buffer as a string.  
 Each bit has its own letter. They can all be  
 specified in any order. The existing valid  
 letters are as for the AmigaDOS command  
 'protect': "RWEDSPA". Default is "RWED".  
 protectionbits I G H I The protection bits of the buffer.  
 1 - delete  
 2 - execute  
 4 - write

---

```

8 - read
16 - archive
32 - pure
64 - script
(See also "include:dos/dos.h")
public_screen S G R S Which screen FrexxEd should open on and get
information from with the CloneWB() function.
real_screen_height IGR S These 'real_#?' settings are the actual
results of the settings without the 'real_'
prefix. Those without 'real_' tells FrexxEd
how the user wants it to be, these tells the
actual numbers used.
real_screen_width I G R S
real_window_height I G R S
real_window_width I G R S
real_window_xpos I G R S
real_window_ypos I G R S
replace_buffer S G R Y The string in the 'replace buffer'.
req_ret_mode B G HW Y Makes FrexxEd accept the 'return' key to go to
the following field and then 'OK' in dual-
string field requesters like search/replace!
NOTE: this variable is hidden!
request_font S G W S The request and menu font of FrexxEd. Default
is "Topaz 8".
request_pattern S G H I Filerequest pattern. Default is "".
right_mbutton B G W I If this is TRUE, the right mouse button will
not bring up the menus if pressed in a the
FrexxEd view. Only if pressed in the window
title. This allows programming of the right
as well as the left mouse button.
rwd_sensitive B G W I FrexxEd reacts on the protection bits of the
file it loads.
safe_save B G W I If enabled, all save operations in FrexxEd will
first write a temporary file and then rename
it to the destination file name. When this is
disabled, all writes to disk will be done
using the real name at once. Switch off this
setting to enable saving to a filelink.
save_icon C L W I "always" (1) if a .info file should be
created/ updated whenever this buffer is
saved. "never" (0) if no icons should be
created, and "parent" (2) if icons should be
created if the directory the file is written
in has an icon!
screen_depth I G W S Depth of the FrexxEd screen in bitplanes. 1 - 8
(1 = clone the WB depth)
screen_height I G W S Height of the FrexxEd screen in pixels. Min: 100
screen_width I G W S Width of the FrexxEd screen in pixels. Min: 320
search_block B G H Y Search/replace flag.
search_buffer S G R Y The string in the 'search bufffer'. If a third
parameter is given, that place in the search
history is returned.
search_case B G H Y Search/replace flag.
search_flags I G H Y All search and replace flags represented as
an integer. Do not assumpt that the bits
represent flags identical all the time.
search_fold B G H Y Search/replace flag.

```

---

search_forward B G H Y Search/replace flag.  
 search_limit B G H Y Search/replace flag.  
 search_prompt B G H Y Search/replace flag.  
 search_wildcard B G H Y Search/replace flag.  
 search_word B G H Y Search/replace flag.  
 shared I L R Y Number of entries of the same buffer.  
 shared_number I L RH D Number of the entry among the entries to the same buffer.  
 show_path B G W D Show the file path in the status line.  
 size I L R Y Size of the current buffer.  
 slider C G W S How to visualize the vertical slider. 0 - Off, 1 - Right side, 2 - Left side. Default is 1. On windows, it will always be on the right side.  
 startup_file S G W I The FPL file to execute at startup. More than one file can be specified by separating them with '|'. Default is "FrexxEd.FPL".  
 status_line S G W D Status line setup.  
 system_font S G W S FrexxEd's general text font. Default is the system's screen font or "Topaz 8"  
 tab_size I L W D Width of a TAB character. Default is 8. Range 1 - 1000.  
 taskpri I G W Y Priority of the FrexxEd task. Default is 1. Range -25 - 25.  
 topline I L H D The number of the topmost line of the view.  
 top_offset I L H S The line of the window that the current view starts at.  
 type I L Y Buffer type.  
 Bit 0 = standard file buffer  
 Bit 1 = macro buffer  
 Bit 2 = block buffer  
 Bit 3 = invisible buffer  
 undo B L W Y Undo on/off for this buffer.  
 undo_lines I L RH Y Number of lines stored in the undo buffer.  
 undo_max I G W Y Maximum number of bytes that the undo buffer is allowed to use.  
 undo_memory I L RH Y Number of bytes used by the undo buffer.  
 undo_steps I G W Y Maximum number of undo steps stored.  
 unpack B G W I Unpack packed files if possible,  
 version I G R Version number of FrexxEd. This is returned as Version * 10000 + Revision.  
 version_id S G R This FrexxEd's version ID string!  
 view_columns I L RH D Number of columns in the view.  
 view_lines I L RH D Number of lines in the view.  
 views I G RH D Number of views in the window.  
 visible I L RH D Holds non-zero if the entry is visible.  
 window C G W S How to open FrexxEd. 0 - Screen, 1 - Window, 2 - Backdrop window, 3 - WinScreen.  
 window_height I G W S Window height in pixels. Default is 250. Min 18.  
 window_pos C G W S How to position the window when FrexxEd is started. 0 - Visible (FrexxEd will open in the visible area of the screen) 1 - Absolute.  
 window_title S G W S Window title. Only possible to change on registered FrexxEd versions.  
 window_width I G W S Window width in pixels. Default is 640. Min 18.  
 window_xpos I G W S Window x position. Default is 0.  
 window_ypos I G W S Window y position. Default is 0.

---

windows_allocated IGRH S Number of allocated windows.  
windows_open I G RH S Number of opened windows.

**RESULT**

The contents of the setting.

**SEE ALSO**

SetInfo(), PromptInfo()

## 1.119 RemoveView()

**NAME**

RemoveView - Take away a view from screen.

**SYNOPSIS**

```
RemoveView ( );
```

```
void RemoveView ( );
```

**FUNCTION**

This function removes the current or specified view. The view above the removed view will become larger. This function can be called with or without argument.

If the current view is the only one left or the only non-hidden, nothing will happen.

**INPUTS****RESULT****SEE ALSO**

MaximizeView(), ResizeView()

## 1.120 Rename()

**NAME**

Rename - Change name of the buffer.

**SYNOPSIS**

```
Rename ( NewName );
```

```
void Rename ( string );
```

**FUNCTION**

This function changes the name of the current buffer.

**INPUTS**

NewName - The new name of the buffer.

---

RESULT

SEE ALSO

Load(), Save()

## 1.121 Replace()

NAME

Replace - Replaces strings in buffer.

SYNOPSIS

```
Fail = Replace ( Prompt, Search, Replace, Flags, Range );
```

```
int Replace ( INT, STRING, STRING, STRING, INT );
```

FUNCTION

This function searches for the occurrence of the 'SearchString' in the current buffer (or if not specified, the previous SearchString). The found string will be replaced with the 'ReplaceString' parameter (or if not specified, the previous ReplaceString). For details, check out the "Search and Replace" chapter.

All parameters are optional.

INPUTS

Prompt - Specify if you want a query before actually performing any replace. -1 - default, 0 - query, 1 - no query, 2 - replace once. Default is set with the 'search_prompt' info variable through the ReplaceSet() function.

Search - String to search for. A zero length string ("") means that the previous SearchString will be used.

Replace - String to replace the search string with.

Flags - Flags specified as to the ReplaceSet().

Range - Maximum number of bytes to check for match.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

ReplaceSet(), Search(), SearchSet()

Search and replace

Regular expressions

## 1.122 ReplaceMatch()

---

## NAME

ReplaceMatch - Match pattern and generate replacement.

## SYNOPSIS

```
Result = ReplaceMatch ( Pattern, Replacement, Range, Flags );
```

```
string ReplaceMatch ( string, string, INT, STRING );
```

## FUNCTION

Matches the current position with the 'Pattern' specified. If it matches, the 'Replacement' parameter will be used to generate a replacement string which is returned.

'Pattern' is a Regular Expression string.

'Replacement' is a valid wildcard replace string, which means \<num> and \& are supported to insert parts of the match string in the result string.

Setting 'Range' to -1 means until end of buffer.

## INPUTS

Pattern - A FrexxEd Regular Expression string.

Replacement - String to "replace" the 'Pattern' with.

Range - is the number of bytes that this function should be allowed to search to find a match.

Flags - Standard FrexxEd search flags telling the function how to search.

## RESULT

The replace string, if the pattern matched the current position, otherwise an empty string (""). If the replace string can be an empty string, Errno() should be used to check whether it was OK or an error!

## SEE ALSO

Replace(), Search(), SearchSet()  
Search and replace  
Regular expressions

## 1.123 ReplaceSet()

## NAME

ReplaceSet - Set search string, replace string and options.  
SearchSet - Set search string, replace string and options.

## SYNOPSIS

```
ret = ReplaceSet ( Flags, SearchString, ReplaceString );  
ret = SearchSet ( Flags, SearchString, ReplaceString );
```

```
int ReplaceSet ( STRING, STRING, STRING );  
int SearchSet ( STRING, STRING, STRING );
```

---

## FUNCTION

These functions set the search/replace options. The search and replace string and the flags. If invoked with any parameter missing, a requester will be presented to the user to be filled out. For a more detailed description, see the Search and replace chapter.

The flags are readable and modifiable through the info variables named:

Name      Same as Flag:

----      -----

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

## NOTE

This function adds the specified strings to the search history that appears in their requesters.

## INPUTS

SearchString - String to search for. A zero length string ("") means that the previous SearchString will be used.

ReplaceString - String to replace the search string with.

Flags - A string specifying the replace options. Flags are set by specifying '+' after any number of flags, cleared by setting '-' and if '=' is specified, all flags are cleared. The string is read in a strict left-to-right order. Available flags are:

```
w - Enable wildcard search and replace
c - Case sensitive search
o - Only search for entire words
b - Only search within the marked block
f - Search forward (does not affect Replace())
p - Prompt replace
l - Limit wildcards to be line oriented
i - Inside folds
```

'H-' - Will clear the search history.

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

## EXAMPLE

Set the wildcard and switch off the forward flag:

```
SearchSet("=w+f-");
```

## SEE ALSO

Replace(), Search(), ReadInfo()  
Search and replace

Regular expressions

## 1.124 Request()

### NAME

Request - Presents a multi-button requester.

### SYNOPSIS

```
Choice = Request ( Text, Header, Buttons );
```

```
int Request ( STRING, STRING, STRING );
```

### FUNCTION

This function presents a requester with the 'Text' parameter as body text and 'Header' as window title. The buttons are specified in one string separated with a vertical bar sign (|) like: "OK|Cancel" or "OK|Both|Cancel". There is not limit in the number of possible buttons but the width of the screen the requester appears on.

### INPUTS

Text - Body text of the requester

Header - Window title

Buttons - "Button1|Button2|Button3|..."

### RESULT

If the leftmost button was pressed, this function returns 1, and for each consecutive response will return 1 more. The rightmost response will return 0. A five button requester will then return 1,2,3,4,0.

### SEE ALSO

PromptInt(), PromptString(), reqtools.library/rteZRequest()

## 1.125 RequestWindow()

### NAME

RequestWindow - User defined request window.

### SYNOPSIS

```
Ok = RequestWindow ( Title, [Width], Name, Type, Ref, ... );
```

```
int RequestWindow ( string, [INT], string, string, string/int, ... );
```

### FUNCTION

Presents a window with user-defined gadgets. The gadgets can be of different kinds (string, boolean, cycle, integer, array) and the results from the users alterations are received in the variable sent to the function as variable references.

When presenting an array style gadget (listview), a string input field will follow it right below.

---



## INPUTS

Title - Request window title.

Width - If this is a number, this is the default width in characters.

Name - Name of gadget to put in the window.

Type - Gadget type. This is a string holding one or more of the following characters (case insensitive):

"i" - integer

"s" - string

"c" - cycle (requires additional parameter!)

"b" - boolean

"r" - read-only

"a" - array (the 'Name' will be present just above the input field) This tag requires two extra arguments, the default string (of the input field) and the number of elements in the list.  
The variable reference should be a string array!  
If this is used a second time, it forces a second array string field to appear! (To make a name appear above this second field, there must be a name above the first) see example!

Ref - FPL variable reference.

## RESULT

1 if the "OK" button was pressed, otherwise 0.

## EXAMPLE

Present a request window and prompt the user for some informations regarding his/hers personal status:

```
{
  int age=23;    /* age storage variable, 23 is default age! */
  string name="Kjell"; /* name storage variable, "kjell" is
                      default name */
  int sex=0;    /* sex storage variable, default is 0 (male) */
  int stupid=1; /* stupid storage variable, default is 1
               (true) */
  RequestWindow("Status",
    "age", "i", &age,
    "name", "s", &name,
    "sex", "c", &sex, "male|female", /* extra parameter */
    "stupid", "b", &stupid);
}
```

Present a listview, default width is 20 characters:

```
{
    string arrstr="array";
    string arrstr2="array2";
    string arr[6]={"choice 1", "choice 2",
"choice 3", "choice 4",
"FrexxWare", "choice 6"};

    RequestWindow("Window Title", 20,
```

```
        "Right", "A", &arr, &arrstr, 6
        "Left", "A", &arrstr2
    );
}
```

SEE ALSO

`PromptInt()`, `PromptString()`, `Request()`

## 1.126 ResizeView()

NAME

`ResizeView` - Make a view change size.

SYNOPSIS

```
Actual = ResizeView ( NewSize, BufferID );
```

```
int ResizeView ( INT, INT );
```

FUNCTION

Changes the size of the current view or specified view. Can be called with no, one or two parameters.

Brings up a requester if invoked without parameter.

INPUTS

`NewSize` - The new size of the view.

`BufferID` - A valid buffer/entry ID of the view that should be resized.

RESULT

The actual size that the view was set to. May differ to the requested due to limits in screen and window sizes. 0 is returned if the view disappeared.

SEE ALSO

`RemoveView()`, `MaximizeView()`

## 1.127 ReturnStatus()

NAME

`ReturnStatus` - Status line message at return.

SYNOPSIS

```
ReturnStatus ( Text/FailCode );
```

```
void ReturnStatus ( string/int );
```

FUNCTION

This message sets the string that appears at the status line when the program exits. If the parameter is an integer, it is supposed to be a

---

general FrexxEd error code, and the appropriate error message is then being displayed. The message that will appear is the one that is set with `ReturnStatus()` last.

#### INPUTS

Text - A text string to visualize in the status line.

FailCode - A regular FrexxEd error code.

#### RESULT

#### SEE ALSO

`GetErrNo()`, `GetReturnMsg()`

## 1.128 Save()

#### NAME

Save - Write buffer to disk.

#### SYNOPSIS

```
ret = Save ( Filename, Packmode );
```

```
int Save ( STRING, STRING );
```

#### FUNCTION

This function writes the contents of the buffer to disk using the buffer name or the specified name. Using a specified name will **not** rename the buffer. The 'Packmode' parameter enables forced packmodes or non-packmodes.

If you try to 'Save()' an unnamed buffer without the 'Filename' parameter set, a requester will appear prompting for a file name. The buffer will then get renamed to that name.

This function can be invoked with none, one or two parameters.

#### INPUTS

Filename - File name of the file (instead of the default). If "" is specified, the current name is used.

Packmode - A four letter XPK packmode description or "" for no packing at all.

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### SEE ALSO

`Load()`, `SaveString()`

## 1.129 SaveString()

---

## NAME

SaveString - Store string on disk. [*]

## SYNOPSIS

```
ret = SaveString ( Filename, Data );
```

```
int SaveString ( string, string );
```

## FUNCTION

This function writes a string to a file.

## INPUTS

Filename - Full path name of the file to create.

Data - The string to write.

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

## SEE ALSO

Save(), LoadString()

## 1.130 Screenmode()

## NAME

Screenmode - Change screenmode.

## SYNOPSIS

```
ret = Screenmode ( Window );
```

```
int Screenmode ( INT );
```

## FUNCTION

This function brings up a requester to enable the user to change screenmode of one of FrexxEd's screens. This function does not do anything if FrexxEd isn't opened on a screen.

Can be called with or without parameter.

## NOTE

This requester features a 'colour' field which can contain 1 when the requester pops up. If that colour isn't changed, FrexxEd will clone the screen depth of the Workbench screen at startup instead of using a static depth.

## INPUTS

Window - Window ID of which window's screen that should change screenmode.

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

---

SEE ALSO

The settings chapter

## 1.131 Scroll up or down

NAME

ScrollDown - Scroll down screen  
ScrollUp - Scroll up screen

SYNOPSIS

```
Actual = ScrollDown ( Steps );  
Actual = ScrollUp ( Steps );  
  
int ScrollDown ( INT );  
int ScrollUp ( INT );
```

FUNCTION

These functions move the screen upwards or downwards a specified number of steps without letting the cursor change visual row.

INPUTS

Steps - The requested number of steps to scroll. Specifying more than it is possible to scroll, will make it scroll as much as possible.

RESULT

The actual number of steps that the screen scrolled.

SEE ALSO

PageUp(), PageDown()

## 1.132 Search()

NAME

Search - Search for string

SYNOPSIS

```
Fail = Search ( Text, Flags, Range );  
  
int Search ( STRING, STRING, INT );
```

FUNCTION

This function searches for the specified or previously specified substring in the current buffer from the current position. If found, the cursor will be put on the first character of the found string.

NOTE

When using this function and a specified text to search for, no text will be added to the search history that appears in the SearchSet() requester!

INPUTS

---

Text - What to search for.

Flags - Search flags specified as to the SearchSet().

Range - Maximum number of bytes to check for match.

#### RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

Extra information; which line and byte position the found string ends on, can be read with 'GetList("extra", &array[]);'

#### SEE ALSO

SearchSet(), Replace(), ReplaceSet()

Search and replace, GetList()

Regular expressions

## 1.133 SetEnv()

#### NAME

SetEnv - Set the value of an environment variable.

#### SYNOPSIS

```
progress = GetEnv( Name, Contents );
```

```
int SetEnv( string, string );
```

#### FUNCTION

Sets the value of an environment variable. If the variable doesn't already exist, it'll be created before set.

#### INPUTS

Name - Name of the variable to set.

Contents - The new contents of the variable.

#### RESULT

Non-zero if OK, zero if something went wrong.

#### EXAMPLE

Set, and read an environment variable:

```
SetEnv("test", "foobar");  
Request(GetEnv("test"));
```

#### SEE ALSO

GetEnv()

## 1.134 SetInfo()

---

## NAME

SetInfo - Set info variables. [*]

## SYNOPSIS

```
ret = SetInfo ( ID, Setting, Value [, Setting, Value [, ... ] ] );
```

```
int SetInfo ( int, STRING, STRING or INT, ... );
```

## FUNCTION

With this function you can set info variables.

All internal info variables are listed and described in ReadInfo().

Setting a boolean variable to -1 will make it toggle from false to true and from true to false.

## INPUTS

ID - A valid buffer/entry/window ID, or:  
0 to change the default settings for globals and default values  
-1 to change current buffer  
-2 to change all buffer settings

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

## SEE ALSO

ReadInfo(), PromptInfo()

## 1.135 SetSave()

## NAME

SetSave - Save settings

## SYNOPSIS

```
ret = SetSave ( Filename, BufferID );
```

```
int SetSave ( STRING, INT );
```

## FUNCTION

Generate an FPL program with the specified file name that when run will set all saved info variables to their current values.

Invoked without argument (or file specified as "") will bring up a file requester.

Info variables, created by FPL programs, that have their "write" bit set, will get created when that program is run.

If the 'BufferID' parameter is set, the specified buffer's settings will be stored with the local settings. By default (if the parameter is left out or set to 0), the local settings will get their "default" value when stored using this function.

## INPUTS

Filename - Name of the file to write.

BufferID - A valid buffer ID. -1 = current, 0 = default

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

## SEE ALSO

ExecuteFile()

## 1.136 Sort()

## NAME

Sort - Sort an array

## SYNOPSIS

```
ret = Sort ( Array, NumOfItems, Flags );
```

```
int Sort ( &string[], INT, INT );
```

## FUNCTION

This function sorts a one-dimensional string array in a lexicographic order.

## INPUTS

Array - A valid reference to a one-dimensional string array.  
If should be written like '<array name>', i.e if the array variable is called 'foobar' it is '&foobar'.

NumOfItems - If not all items are to get sorted, this specified the amount of sorted items from the start of the array.  
If it is negative, left out or too large, it will sort all items.

Flags - Defines sorting details. Set the following bits to enable the functions:  
0 - case insensitive  
1 - backwards sort

## RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

## EXAMPLE

Sort the three first items in an array:

```
string array[4]={"one", "two", "three", "four"};
```

```
Sort(&array, 3);
```

## SEE ALSO

BlockSort(), BSearch()



## 1.137 Status()

### NAME

Status - Write text on the status line

### SYNOPSIS

```
Status ( EntryID, Text, Update );  
  
void Status ( INT, STRING, INT );
```

### FUNCTION

Outputs the specified string on the default or specified status line, immediately. No string or "" means that the default status line will be updated. The 'Update' parameter control which part of the status line to update or to make the 'Text' appear in.

If the entry isn't visible, the text will appear in the status line at the bottom of the FrexxEd window.

Can be called with none to three parameters.

### INPUTS

ViewID - Entry ID of the status line to write on. 0 is the current view.

Text - The string to write in the status line.

Update - Flags telling which part of the status line to update or put the 'Text' in. Set bit 0 for the left part, bit 1 for the right part or both for both parts!

### RESULT

### EXAMPLE

Update the right part of the current status line:

```
Status(0, "", 2);
```

### SEE ALSO

ReturnStatus()

## 1.138 Stcgfn() and Stgfp()

### NAME

Stcgfn - Returns the file name part of an entire path string.  
Stgfp - Returns the path part of an entire path string.

### SYNOPSIS

```
FileName = Stcgfn ( EntirePath );  
DirName = Stgfp ( EntirePath );  
  
string = Stcgfn ( string );  
string = Stgfp ( string );
```

**FUNCTION**

Stcgfn() returns the file part of a string that specifies the full search path to a file. Stcgfp() returns the path part of the same string.

**INPUTS**

EntirePath - A string holding the full path name of a string. I.e, "disk:foobar/test/myfile.txt".

**RESULT**

The string according the function.

**SEE ALSO**

ReadInfo()

## 1.139 Stricmp()

**NAME**

Stricmp - Compare strings case insensitive. [*]

**SYNOPSIS**

```
Result = Stricmp ( String1, String2, Length );
```

```
int Stricmp ( string, string, INT );
```

**FUNCTION**

This function compares two strings using the ASCII collating sequence, but does not distinguish between uppercase and lowercase.

If 'Length' is specified, not more than 'Length' characters are compared.

The relative collating sequence of the strings is indicated by the sign of the return value, as follows:

Sign	Meaning
----	-----
negative	first string is below the second
zero	strings are equal
positive	first string is above the second

This function can be called with two or three parameters

**INPUTS**

String1 - The string being compared to String2.

String2 - The string being compared to String1.

Length - Number of characters to compare.

**RESULT**

The sign of the return value indicates the relative collating sequence of the strings, as noted above.

**SEE ALSO**

`strcmp()`

## 1.140 StringChangeCase(), StringToLower(), StringToUpper()

### NAME

StringChangeCase - Change case of string.  
StringToUpper - Upper case string.  
StringToLower - Lower case string.

### SYNOPSIS

```
NewString = StringChangeCase ( Original, FACT );  
NewString = StringToUpper ( Original, FACT );  
NewString = StringToLower ( Original, FACT );  
  
int StringChangeCase ( string, STRING );  
int StringToUpper ( string, STRING );  
int StringToLower ( string, STRING );
```

### FUNCTION

These functions convert the specified the input string according to the specified or current FACT.

StringChangeCase() swaps all lower case letters into upper case versions and all upper case letter into lower case versions.

StringToUpper() makes all letters upper case.

StringToLower() makes all letters lower case.

These functions can be called with one or two parameters

### INPUTS

Original - Text to convert.

FACT - Name of the FACT to use.

### RESULT

The converted string.

### SEE ALSO

DownCase(), SwapCase(), UpCase()

## 1.141 StringToBlock()

### NAME

StringToBlock - Copy string to block. [*]  
StringToBlockAppend - Append string to block. [*]

### SYNOPSIS

```
ret = StringToBlock ( String, BlockID );  
ret = StringToBlockAppend ( String, BlockID );
```

---

```
int StringToBlock ( string, INT );  
int StringToBlockAppend ( string, INT );
```

**FUNCTION**

StringToBlock copies the specified string to the default or specified block. StringToBlockAppend appends the specified string to end of the default or specified block. These functions can be called with one or two parameters.

**INPUTS**

String - The string to copy.

BlockID - ID of the destination block. By specifying 0 (zero), the default block will be used.

**RESULT**

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

**SEE ALSO**

StringToClip(), ClipToString(), GetBlock()

## 1.142 StringToClip()

**NAME**

StringToClip - Copy string to clipboard. [*]

**SYNOPSIS**

```
ret = StringToClip ( Unit, String );  
  
int StringToClip ( int, string );
```

**FUNCTION**

Copies the specified string to clipboard.device with the specified unit (likely to be 0 (zero) since interprocess use of clipboard most often uses that unit).

**CRIPPLE**

Unregistered versions of FrexxEd are not able to use this.

**INPUTS**

Unit - clipboard.device unit number.

String - String to send to clipboard.

**RESULT**

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

**SEE ALSO**

ClipToString()

---

## 1.143 Strmfp()

### NAME

Strmfp - Merge a path string with a file string.

### SYNOPSIS

```
Path = Strmfp ( Directory, File );

string Strmfp ( string, string );
```

### FUNCTION

Merges the directory part and the file part and returns the full path name. A slash will be entered as separator if the directory part didn't end with a colon or slash.

### INPUTS

Directory - String holding the directory name.

File - String holding the file name.

### RESULT

The full path name of the merged parts.

### SEE ALSO

Stcgfp(), Stcgfn()

## 1.144 System()

### NAME

System - Perform a system command. [*]

### SYNOPSIS

```
Progress = System ( Command, Input, Output );

int System ( string, STRING, STRING );
```

### FUNCTION

This is a standard system call. 'Command' will be executed as it was written on a standard shell (CLI). The function is synchronous. If you want it to run asynchronously, start the command line with "run".

'Input' and 'Output' are two optional parameters that specify from where to read input and from where to send output. An empty string is like no parameter.

### NOTE

AmigaOS does have a huge (IMHO) weakness in programs launched from WB don't inherit the Workbench process' path. FrexxEd tries to solve that problem by cloning the Workbench's path if invoked from it.

### INPUTS

Command - Shell command line to perform.

Input - File to get input from.

Output - File to send output to.

**RESULT**

Standard AmigaDOS error code from the executed System() call.

**SEE ALSO**

dos.library/System()

## 1.145 TimerAdd()

**NAME**

TimerAdd - Start timer execution.

**SYNOPSIS**

```
TimerID = TimerAdd ( Repeat, FPLprogram, Seconds, Micros );
```

```
int TimerAdd ( int, string, int, INT );
```

**FUNCTION**

Set up a string to get executed after a specified time. The execution can be made to repeat the same interval or to do it only once.

Timer controlled executions will run as fast as possible after the time has passed. FrexxEd's synchronos operations can prevent it to run vituyally as long as it wants to!

**INPUTS**

Repeat - 0 - only once  
1 - repeat this!

FPLprogram - Valid FPL program.

Seconds - Number of seconds before execution.

Micros - Optional number of additional micro seconds before the execution.

**RESULT**

0 if anything went wrong, otherwise a non-zero number to be used for e.g TimerDelete().

**SEE ALSO**

TimerDelete()

## 1.146 TimerDelete()

**NAME**

TimerDelete - Remove a timer execution.

**SYNOPSIS**

```
TimerDelete ( TimerID );
```

---

```
void TimerDelete ( int );
```

**FUNCTION**

Cancels an added timer execution with the specified TimerID. The TimerID specified as parameter is the number received from the TimerAdd() function call.

**INPUTS**

TimerID - A valid TimerAdd() return code. Invalid codes are ignored.  
-1 will cancel *all* timers!

**RESULT****SEE ALSO**

TimerAdd()

## 1.147 Undo()

**NAME**

Undo - Undos changes.

**SYNOPSIS**

```
Actual = Undo ( Steps );
```

```
int Undo ( INT );
```

**FUNCTION**

This function undoes the latest or a specified number of changes. If called without parameter, it defaults to 1.

**INPUTS**

Steps - Number of undos to do.

**RESULT**

The actual number of undos that was performed.

**SEE ALSO**

UndoRestart()  
Undo concept

## 1.148 UndoRestart()

**NAME**

UndoRestart - Restart an undo session.

**SYNOPSIS**

```
Actual = UndoRestart ( Steps );
```

```
int UndoRestart ( INT );
```

**FUNCTION**

---

This function restarts the undo session and performs one or a specified number of undos. This lets you undo previous undos. If called with no parameter, it defaults to one step.

**INPUTS**

Steps - Number of undos to do.

**RESULT**

The actual number of undos that was performed.

**SEE ALSO**

Undo()

Undo concept

## 1.149 Visible()

**NAME**

Visible - Turn visualization on or off.

**SYNOPSIS**

```
Prev = Visible ( Visual );
```

```
int Visible ( INT );
```

**FUNCTION**

This function toggles visibility of what is changed in the buffer. If set to zero (0), nothing will be seen when changed, but if set to non-zero, all changes will be instantly visualized. Multiple changes may be done invisible and result in a great time reduction spent doing those changes. This function can be called without parameter, but will then only return the current state without changing it.

**INPUTS**

Visual - On (1) or off (0).

**RESULT**

Returns the previous visual state, the state which was the current before you called Visible().

**SEE ALSO**

RedrawScreen();

## 1.150 WindowAlloc()

**NAME**

WindowAlloc - Allocate a window data

WindowFree - Free window data

WindowOpen - Open a window

WindowClose - Close a window

**SYNOPSIS**

```
WindowID = WindowAlloc ( BufferID, CloneID );
```

---



```
Result = WindowFree ( WindowID );
Result = WindowOpen ( WindowID );
Result = WindowClose ( WindowID )

int WindowID WindowAlloc ( int, int );
int WindowFree ( int );
int WindowOpen ( int );
int WindowClose ( int )
```

#### FUNCTION

These functions deal with the window system in FrexxEd.

WindowAlloc() is used first to set up window data. WindowOpen() pops up the window previously allocated, WindowClose() closes the visible window and WindowFree() removes the window data from memory completely.

#### INPUTS

BufferID - The ID of the buffer that should appear in the new window.

CloneID - Which window to use as template for how the new window's startup-looks should be.

WindowID - WindowID of the window to affect.

#### RESULT

WindowAlloc() returns a WindowID or 0 if anything goes wrong.

For the rest, they return regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

#### SEE ALSO

NextWindow, SetInfo

## 1.151 WindowToFront()

#### NAME

WindowToFront - Bring FrexxEd to front.

#### SYNOPSIS

```
WindowToFront ( Mode, Id );

void WindowToFront ( INT, INT );
```

#### FUNCTION

This function asks intuition to bring FrexxEd's current or specified screen and window in front of all other screens and windows. This function can get called with no, one or two parameters. When called without parameter, it acts as if called with Mode set to '0'.

#### INPUTS

Mode - 0 bring window to front and activate window  
1 simply bring window to front without activating it

---

```
2 simply activate the window
-1 Make a WindowToBack/ScreenToBack and deactivate the window
```

Id - Window ID of the Window to manipulate.

RESULT

SEE ALSO

`intuition.library/WindowToFront()`

## 1.152 Yank()

NAME

Yank - Paste the yank buffer.

SYNOPSIS

```
ret = Yank ( Number );
```

```
int Yank ( INT );
```

FUNCTION

This function outputs the yank buffer a specified number of times. By default, one. The yank buffer was constructed by multiple character deleting functions that were invoked in one uninterrupted sequence. This function can be called with or without parameter.

INPUTS

Number - Number of outputs.

RESULT

Regular FrexxEd error code. If zero or a positive number is returned, everything is OK, otherwise something went wrong.

SEE ALSO

`Delete()`, `DeleteLine()`

---