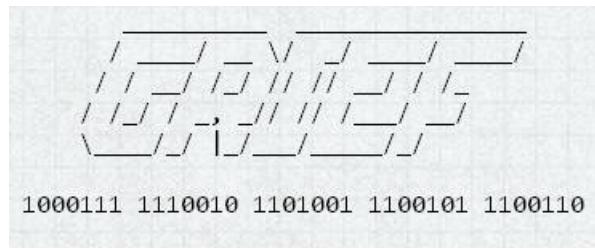




GRIEF

Quick Start and Programmers Guide





The Grief Editor

Quick Start and Programmers Guide

Version 3.3

Copyright © Adam Young 2014 - 2024
All Rights Reserved. No parts of this documentation maybe reproduced without permission in writing from the author.

All product names mentioned herein are the trademarks or registered trademarks of thier respective owners.

The author(s) have taken care in the preparation if this documentation, but make no expressed or implied warranty and assumes no reponsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Table of Contents

Table of Contents	3
Introduction	16
GRIEF - The Glorious Reconfigurable Interactive Editing Facility	16
Macro Language differences	17
History	19
Authors	19
Brief	19
Copyright	21
Documentation	21
GRIEF Software License	21
Quick Start	23
Setup	23
Basics	24
Main Screen	24
Movement	26
Help	26
Text Editing	27
Cut and Paste	28
Search and Replace	29
Undo and Redo	30
Command Prompt	30
File and Buffer Manipulation	31
Window Manipulation	32
Record and Playback	33
Features	33
Macro Tutorial	35
GRIEF Macros	35
Compilation Model	35
A Quick Macro Tutorial	36
Compiler Usage	38
Compatibility	39
Language Specification	40
Lexical elements	41
Tokens	45
Literals	46
Types	52
Expressions	57
Declarations	64
Statements	70
Macros	78
Types of Macros	78
Macro Resources	80
Regular Expressions	86
Preprocessor	87
Implementation	87
Directives	87
Conditional Compilation	89
Condition evaluation	90
Replacment Macros	90
Predefined Symbols	91
File Inclusion	91
Message Generation	91
Source Information	92
Debugging	94
Macro Errors	94
Compiler Errors	94
Library Reference	95
Return Value	95
Function Reference	95
Arithmetic Operators	107

Macros	108
above	108
above_eq	108
abs	108
acos	109
asin	109
atan	109
atan2	109
below	110
below_eq	110
ceil	111
compare	111
cos	111
cosh	112
cvt_to_object	112
exp	113
fabs	113
floor	114
fmod	114
frexp	114
isclose	115
isfinite	115
isinf	115
isnan	116
ldexp	116
log	116
log10	117
modf	117
pow	117
sin	118
sinh	118
sqrt	118
tan	118
tanh	119
Functions	119
!	119
!=	119
%	120
%=	120
&	120
&&	121
&=	121
*	121
*=	122
+	122
++	122
+=	123
-	123
--	123
-=	124
/	124
/=	124
<	124
<<	125
<<=	125
<=	125
<=>	126
=	126
==	126
>	127
>=	127
>>	127
>>=	128
^	128
^=	128

post++	129
post-	129
	129
!=	129
	130
~	130
Buffer Primitives	132
Constants	132
Buffer Flags	133
Macros	134
attach_buffer	134
create_buffer	135
create_nested_buffer	135
delete_buffer	136
delete_char	136
delete_line	137
delete_to_eol	137
find_line_flags	137
find_marker	138
goto_bookmark	138
inq_attribute	139
inq_buffer_flags	140
inq_buffer_title	140
inq_buffer_type	140
inq_byte_pos	141
inq_encoding	142
inq_file_change	142
inq_indent	143
inq_line_flags	143
inq_line_length	143
inq_lines	144
inq_margins	144
inq_modified	144
inq_names	145
inq_position	145
inq_process_position	146
inq_ruler	146
inq_system	147
inq_tab	147
inq_tabs	147
inq_terminator	148
inq_time	148
mark_line	148
mode_string	149
next_buffer	150
previous_buffer	151
print	151
set_attribute	151
set_buffer	152
set_buffer_flags	152
set_buffer_title	153
set_buffer_type	153
set_encoding	154
set_indent	157
set_line_flags	157
set_margins	158
set_process_position	158
set_ruler	159
set_tab	159
set_terminator	160
sort_buffer	160
tabs	161
tagdb_close	162
tagdb_open	162
tagdb_search	163

write_buffer	163
Callbacks	165
Macros	165
_bad_key	165
_chg_properties	165
_default	166
_extension	166
_fatal_error	166
_init	167
_invalid_key	167
_prompt_begin	168
_prompt_end	168
_startup_complete	168
main	169
Debugging Primitives	170
Macros	170
abort	170
debug	170
debug_support	171
dprintf	172
error	172
inq_debug	173
message	173
pause_on_error	176
pause_on_message	177
printf	178
profile	179
watch	179
Dialog Primitives	180
Macros	180
dialog_create	180
dialog_delete	182
dialog_exit	182
dialog_run	183
inq_dialog	183
widget_get	184
widget_set	184
Environment Primitives	185
Macros	185
cd	185
create_char_map	186
get_term_characters	187
get_term_feature	188
get_term_features	188
get_term_keyboard	189
getenv	189
geteuid	189
getpid	190
getuid	190
getwd	190
inq_backup	190
inq_backup_option	191
inq_brief_level	191
inq_char_timeout	192
inq_environment	192
inq_feature	192
inq_hostname	193
inq_idle_default	193
inq_idle_time	193
inq_profile	194
inq_username	194
putenv	194
set_backup	195
set_backup_option	195

set_char_timeout	196
set_idle_default	197
set_term_characters	197
set_term_feature	198
set_term_features	201
set_term_keyboard	202
uname	204
File Primitives	205
Macros	206
access	206
basename	206
chdir	206
chmod	207
chown	207
compare_files	207
copy_ea_info	208
del	208
dirname	208
edit_file	209
edit_file2	210
exist	211
expandpath	211
fclose	211
feof	212
ferror	212
fflush	212
file_canon	213
file_match	213
file_pattern	215
filename_match	215
filename_realpath	216
find_file	216
find_file2	217
find_macro	218
fioctl	218
flock	218
mktemp	218
fopen	219
fread	220
fseek	220
fstat	220
ftell	221
ftest	221
ftruncate	222
fwrite	222
glob	223
inq_buffer	223
inq_file_magic	224
inq_home	224
inq_tmpdir	224
inq_vfs_mounts	225
link	225
lstat	226
mkdir	227
mktemp	227
output_file	227
read_ea	228
read_file	228
readlink	229
realpath	229
remove	229
rename	230
rmdir	230
searchpath	230
set_binary_size	231

set_ea	231
set_file_magic	231
stat	234
symlink	235
umask	235
unlink	236
vfs_mount	236
vfs_unmount	237
File Modes	237
Keyboard Primitives	239
Macros	239
assign_to_key	239
copy_keyboard	241
get_mouse_pos	241
input_mode	243
inq_assignment	243
inq_kbd_char	243
inq_kbd_flags	244
inq_kbd_name	244
inq_keyboard	245
inq_keystroke_macro	245
inq_keystroke_status	245
inq_local_keyboard	246
inq_mouse_action	246
inq_mouse_type	246
inq_remember_buffer	247
int_to_key	247
key_list	248
key_to_int	248
keyboard_flush	250
keyboard_pop	250
keyboard_push	251
keyboard_typeables	253
load_keystroke_macro	253
pause	253
playback	254
process	254
push_back	254
read_char	255
remember	255
save_keystroke_macro	256
set_kbd_name	256
set_mouse_action	257
set_mouse_type	257
undo	257
use_local_keyboard	258
List Primitives	260
Macros	260
car	260
cdr	261
command_list	261
delete_nth	261
file_glob	262
get_nth	262
length_of_list	263
list	263
list_each	263
list_extract	264
list_reset	264
make_list	265
nth	265
pop	266
push	266
put_nth	267
quote_list	267

search_list	268
shift	268
sort_list	269
splice	270
strlen	271
strnlen	271
unshift	272
Macro Language Primitives	273
Constants	274
BPACKAGES	274
GRBACKUP	275
GRDICTIONARIES	275
GRDICTIONARY	276
GRFILE	276
GRFLAGS	276
GRHELP	276
GRINIT_FILE	277
GRKBDPATH	277
GRLEVEL	278
GRLOG_FILE	278
GRPATH	278
GRPROFILE	279
GRRESTORE_FILE	279
GRSTATE_DB	279
GRSTATE_FILE	280
GRTEMPLATE	280
GRTERM	280
GRTERMCAP	281
GRTMP	281
GRVERSIONMAJOR	281
GRVERSIONMINOR	282
GRVERSIONS	282
errno	282
Macros	283
__breaksw	283
autoload	283
bless	284
break	284
call_registered_macro	285
case	285
catch	285
continue	286
create_dictionary	286
delete_dictionary	286
delete_macro	287
dict_clear	287
dict_delete	288
dict_each	288
dict_exists	289
dict_keys	289
dict_list	289
dict_name	290
dict_values	290
do	291
else	291
execute_macro	291
exit	292
finally	292
first_time	293
for	293
foreach	294
fstype	294
get_property	294
if	295
inq_btn2_action	295

inq_called	296
inq_macro	296
inq_macro_history	297
inq_module	297
inq_msg_level	298
list_of_dictionaries	298
load_macro	298
macro	299
module	299
nothing	300
register_macro	301
reload_buffer	302
replacement	302
require	303
reregister_macro	303
restore_position	304
return	304
returns	304
save_position	305
set_btn2_action	305
set_calling_name	306
set_macro_history	306
set_property	306
switch	307
throw	308
try	308
unregister_macro	308
while	309
Miscellaneous Primitives	310
Macros	310
__regress_op	310
__regress_replacement	310
beep	311
color_index	311
date	311
gmtime	312
grief_version	312
inq_clock	313
localtime	313
process_mouse	314
rand	314
set_msg_level	314
srand	315
time	315
version	316
Movement Primitives	318
Macros	318
backspace	318
beginning_of_line	319
bookmark_list	319
delete_bookmark	319
down	320
drop_bookmark	320
end_of_buffer	321
end_of_line	321
end_of_window	322
goto_line	322
goto_old_line	322
left	323
move_abs	323
move_rel	323
next_char	324
page_down	324
page_up	325
parse_filename	325

prev_char	326
swap_anchor	326
top_of_buffer	326
top_of_window	327
Process Management Primitives	328
Macros	328
connect	328
disconnect	329
dos	329
inq_connection	330
insert_process	330
perror	331
send_signal	331
shell	332
sleep	334
strerror	334
strsignal	334
suspend	335
wait	335
wait_for	336
Signals	336
Scrap Primitives	338
Macros	338
copy	338
cut	338
delete_block	339
drop_anchor	339
end_anchor	340
get_region	340
inq_mark_size	341
inq_marked	341
inq_marked_size	342
inq_scrap	342
mark	342
paste	343
raise_anchor	343
redo	344
set_scrap_info	344
transfer	344
write_block	345
Screen Primitives	347
Macros	347
borders	347
color	348
copy_screen	348
cursor	349
display_mode	349
display_windows	350
echo_line	351
ega	352
get_color	353
inq_borders	354
inq_cmd_line	355
inq_color	355
inq_command	356
inq_display_mode	356
inq_echo_format	357
inq_echo_line	357
inq_font	357
inq_line_col	358
inq_message	358
inq_prompt	358
inq_screen_size	359
inq_window_color	359

inq_window_priority	359
inq_window_size	360
redraw	360
refresh	361
screen_dump	361
set_color	361
set_echo_format	366
set_window_priority	368
use_tab_char	369
view_screen	369
window_color	369
Search and Translate Primitives	371
Macros	371
quote-regexp	371
re_comp	371
re_delete	372
re_result	372
re_search	373
re_syntax	374
re_translate	375
search_back	376
search_case	377
search_fwd	377
translate	378
Spell Checker Primitives	380
Macros	380
spell_buffer	380
spell_control	380
spell_dictionary	381
spell_distance	381
spell_string	382
spell_suggest	383
String Primitives	384
Macros	384
atoi	384
ctime	385
characterat	385
compress	386
diff_strings	386
firstof	387
format	387
index	388
isalnum	388
isalpha	389
isascii	389
isblank	390
iscntrl	390
iscsym	391
isdigit	392
isgold	392
isgraph	393
islower	393
isprint	394
ispunct	394
isspace	395
isupper	395
isword	396
isxdigit	397
itoa	397
lastof	397
lower	398
ltrim	398
macro_list	398
read	399

rindex	399
rtrim	400
search_string	400
split	401
split_arguments	401
sprintf	402
sscanf	402
strasecmp	404
strcasestr	405
strcmp	405
strfilecmp	405
strftime	406
string_count	407
strpbrk	407
strpop	408
strrstr	408
strstr	408
strtod	409
strtof	409
strtol	409
strverscmp	410
substr	411
tokenize	411
trim	412
upper	412
ctype	413
Syntax Highlighting Primitives	414
Macros	414
attach_syntax	414
create_syntax	414
define_keywords	415
detach_syntax	416
get_color_pair	416
hilite_create	417
hilite_delete	417
hilite_destroy	418
inq_hilite	418
inq_syntax	418
set_color_pair	419
set_syntax_flags	419
syntax_build	420
syntax_column_ruler	420
syntax_rule	421
syntax_token	423
Variable Declaration Primitives	425
Macros	425
__lexicalblock	425
arg_list	426
array	426
bool	427
const	427
declare	427
double	428
extern	428
float	429
get_parm	430
getopt	431
getsubopt	433
global	434
inq_symbol	435
int	436
is_array	436
is_float	436
is_integer	437
is_list	437

is_null	437
is_string	438
is_type	438
make_local_variable	438
put_parm	439
ref_parm	440
register	441
static	441
string	442
typeof	442
Window Primitives	444
Macros	444
change_window	444
change_window_pos	445
close_window	445
create_edge	446
create_menu_window	446
create_tiled_window	447
create_window	447
delete_edge	448
delete_window	448
distance_to_indent	449
distance_to_tab	449
inq_char_map	449
inq_ctrl_state	450
inq_mode	450
inq_top_left	451
inq_views	451
inq_window	452
inq_window_buf	452
inq_window_flags	452
inq_window_info	453
inq_window_infox	454
insert	454
insert_buffer	455
insert_mode	455
insertf	455
move_edge	456
next_window	456
right	457
self_insert	457
set_buffer_cmap	458
set_ctrl_state	458
set_feature	459
set_font	459
set_top_left	460
set_window	460
set_window_cmap	461
set_window_flags	461
set_wm_name	462
translate_pos	462
up	463
Third Party Packages	464
Common Modules	464
Crisp	465
ND+ - Natural Docs Plus	465
ucpp	465
makedepend	466
mandoc	466
extags	467
flex	467
Oniguruma	468
TRE	469
libregex	470

libteken	470
libmagic	471
libcharudet	471
libguess	472
hunspell	473
libarchive	474
iconv	475
libz	475
libbz2	476
liblzma	477
Appendix A - Error Codes	479
errno	479
Appendix B - Backus Naur Form	481
Syntax	481
Appendix C - ASCII Chart	486
Appendix D - BRIEF Macros	489
INDEX	493

Introduction

Summary

Introduction

GRIEF - The Glorious Reconfigurable Interactive Editing Facility

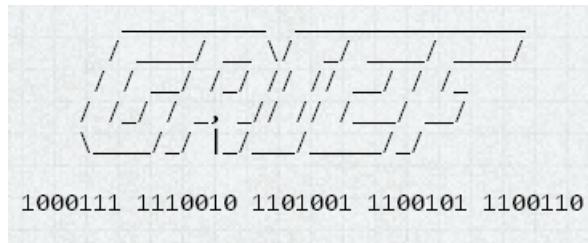
Macro Language differences

Firstly **GRIEF** is primarily a source editor and is designed for manipulating pieces of code, although it is capable of formatting and printing documents like a word processor.

On the face of things **GRIEF** looks and feels like standard C, yet the following differences between the standard implementation of the C language and the **GRIEF** Macro language should be observed.

GRIEF - The Glorious Reconfigurable Interactive Editing Facility

Firstly **GRIEF** is primarily a source editor and is designed for manipulating pieces of code, although it is capable of formatting and printing documents like a word processor.



It provides commands to manipulate words and paragraphs, syntax highlighting for making source easier to read, and *macros* for performing user-defined batches of editing commands.

Supported Platforms

Current support platforms.

- Works on most POSIX-like systems (Linux, Solaris, AIX, HP/UX) in either a console or X11/xlib.
- Windows (Native, Cygwin and MinGW).
- Mac OS/X (last build 2012).
- DOS (DJPGPP - last build 2002; minor work required).

What is a Macro

The *GRIEF Macro language* gives you the ability to modify and secondary extend much of this editing environment to suit your needs.

The language syntax is based on a C-style module which should allow most users to quickly familiarise themselves.

Sections

The following sections detail the specifics.

- History
- Quick Start
- Macro Tutorial
- Language Specification
- Library Reference
- Preprocessor

Goals

GRIEF has a long history and is being maintained and developed with the primary goal of being an enhanced console based **BRIEF** clone.

The development of GRIEF is an on going process, with these some of the expected additional features.

Future Features

- LUA language binding; providing access to the large set of pre-existing LUA based applications.
- Macro debugger; similar to the functionality available within the original BRIEF implementation.
- Minimal GUI interface, most likely QT5+ based.
- Alternative syntax highlighting engine(s); examples include *GtkSourceView*, *highlighter* and *Kate*.
- Code Folding.
- CLang integration, code completion.
- JavaScript binding, most likely SpiderMonkey or V8; the long term goal making Javascript the primary language for new macros.

- EditorConfig support (<http://editorconfig.org>)
- plus may more ...

Macro Language differences

On the face of things **GRIEF** looks and feels like standard C, yet the following differences between the standard implementation of the C language and the **GRIEF** Macro language should be observed.

Data Types

Only the standard C data-types *int* and *float/double* are supported.

In addition to these **GRIEF** supports a *list* and *string* type, plus a polymorphic container type *declare* which may hold any of the supported data-types.

No pointers, complex types ('*struct*, *unions* and bit-fields) nor *typedefs* are supported.

The *register* storage class is not supported.

GRIEF does not support the C style cast mechanisms; note automatic *parameter coercion* shall occur when passing arguments to functions.

External declarations

Both function and variable declarations maybe hidden within the scope of a named module. There is no equivalent construct of module's in C, yet these can be compared to C++ namespaces.

Function declarations

A functions parameter maybe declared as optional regardless of their position within the parameter list, unlike C which only allows the trailing.

The NULL value is passed by the caller to represent an argument which was omitted.

Similar to C++, parameters maybe declared as references and assigned default values.

Function parameters

A functions parameter can be declared without specifying its names. Instead parameters can be retrieved and optionally prompt the user anywhere in the function with the primitive *get_parm*; optional parameters may only be handled in this fashion.

In addition passed parameters can be modified using the *put_parm* primitive.

Similar to C++, reference parameters can be created using the primitive *ref_parm*.

Furthermore, as the result of this interface parameters are passed using *lazy evaluation*. That is the arguments to a function are not evaluated at the time a function is called, as is the case with the C language. Instead they are evaluated at the time they are referenced within the invoked function, like with most Lisp style languages. This may leading to what may seem at times unexpected results, so time should taken to understand this concept (See: [Lazy Evaluation](#)).

Replacement Functions

There is no equivalent construct of replacement functions in the C language. C++ function overloading is a similar construct.

GRIEF Function declarations

Internal primitives need not be prototyped, as their definitions are builtin to the compiler.

Variable Scoping

Within **GRIEF** variable access utilises *dynamic scoping* at run-time; dynamic scoping is similar to the scoping rules of *Pascal* rather than C permitting macros to access variables declared within their callers (See: [Scope](#)).

Goto Statement

The *goto* statement is not implemented.

Switch Statement

Unlike the C language, switch statement control does not flow from one case group to next. In other words there is an implied *break* at the end of each case group; when the statements executed against the group are complete the switch statement is exited.

The *break* statement can be used at the end of each case group to denote this fact; yet is only needed to terminate loops constructs (ie. *for*, *foreach*, *do* and *while*).

Unlike C, string literals maybe be used for either the switch-expression or the case-expressions.

Enumerations

GRIEF permits both integer and string-literals to be assigned to an enumeration constant.

sizeof

The *sizeof* statement is not implemented.

Dynamic memory allocation and pointers

Either the memory allocation functions *malloc*, *calloc*, *realloc* nor *free* are available, in addition pointers are not supported.

Dynamic components can be managed using list's.

Miscellaneous

Both C multi-line and C++ single-line comments are supported.

Trigraphs are not supported.

\$Id: introduction.txt,v 1.6 2014/11/27 18:08:41 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

History

The **GRIEF** Editor, despite the fact this package was only formally released in 2013, its development has its roots from the 80's as a Brief clone. This version of the editor has been actively used since the 1998.

Summary

History

The **GRIEF** Editor, despite the fact this package was only formally released in 2013, its development has its roots from the 80's as a Brief clone.

Authors

Brief

As for its development the main history milestones and authors are as follows.
Brief, **BRIEF**, or **B.R.I.E.F.**, an acronym for *Basic Reconfigurable Interactive Editing Facility*, was a programmer's text editor in the 1980s and early 1990s.

Authors

As for its development the main history milestones and authors are as follows.

Dave Conroy

Author of the public domain **MicroEmacs** upon which **GRIEF** was originally based.

MicroEMACS is supported on a variety of machines and operating systems, including MS-DOS VMS and UNIX (several versions); which can be found within CUG archives.

Since that time a number of variations have been developed, such as MicroGNUEmacs (or mg) and uEmacs/PK.

Paul Fox

Using MicroEmacs as a basis **Paul Fox** developed the original Crisp as an UnderWare Inc's, later Borland 3.1 **BRIEF™** clone/emulation, targeted for Unix™ and VMS.

This package was known as **CRISP**, the *Custom Reduced Instruction Set Programmers Editor*.

The last public release was Crisp 2.2 in 1991 prior to becoming that is now **CRISP** Visual Text Editor,

Paul Fox's personal site is <http://www.crisp.demon.co.uk>, with the commercial site being <http://www.crisp.com>.

CRISP is a programmers text editor designed to give users the power and flexibility to edit large files on multiple Unix, Linux, Windows and Mac platforms.

CRISP started life as a programmers text editor with BRIEF emulation, however after 15+ years of development, it now includes just about every conceivable editing feature that you could ever feel a need for, while still maintaining BRIEF keyboard and macro emulation.

Paul Fox no longer maintains nor supports the Crisp 2.2 version.

Crisp was based in part on a mix of public domain and specialised components, including MicroEmacs.

The final source and several earlier versions can be found on old mail archive sites.

GRIEF has no connection with the current commercial product.

Adam Young

Heavily modified, replaced and extended the publicly released crisp2.2 to become what is now the **GRIEF** Editor.

Brief

Brief, **BRIEF**, or **B.R.I.E.F.**, an acronym for *Basic Reconfigurable Interactive Editing Facility*, was a programmer's text editor in the 1980s and early 1990s.

It was designed and developed by UnderWare Inc, a company founded in Providence, Rhode Island by *David Nanian* and *Michael Strickman*, and was published by *Solution Systems* and later Borland.

In 1990, UnderWare sold BRIEF to Solution Systems which released version 3.1.

Borland later purchased BRIEF and the full suite of software tools from Solution Systems.

Solution Systems closed permanently after the sale to Borland. Much to our loss, BRIEF is no longer sold by Borland.

\$Id: history.txt,v 1.3 2014/10/31 01:09:05 ayoung Exp \$

To send feedback on this topic **email:** griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Copyright

Summary

Copyright

Documentation

Information in this document, including URL and other Internet Web site references, is subject to change without notice.

GRIEF Software License

GRIEF is © 1998-2024, Adam Young.

Documentation

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of the copyright holders or within the rights stated by the license.

Somebody may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Somebody, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

GRIEF Software License

GRIEF is © 1998-2024, Adam Young.

GRIEF is distributed under the terms of the *Q Public License version 1.0 (QPL-1.0)*; with a change to choice of law, see below.

Source, <http://opensource.org/licenses/QPL-1.0>

The Q Public License Version 1.0 (QPL-1.0)

Copyright © 1999 Trolltech AS, Norway.

Everyone is permitted to copy and distribute this license document.

The intent of this license is to establish freedom to share and change the software regulated by this license under the open source model.

This license applies to any software containing a notice placed by the copyright holder saying that it may be distributed under the terms of the Q Public License version 1.0. Such software is herein referred to as the Software. This license covers modification and distribution of the Software, use of third-party application programs based on the Software, and development of free software which uses the Software.

Granted Rights

1. You are granted the non-exclusive rights set forth in this license provided you agree to and comply with any and all conditions in this license. Whole or partial distribution of the Software, or software items that link with the Software, in any form signifies acceptance of this license.
2. You may copy and distribute the Software in unmodified form provided that the entire package, including -but not restricted to- copyright, trademark notices and disclaimers, as released by the initial developer of the Software, is distributed.
3. You may make modifications to the Software and distribute your modifications, in a form that is separate from the Software, such as patches. The following restrictions apply to modifications;
 - a. Modifications must not alter or remove any copyright notices in the Software.
 - b. When modifications to the Software are released under this license, a non-exclusive royalty-free right is granted to the initial developer of the Software to distribute your modification in future versions of the Software provided such versions remain available under these terms in addition to any other license(s) of the initial developer.
4. You may distribute machine-executable forms of the Software or machine-executable forms of modified versions of the Software, provided that you meet these restrictions;
 - a. You must include this license document in the distribution.
 - b. You must ensure that all recipients of the machine-executable forms are also able to receive the complete machine-readable source code to the distributed Software, including all modifications, without any charge beyond the costs of data transfer, and place prominent notices in the distribution explaining this.
 - c. You must ensure that all modifications included in the machine-executable forms are available under the terms of this license.

5. You may use the original or modified versions of the Software to compile, link and run application programs legally developed by you or by others.
6. You may develop application programs, reusable components and other software items that link with the original or modified versions of the Software. These items, when distributed, are subject to the following requirements:
 - a. You must ensure that all recipients of machine-executable forms of these items are also able to receive and use the complete machine-readable source code to the items without any charge beyond the costs of data transfer.
 - b. You must explicitly license all recipients of your items to use and re-distribute original and modified versions of the items in both machine-executable and source code forms. The recipients must be able to do so without any charges whatsoever, and they must be able to re-distribute to anyone they choose.
 - c. If the items are not available to the general public, and the initial developer of the Software requests a copy of the items, then you must supply one.

Limitations of Liability

In no event shall the initial developers or copyright holders be liable for any damages whatsoever, including - but not restricted to - lost revenue or profits or other direct, indirect, special, incidental or consequential damages, even if they have been advised of the possibility of such damages, except to the extent invariable law, if any, provides otherwise. No Warranty

The Software and this license document are provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Choice of Law

These Terms and Conditions shall be governed by and construed in accordance with English law. Disputes arising in connection with these Terms and Conditions shall be subject to the exclusive jurisdiction of the English courts.

Termination

This License shall terminate automatically and You may no longer exercise any of the rights granted to You by this License as of the date You commence an action, including a cross-claim or counter-claim, against Licensor or any licensee alleging that the Original Work infringes a patent. This termination provision shall not apply for an action alleging patent infringement by combinations of the Original Work with other software or hardware.

\$Id: copyright.txt,v 1.10 2024/04/18 13:11:33 cvsuser Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Quick Start

Welcome to **GRIEF**.

GRIEF is a full-featured file and textual editor offering a wealth of facilities to programmers and non-programmers alike. It edits plain text files and has numerous options depending on the focus of your work.

Based on a long standing interface, GRIEF is an intuitive and easy editor to both novice and seasoned developers, inheriting its clean user interface from the BRIEF family of programmers editors. BRIEF was a programmer's editor for MS-DOS written by Underware, and later acquired by Borland. BRIEF was an acronym for *Basic Reconfigurable Interactive Editing Facility*.

This introduction is an outline on how to use GRIEF, based on the default keyboard layout made popular by BRIEF. The fundamental GRIEF commands you need to know are shown below.

Summary

Quick Start

Welcome to **GRIEF**.

Setup	GRIEF's local configuration may be viewed using the <code>--config</code> option.
Basics	Editing any file is as simple as running GRIEF against the file image.
Main Screen	The main features of the screen are the window arena, command line and status area.
Movement	Buffer navigation is available using a rich set of the cursor commands.
Help	Use of <code><Alt-H></code> or command <code>help</code> invokes the GRIEF help interface.
Text Editing	Inserting text is simple, just start typing; see self_insert .
Cut and Paste	If we want to copy a block of text from one part of a buffer (think of a buffer as a file) to another, or from one buffer to another, we use the <code>scrap</code> .
Search and Replace	Text are be manipulated by searching and/or translating selected text by the use of Regular expression search patterns.
Undo and Redo	The <code>undo</code> command can be used to undo any commands in the current buffer.
Command Prompt	Command line mode is entered by typing <code><F10></code> ; see execute_macro .
File and Buffer Manipulation	Files are always accessed by loading them into a buffer.
Window Manipulation	GRIEF windows can be <i>tiled</i> and used to look at more than one file at the same time, or different parts of the same file at the same time.
Record and Playback	GRIEF supports a facility to save keystroke sequences in a macro file that can be used later to save time.
Features	There are numerous additional features available, many are directly available via the <i>Feature Menus</i> <code><Alt-F></code> and/or general menu, these include.

Setup

GRIEF's local configuration may be viewed using the `--config` option.

```
$gr --config
```

```
GRIEF 3.2.2.25 compiled Apr 10 2024 00:40:33 (cm version 3.1)
```

```
PROGNAME=C:/Program Files (x86)/Grief/bin/gr.exe
MACHTYPE=Win32
GRPATH=C:/Program Files (x86)/Grief/macros;C:/Program Files (x86)/Grief/src;
GRHELP=C:/Program Files (x86)/Grief/help;
GRPROFILE=
GRLEVEL=1
GRFILE=newfile
GRFLAGS=-i60
GRBACKUP=
GRVERSIONS=
GRDICTIONARIES=
GRDICTIONARY=
GRTERM=win32
```

The more significant configuration elements which are required for correct operation are:

- | | |
|---------------------|--|
| <code>GRPATH</code> | The <code>GRPATH</code> global string is used to specify one or more directory names stating the search path which GRIEF shall utilise to locate macro objects during <code>autoload</code> and <code>require</code> operations. The initial value of <code>GRPATH</code> is either imported from the environment or if not available is derived from the location of the running application. |
| <code>GRHELP</code> | The <code>GRHELP</code> global string is used to specify the directory name stating the search path which GRIEF shall utilise to locate the help database. The initial value of <code>GRHELP</code> is either imported from the environment or if not available is derived from the location of the running application. |

GRPROFILE The *GRPROFILE* global string is used to specify an override to the standard users home directory; it is utilised by several macros to source runtime configuration details. *GRPROFILE* provides the user a means of stating an alternative location.

Basics

Editing any file is as simple as running GRIEF against the file image.

```
gr m_ruler.c
```

starts the editor, loading the specified file.

```
Grief (v2.6.1)
m_ruler.c
otherwise if the user was prompted and they aborted -l is
returned.

Macro Portability:
BRIEF limited the number of unique tab stops at 8, under
Grief this limit is 80.

Macro See Also:
inq_tabs, set_indent, distance_to_tab, distance_to_indent
*/
void
do_tabs(void) /* int ([string tabs | list tabs | int tab, ...]) */
{
    LINENO newtabs[BUFFER_NTABS + 1] = {0};
    int tabi = 0;

    tabi = ruler_import("tabs", 1, newtabs, BUFFER_NTABS);
    if (tabi >= 0 && curbp) {
        memcpy(curbp->b_tabs, (const void *)newtabs, sizeof(curbp->b_tabs));
    }
    win_modify(WFHARD);
    acc_assign_int((accint_t)tabi);
}

/* Function:          do_inq_tabs
 *      inq_tabs primitive.
 *
 * Parameters:
 *      none.
 *
 * Returns:
 */

[...]
Line: 251 Col: 1 11:39pm
```

The file can be reviewed by navigating using **Movement** commands.

At any time during the file session online **Help** is available, plus the <F10:key_map> macro shall present the current keyboard binding.

The file content may be directly manipulated using **Text Editing** commands, components can be relocated using **Cut** and **Paste** or translated **Search and Replace**. Any unwanted edits maybe corrected using **Undo** and **Redo** commands.

Once complete the edit session is complete a number of options are available.

Key	Description
Alt-W	Save file; see write_buffer .
Alt-X	Exit; see exit .
Alt-Z	Spawn a sub-shell; see shell
F10	Run additional commands against the file using the Command Prompt .

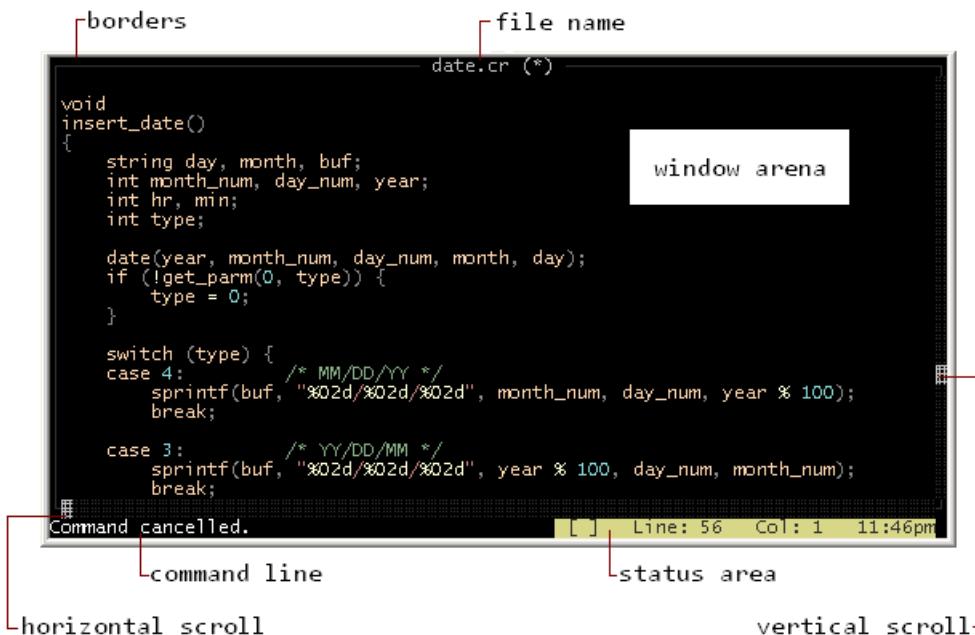
If you execute the Exit command after text has been entered in any of the open files, you will be given a number of choices:

```
1 buffer has not been saved. Exit [ynw]?
```

- w Writes the file back to the file-system and exit to the operating system.
- y The buffer is not saved, and you return to the operating system; **note** your local changes shall be lost.
- n/ESC The command is cancelled and you return to the editor.

Main Screen

The main features of the screen are the window arena, command line and status area.



Window Arena

The window arena is the area bounded by single and double lines, or borders, that displays the file content. If the file is a new image, this space shall be empty and the window is blank.

The top border of the window contains the file name associated with the visible buffer, plus an optional modification indicator. On right and bottom borders scroll bars represent the vertical and horizontal cursor position within the buffer when the buffer length or maximum line width are larger than the window arena.

Command Line

The command line, positioned below the window area, is dual purpose, used to display both messages and to prompt for information. The command line is also referred to as the **Command Prompt** when acting to request user input.

Status Area

The status area, also referred to as the echo line, displays information about the current active buffer and general editor status.

GRIEF allows the full customization of the text displayed in the status area. This is done through the `echo_line` and `set_echo_format` primitives.

The following attributes are displayed by the echo line format.

```
[x] $ Line: ### Col: ### OV RE/PA hh:mm
```

Active Character

The first elements identify the character under the cursor. Normal printable characters are enclosed within a set of square brackets, with non-printable characters represented by their hexadecimal value. When the cursor is positioned past the end of the current line, *EOL* is displayed, and when past the end of file, *EOF* is displayed.

Virtual Character

The character value is followed by the virtual character status from one of the following otherwise blank if a normal character:

- `X` Virtual space, for example logical space created as the result of tab expansion.
- `$` End of line.
- `+` Position is past the end-of-line.
- `#` Is a Unicode encoded character.
- `!` Is an illegal Unicode character code.

Buffer Coordinates

The next two elements identify the line (row) and column where the cursor is located. Unless invoked with restore enabled, when GRIEF is started the cursor is located at top of the current buffer, being line one(1) and column one(1).

Cursor Mode

On systems which have means of controlling the cursor, the current insert/overstrike mode is represented by the cursor shape; when in overstrike mode a large/block cursor is used and in insert mode a small/underline cursor is

utilised.

Otherwise on systems without cursor control a mode indicator shall be displayed, *OV* when overstrike is active otherwise blank when in insert mode.

Remember Status

When macro recording is active or paused, *RE* and *PA* respectively are displayed.

Time

Last element in the status area is the time, which is displayed in hours and minutes, with a colon as a separator using a 12 hour format.

Movement

Buffer navigation is available using a rich set of the cursor commands.

Cursor Movement

Key	Description
Right,Space	Move cursor right one position; see right .
Left,Backspace	Move cursor left one position; see left .
Down	Move cursor down one line, maintaining same column position; see down .
Up	Move cursor up one line, maintaining same column position; see up .
PgUp,Wheel-Up	Move cursor up visible display page; see page_up .
PgDn,Wheel-Up	Move cursor down visible display page; see page_down .
Home	Move cursor to beginning of current line; see beginning_of_line .
Home+Home	Move cursor to top of current window; see top_of_window
Home+Home+Home	Move cursor to beginning of the buffer; see top_of_buffer .
End	Move cursor to last character of current line; see end_of_line .
End+End	Move cursor to end of current window; see end_of_window
End+End+End	Move cursor to end of the buffer; see end_of_buffer .
Ctrl-PgUp	Move cursor to top of the buffer; see end_of_buffer .
Ctrl-PgDn	Move cursor to end of the buffer; see end_of_buffer .
Ctrl-Up	Scroll lock window movement, moving the text view up one line retaining the cursor display within the window.
Ctrl-Down	Scroll lock window movement, moving the text view down one line retaining the cursor display within the window.
Ctrl-Left	Move cursor to start of previous word.
Ctrl-Right	Move cursor to start of next word.
Ctrl-Left	Move cursor to start of previous word.
Alt-G	Goto line; see goto_line .
Ctrl-G	For supported source types list the local function definitions.

Scroll Locking

Scroll-locking is a facility for keeping the cursor in a fixed position inside a window, accessed via the <ScrollLock> key toggling alternately between enabled and disabled.

The behaviour of scroll-locking is dependent on the number of visible windows.

When a single window is active, the cursor is locked into position. Upon movement the cursor shall retain the same screen location, instead the buffer content is panned within the window where possible.

If multiple windows are active, scroll-locking allows two windows to be scrolled together. This feature permits two files to be compared with one another, without the user needing to switch between windows in order to check the two file views in sync. When enabled the user shall be prompted to select which other window the current window shall be synced with.

Scroll-lock to other window [<^V>]

Help

Use of <Alt-H> or command *help* invokes the GRIEF help interface.

Help System

Pressing <Alt-H> during a normal editing session the menu of help topics appears.

```
+-----+
| Help Menu          |
| Keyboard Summary   |
| Keyboard Mapping    |
| Macro Primitives  => |
| Explain            => |
| Features           => |
| Programmer Utils   => |
| User Guide          => |
| Programmers Guide  => |
| About              |
+-----+
```

- Use the Up and Down arrows to move to the desired topic.
- Press *Enter* to review sub-topics.

Repeat this process until the desired help screen appears. *Esc* or *Backspace* returns to the previous item or exits the help system when positioned at the top level menu.

Within the help text of the selected topic.

- The *Up* and *Down* cursor keys navigate the topic text.
- The *Left* and *Right* cursor keys plus *Tab* move between the available linked topics.
- *Enter* follows the current selected link.
- *F5* searches the content of the current topic.

A more specialised *<explain>* interface also exists, that attempts to locate the help specific to the stated topic or function.

F10:explain

Command Help

Using *<Alt-H>* whilst at the command prompt, context specific shall be retrieved resulting in help for the current command being displayed.

For example, *<Alt-G>* executes the GoTo line command. Whilst at the line prompt:

Go to Line:

The use of *<Alt-H>* shall now display help specific to the GoTo line command. Once displayed the standard help keys can be used to navigate the help content.

Text Editing

Inserting text is simple, just start typing; see [self_insert](#).

Pressing any character key inserts character into the current position. If current position is in out-of-text area, empty area is filled with space or tab characters depending on editor settings.

GRIEF is a *modeless* editor, compared to vi, and its successor Vim, which are *modal* editors. Modeless meaning that text is entered directly into the buffer as typed and commands are generally not context specific, behaving the same most of the time. Modal editing, on the other hand, means that the editor switches between the state of inserting text and taking commands.

Editing Modes

The normal editing environment is referred to as the editing mode, and all of GRIEF's commands and editing capabilities are available from it.

There are two variations on the editing mode: insert mode and overstrike mode. In insert mode, typed text is inserted at the cursor. In overstrike, existing text is overwritten as you type. The insert mode is reflected in state of the cursor plus its status shall be visible within the status area.

Deletion and relocation of text is possible using the scrap buffer, see [Cut and Paste](#).

Edit Commands

Key	Description
Backspace	Delete the last character typed; see backspace .
Delete	Delete the character under the cursor; see delete_char .
Alt-D	Delete the current line; see delete_line .

Key	Description
Alt-K	Delete characters to right of cursor on current line; see delete_to_eol .
Alt-I	Toggle insert and overwrite modes; see insert_mode .
Ctrl-K	Delete word to left of the cursor
Ctrl-L	Delete word to right of the cursor
Ctrl-F	Format block.
F10:set spell	Enable spell checking.
F10:center	Center the current line.

Cut and Paste

If we want to copy a block of text from one part of a buffer (think of a buffer as a file) to another, or from one buffer to another, we use the *scrap*. The scrap is a temporary storage area for text which has been cut or copied from a buffer and can then be inserted as many times as necessary to some other buffer. Cutting text is different from deleting text. Cutting text deletes the original text but saves it so it can be inserted elsewhere. The deleted text is gone.

In order for a piece of text to be cut/copied and then pasted it must first be highlighted. To highlight a region of text, the user first drops an anchor. As the cursor is moved away from the anchor, the text between where the anchor was dropped and the current cursor position is highlighted, showing the text which can be cut or copied.

There are three types of regions, block, column and line. Block and line type is used to cut/copy and paste whole lines. A column type is used to cut/copy and paste columns of text.

Search and Replace Commands

Key	Description
Alt-M	Drops a normal block. Text falling within the current cursor position and from where the original anchor was dropped shall be highlighted; see mark mode 1 for details.
Alt-C	Drops a column marker. Text falling within a rectangular region from where the anchor was dropped to the current cursor will be highlighted, see mark mode 2 for details.
Alt-L	Drop a line marker. Text falling within a rectangular region from where the anchor was dropped to the current cursor will be highlighted; see mark mode 3 for details.
Alt-A	Drops an inclusive block, similar to a normal block; see mark mode 4 for details.
KP-Plus	If no region is currently highlighted, then the current line is copied to the scrap buffer. If there is a highlighted region, then that region is copied to the scrap without being deleted; see copy .
KP-Minus	If no region is currently highlighted, then the current line is cut to the scrap buffer. If there is a highlighted region, then that region is copied to the scrap and deleted; see cut .
Ins	Paste the contents of the scrap buffer into the current buffer at the current cursor position. For line-types regions, the lines are inserted before the current line rather than inserting where the cursor is; see paste .
Ctrl-O	Search options.

Search Attributes

Several options are available that control search behaviour, see `<search_options>`. Most options are both mapped to keys in addition to being accessible on the `<options>` menus (Ctrl-O).

Regular Expression

Regular expression matching indicates whether certain characters have special meaning when performing searches. When regular expressions are disabled search patterns are treated as string literals otherwise patterns are interpreted against the current syntax mode.

Case Sensitivity

Case sensitivity controls whether alphabetic characters should be compared as equivalent if they only differ in their case, for example whether A should match both a and A.

Block Selection

Block searching controls the action to be taken when a search is requested and the current buffer has a highlighted section.

If this option is *off* then the search is performed on the entire buffer, ignored the marked region.

If this option is *on* then the search begins at the start of the marked region and continues until a match is found or the end of the region is reached.

Syntax mode

Regular expression syntax mode indicates which expression syntax patterns are interpreted as.

Naming Scrap Commands

Similar to the scrap buffer primitives [copy](#), [cut](#) and [paste](#), exists the ability to act upon a named scrape buffer. A named scrap behaves the same as the normal scrap, yet allows the user to keep separate copies of data in these temporary buffers.

The named scrap commands shall prompt the user for the name of buffer to be used. Using <Tab> at these prompts shall present a popup showing the currently defined scrap names, allowing for both buffer management and selection operations.

Key	Description
Ctrl-KP-plus	If no region is currently highlighted, then the current line is copied to the <i>named</i> scrap buffer. If there is a highlighted region, then that region is copied to the <i>named</i> scrap without being deleted; see <code><copy_named_scrap></code> and copy
Ctrl-KP-Minus	If no region is currently highlighted, then the current line is cut to the <i>named</i> scrap buffer. If there is a highlighted region, then that region is copied to the <i>named</i> scrap and deleted; see <code><cut_named_scrap></code> and cut .
Ctrl-Ins	Paste the contents of the <i>named</i> scrap buffer into the current buffer at the current cursor position; see <code><paste_named_scrap></code> and paste .

Search and Replace

Text are be manipulated by searching and/or translating selected text by the use of Regular expression search patterns.

Search and Replace Commands

Key	Description
F5, Alt-S	Search in a forward direction. The user is prompted for the search item; see search_fwd .
Alt-F5, Alt-Y	Search in a backwards direction. The user is prompted for the search item; see search_back .
KP-5, Shift-F5	Search for the next occurrence of an item in either the forward or backwards direction, depending on the last search.
F6, Alt-T	Performs a replace in the forwards direction. Prompts the user for a item to search for (translate) and an item to replace it with. For each matched occurrence of the item, the user is prompted for whether to change or not; see translate
Alt-F6	Performs a replace in the backwards direction.
Shift-F6	Repeats last replace in the same direction.
Ctrl-F5	Toggles the case sensitivity. The default is for case sensitivity to be turned on. When turned off, lower case characters match against upper case and vice versa; see search_case .
Alt-O	Options menu, permits access to the global word processing options.
F10:bufinfo	Buffer information dialog, configures the buffer options.

Search Expressions

Search patterns are expressed in terms of a Regular expression. Regular expressions are special characters in search or translate strings that let you specify character patterns to match, instead of just sequences of literal characters.

Regular expression characters are similar to shell wild-cards, yet are far more powerful. There are several supported expression syntaxes, with the original BRIEF syntax being the default; see [re_syntax](#).

These are the BRIEF regular expressions.

Expression	Matches
?	Any character except a newline.
*	Zero or more characters (except newlines).
\t	Tab character.
\n	Newline character.
\c	Position cursor after matching.
\\	Literal backslash.
< or %	Beginning of line.
> or \$	End of line.
@	Zero or more of last expression.
+	One or more of last expression.
	Either last or next expression.
{}	Define a group of expressions.

Expression	Matches
[]	Any one of the characters inside [].
[~]	Any character except those in [~].
[a-z]	Any character between a and z, inclusive.

Note: Under the BRIEF syntax mode the *, @, and + expressions will always match as few of the expression in question as possible; also known as non-greedy matching. UNIX-style greedy matching modifiers are also available for macro use.

Within replacement text the following are allowed.

Expression	Inserts
\t	Tab character.
\n	Newline character.
\<n>	Substitute text matched by the associated nth group, where (0 <= n <= 9).

Regular Expression Examples

Pattern	Result
the	Find the next occurrence of "the".
{him} {her}	Find the next occurrence of "him" or "her".
<alone> or %alone\$	Finds next occurrence where "alone" is alone on a line.
stuff*between	Find next occurrence of "stuff" followed by "between" on the same line.
th[eo]se	Find next occurrence of "these" or "those".
[A-Z][a-z]@;	Find next capitalized word delimited by a trailing semicolon.
[0-9]+	Find one or more consecutive digits.
[~ \t\n]	Find any character but a space, tab, or newline.

The Regular Expression Toggle command turns expressions on or off.

Case sensitivity for searches and translates can be turned on and off with the Case Sensitivity Toggle. When case sensitivity is off, "grief" will match "GRIEF", ; when case sensitivity is on, it will only match "grief".

Undo and Redo

The **undo** command can be used to undo any commands in the current buffer.

The undo facility can be compared to an edit audit trail, which tracks all modifications to the buffer, including edits, marked regions and cursor movement. Each buffer records changes within the scope of the current editor session independent of other buffers, with an infinite level of undo information for each buffer.

The undo command reverses recent changes in the buffer's text, and the redo command always applies to the current buffer. Commands can be undone sequentially all the way back to the point where the buffer was opened or created. If you undo too much you can use **redo** to cancel the last undo.

Undo Commands

Key	Description
Alt-U,KP-star	Undoes previously executed command; see undo .
Ctrl-U	Redoes the previous undo; see redo .

Command Prompt

Command line mode is entered by typing <F10>; see [execute_macro](#). The prompt allows access to commands which are not assigned to specific keystrokes to be executed by typing <F10> and entering the full command name for example <explain>.

Once typed the *echo line* along the bottom of the console shows the "Command:" prompt. The command line is case-sensitive and all commands generally need to be entered in lower case to be recognised.

Command:

The command prompt shall also become active when user input is required by an interactive command, for example Goto Line.

Go to Line:

The command line has full editing features. The *Left* and *Right* cursor keys allows navigation within the prompt, permitting text to be inserted and deleted. Whilst at the prompt context specific help is available using <Alt-H> display help related to the current command.

Command History

Up to the last sixteen responses are saved for each prompt displayed; single character responses are ignored and not saved. Use the cursor *Up* and *Down* keys to cycle through the list of remembered responses. The *Esc*, which cancels the command is not stored.

You can also recall the last response entered at any prompt using <Alt-L>. This is useful when you find you entered the correct response to the wrong prompt. Press <Esc> to cancel the first command, issue the new command, and press <Alt-L> to recall the last response.

Command Completion

Certain commands for example Edit File can take advantage of the file name completion feature. Type the first letters of the name and press <Tab>. If only one file in the directory matches, it is completed. If more than one file is found, a selection dialog is displayed allowing selection of the desired file.

Command Line Commands

During any command line interaction the available key binding are the following.

Key	Description
Enter	Execute the current command.
Esc	Cancel the input.
Home, End	Moves the cursor to the head or tail of the text.
Left, Right	Moves the cursor within the input text.
Del	Delete the current character.
Backspace	Delete previous character.
Ctrl-Left	Move left one word at a time.
Ctrl-Right	Move right one word at a time.
Up, Down	Navigate history; up/down scrolls through previous responses to the prompt.
Ins	Paste scape buffer.
Tab	File completion.
Alt-B	Current buffer name.
Alt-F	Current path and file name.
Alt-L	Recalls the last response that was typed by the user.
Alt-P	Paste text under cursor.
Alt-W	History buffer.
Alt-I	Toggle between insert and over-type mode.
Alt-Q	Quote next input character. This allows user to type things like <Tab> and <Esc> as part of the input text.

File and Buffer Manipulation

Files are always accessed by loading them into a buffer. GRIEF is able to keep multiple files in memory at once, which allows the user to move among them easily. Once a user is in GRIEF they can call up as many files as they need to use by loading them into a buffer within GRIEF. Pop-up menus are often used when moving among files. How to choose options on the pop-up menu is self-evident. To exit a pop-up menu use the <Esc> key

Buffer Manipulation Commands

Key	Description
Alt-E	File and Buffer Manipulation in GRIEF Call up a file to be edited. The user is prompted for the name of the file. Hitting <Tab> pops up a menu of file-names the user can select from.
Alt-B	Pops up a window containing a list of the files currently in memory.
Alt-N	Next buffer.
Alt-P	Previous buffer.
Ctrl-AE	Reload the current buffer; This is useful if the file was changed by another application and you want to update it in the editor, or when you want to discard all changes done since last save.
Alt-1 .. 9	Drop a bookmark; see drop_bookmark
Alt-J	Goto a bookmark.

Window Manipulation

GRIEF windows can be *tiled* and used to look at more than one file at the same time, or different parts of the same file at the same time.

By tiling, we mean the window can be split horizontally or vertically. This can be done as many times as you like, i.e. you can split your window into as many sections as you like. Different buffers (files) can be viewed in different sections of the tiled window, or even different parts of the same buffer.

Each section of the tiled window can be treated as an individual GRIEF session. Any changes to a buffer made in one part of a tiled window are displayed in the other parts of the tiled window displaying the same part of the same buffer.

Window Manipulation Commands

Key	Description
F1	Select a different section of a tiled window. User is prompted to point to the section desired, using one of the four arrow keys; see change_window .
F2	Move the boundary between two windows on the screen. User is prompted to point the boundary which is to be moved, and then use the arrow keys to move the boundary. The screen is redrawn as the boundary is moved; see move_edge .
F3	Split the current window into two equal sizes, either horizontally or vertically; see create_edge .
F4	Delete a boundary between two windows and merge them together; see delete_edge .
Ctrl-Z	Subwindow zoom toggle: makes a forward zoom on the current subwindow. This means that the current subwindow will occupy all possible place in the total window. Use Ctrl-Z again to unzoom, i.e. to see again all sub-windows.
F10:wininfo	Window information dialog.

Tiled Windows

Tiled windows are created by splitting the current window in half either horizontally or vertically, by default using <F3>, providing two views of the current buffer.

On request the user is prompted and the current window is split based on the direction of the selected arrow key. On completion the newly created window is made current, with the cursor located at the same coordinates as the parent window.

Select new side [<^v>]

To create a new window, press one of these arrow keys.

Left Splits the current window, creating a vertical window to the left.

Right Splits the current window, creating a vertical window to the right.

```

 builtin.c
 accent_t ival;
 int ret, len = 0;

 ret = str_numparse(cp, &fv
 switch (ret) {
 case NUMPARSE_INTEGER:
 case NUMPARSE_FLOAT:
 cp += len;
 if (0 == *cp || isspace
 if (ret == NUMPARSE_
 lp = atom_push_
 } else {
 lp = atom_push_
 }
 break;
 }
 cp -= len;
 /*FALLTHRU*/
 default:
 goto string;
}

```



```

 builtin.c
 accent_t ival;
 int ret, len = 0;

 ret = str_numparse(cp, &fv
 switch (ret) {
 case NUMPARSE_INTEGER:
 case NUMPARSE_FLOAT:
 cp += len;
 if (0 == *cp || isspace
 if (ret == NUMPARSE_
 lp = atom_push_
 } else {
 lp = atom_push_
 }
 break;
 }
 cp -= len;
 /*FALLTHRU*/
 default:
 goto string;
}

```

Up Splits the current window, creating a horizontal window above.

Down Splits the current window, creating a horizontal window below

```

accint_t ival;
int ret, len = 0;

ret = str_numparse(cp, &fval, &ival, &len);
switch (ret) {
case NUMPARSE_INTEGER: /* integer-constant */
case NUMPARSE_FLOAT: /* float-constant */
    cp += len;
    if (0 == *cp || isspace(*cp)) {
        if (ret == NUMPARSE_INTEGER) {
            lp = atom_push_int(lp, ival);
        }
    }
}
accint_t ival;
int ret, len = 0;

ret = str_numparse(cp, &fval, &ival, &len);
switch (ret) {
case NUMPARSE_INTEGER: /* integer-constant */
case NUMPARSE_FLOAT: /* float-constant */
    cp += len;
    if (0 == *cp || isspace(*cp)) {
        if (ret == NUMPARSE_INTEGER) {
            lp = atom_push_int(lp, ival);
        }
    }
}

```

Line: 122 Col: 1 11:01pm

Once created tiled windows can be resized and/or deleted using similar commands.

The minimum size of a window with borders on is two(2) lines deep by fourteen(14) columns wide. When a split or resize request would result in the lower bounds being violated the follow shall be echoed and the request ignored.

Window would be too small.

Record and Playback

GRIEF supports a facility to save keystroke sequences in a macro file that can be used later to save time. This ability allows repetitively executed commands to be recorded and replayed as a single command; referred to as a keyboard macro.

Keyboard macros are stored in memory, plus these may be saved to file and reloaded, permitting a user to develop a private collection of simple macros. Keystroke macros are saved as normal ASCII text and can be edited just like any other file. You can create any number of remembered sequences during an editing session. However, only one sequence is saved at a time.

The associated commands are summarised below.

Key	Description
F7	Remember; see remember .
Shift-F7	Pause recording during a remember operation; see pause .
F8	Playback; see playback .
Alt-F7	Load a keystroke macro.
F10:keylib	Keyboard macro management.

Features

There are numerous additional features available, many are directly available via the *Feature Menus* <Alt-F> and/or general menu, these include.

Key	Description
Alt-F	Features menu.
Alt-G	Function list.
Alt-V	Display build information at the prompt; see version
Ctrl-O	Options menu.
Ctrl-A	Extra menu.
F10:coloriser	Load a new content coloriser.
F10:colorscheme	View coloriser definition.
F10:about	About dialog.
F12	Menu; enable the pull-down command menu. Can also be access using the <i>menu</i> command and disabled running the <i>menuoff</i> command.

\$Id: quickstart.txt,v 1.4 2014/11/27 18:08:41 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Macro Tutorial

An overview and quick tutorial of GRIEF's macro system.

The GRIEF macro language enables the customization of GRIEFEdition, and enables the creation of new functionality. Many of the actions performed using GRIEF are performed using macros. GRIEF functions are mapped to keys, dialogs and menus, and perform the action behind an event.

The language API is extensive, covering many actions normally performed in a code editor, including navigation and buffer modification plus POSIX inspired system calls. Much of the code in high level functionality of GRIEF is written in the macro language and this source is provided when GRIEF is installed. This enables the customization of the product, and the use of the macro source as an example for writing new macros. After installation, the compiled macros and the associated source are located in the *macros* and *src* subdirectories of the installation directory.

Summary

Macro Tutorial

An overview and quick tutorial of GRIEF's macro system.

[GRIEF Macros](#)

GRIEF follows a C-style development model with the distinction that macros can be loaded dynamically.

[Compilation Model](#)

We will briefly highlight key features of the GRIEF Compilation model

[A Quick Macro Tutorial](#)

Most **GRIEF** macros consist of function definitions and data structures.

[Compiler Usage](#)

The macro compiler grunchn command line usage.

[Compatibility](#)

Compatibility with the original Borland Brief and the current CRISP™ Edit implementation; when known at the time of writing, details of possible porting issues have been documented.

GRIEF Macros

GRIEF follows a C-style development model with the distinction that macros can be loaded dynamically.

Macros are stored in files ending in the **.cr** extension. The GRIEF macro translator compiles these files to byte code which is saved in a corresponding file with the **.cm** extension.

GRIEF uses the a C style preprocessor, with all primary definitions available using,

```
1 #include <grief.h>
2 //or
3 #include <crisp.h>           // Crisp compatibility
```

Functions

Basic structure of a function or macro.

```
1 int
2 function (parameter1, parameter2)
3 {
4 }
```

Firstly the functions return type is given, with the function name followed by the parameter declaration, if any are required. Parameters are surrounded by round brackets (parentheses) and separated by commas.

The body of the function is then encased by curly braces.

Compilation Model

We will briefly highlight key features of the GRIEF Compilation model

Preprocessor

The Preprocessor accepts source code as input and is responsible for

- Comment filtering
- Interpreting special preprocessor directives denoted by a leading #.

Preprocessor Example

`#include` includes contents of a named file. Files usually called header files.

```
#include <grief.h>

#define defines a symbolic name or constant. Macro substitution.

#define MAX_ARRAY_SIZE 100
```

Compiler

The **GRIEF** Macro compiler translates source to a type of byte-code/assembly code.

The source code is received from the preprocessor.

Loader

Upon macro execution, each macro goes through a number steps

Validation Checks permissions, memory requirements etc.

Copy Copies the macro image from the disk into main memory;

Relocation Associates the macro image with the memory into which it is loaded, by adjusting pointer references in the macro to compensate for variations in the objects base addresses.

Initialise Execute the macro initialisation and main entry point.

Note that the initialisation of any given macro may in turn load additional resources.

A macro that is loaded may itself contain components that have not really been loaded themselves, as such one macro may trigger the loading of one or more additional macro until all dependences are met.

This process is controlled using **GRIEF's** autoload primitives (See: [autoload](#)).

Development Tools

GRIEF is packaged with a number of tools, including the *processor* and *compiler* which aid with the development of macros; these shall be demo'd in the following tutorial.

- grunch - GRIEF Macro Compiler.
- grcpp - Macro language preprocessor, which is C/C++ style preprocessor. It is usually only utilised by the *grunch* compiler, yet can be invoked manually during code debugging.
- gm - Macro byte-code compiler/decompiler, which deals with .m source and .cm objects.

A Quick Macro Tutorial

Most **GRIEF** macros consist of function definitions and data structures. Here is a simple macro that defines a single function, called *main*.

```
1 // Source: helloworld.cr
2 //
3 //      GRIEF Macro Tutorial, working example.
4 //
5 #include <grief.h>           // public header
6
7 void    // user macro
8 helloworld()
9 {
10     message("Prompt: Hello, world!");
11 }
12
13 void    // macro initialisation
14 main()
15 {
16     message("Prompt: Welcome");
17     sleep(2);
18 }
```

Opening the macro source are a number of comments line, which serve as documentation to the macros purpose; like with all comments the content has no effect on the definition nor execution of the macro.

Following one is the first statement, being the `#include` directive. In this case the system include file *grief.h* is referenced; which defines all the global constants, variables and function which are accessible to macros. Generally all macros shall be started in this way.

Next is the macro entry function. All functions must have a return value; that is, the value that they return when they finish execution. *hello* has a return value type of `void`. Other types include integers (`int`), floating point numbers (`float`) and lists (`list`). This function declaration information must precede each function definition.

Immediately following the function declaration is the function's name (in this case, hello).

Next, in parentheses, are any arguments (or inputs) to the function. *helloworld* has none, but an empty set of parentheses is still required. After the function arguments is an open curly-brace '{'. This signifies the start of the actual function code. Curly-braces signify program blocks, or chunks of code.

Next comes a series of **GRIEF** statements. Statements demand that some action be taken. Our demonstration program has a single statement, a message (formatted printf). This shall echo the message "Hello, world!" at the command prompt.

The message statement ends with a semicolon (";"). All statements must be ended by a semicolon. The main function is ended by the close curly-brace }.

Compiling and Loading Macros

There are several methods which build macros can be compiled and then executed, the primary being via the command line *Macro Compiler grunch*.

The *grunch* compiler translates the module specified into a binary encoded byte-code object.

```
$grunch helloworld.cr
```

The result should be the byte-code object named by default as *helloworld.cm*, an alternative name maybe given using the -o option.

GRIEF can now be started and using <F9> **load macro** and <F10> **execute command**, the macro object can be loaded and executed.

Upon the macro loading using F9, the macro object shall be loaded and initialised which in turn executes the main() function (if any) within the associated macro, the result shall be as such.

```
Prompt: Welcome
```

Once loading the command prompt shall return, from this point the macro can be executed using F10.

```
Prompt: Hello world!
```

Macro Objects

Macro objects represented by files using the .cm extension are a compact form of a Lisp-like language. The original Brief™ used a similar Lisp-like language using the .m extension.

This lisp dialect should be considered as the *assembly language* or *byte-code* of GRIEF, and the grunch compiler can be made to generate the equivalent output for diagnostics of the underlying compiler logic.

It is feasible possible to directly utilise the language to write macros (See: <cm>), however the higher level C-style language provides much of the same functionality with the added benefits of a more structured and portable interface. Furthermore the syntax and features of the **GRIEF** byte-code may change without notice, as such its direct use is not advised.

```
GRIEF macro decompiler.  
gm 3.2.0 compiled Jul 17 2014 03:16:30 (engine version 20)
```

Usage: gm [options] <file> ...

options:	
-a	Print atom percentages.
-l	List macro expansions.
-L	Print detailed disassembly info.
-q	Quiet error messages.
-s	Print size of .cm file only.
-o file	Name of compiled output file.
-I path	Include path.
-h	Command line help.

Using the *gm* tool the internals of the macros can be studied, as following is a sample of the output from the *helloworld* object from above.

```
$ gm -l helloworld.cm
:
*** Macro 0:

Atom 0000: F_ID      macro
Atom 0003: F_STR    "helloworld"
Atom 0008: F_LIST   --> 14
Atom 000b: F_ID      message
Atom 000e: F_LIT    "Hello, world!"
Atom 0013: F_HALT
Atom 0014: F_HALT

*** Macro 1:

Atom 0015: F_ID      macro
Atom 0018: F_INT     1
Atom 001d: F_STR    "_init"
Atom 0022: F_LIST   --> 45
Atom 0025: F_LIST   --> 31
Atom 0028: F_ID      message
Atom 002b: F_LIT    "Welcome"
Atom 0030: F_HALT
Atom 0031: F_LIST   --> 38
Atom 0034: F_ID      beep
Atom 0037: F_HALT
Atom 0038: F_LIST   --> 44
Atom 003b: F_ID      sleep
Atom 003e: F_INT     2
Atom 0043: F_HALT
Atom 0044: F_HALT
Atom 0045: F_HALT
Atom 0046: F_END    *** End ***

:
```

Compiler Usage

The macro compiler grunch command line usage.

```

GRIEF Macro Compiler.
grunch 3.2.0 compiled Jul 17 2014 03:16:33

Usage: grunch [-acfgmnpqSWw] [-Dvar[=value]] [-Uvar] [-Ipath]
              [-o output_file] [+struct] file-name ...

Options:
  -a          Create core file on exit (for debugging).
  -c          Leave temporary files (.m etc).
  -Dvar       Define var as in #define.
  -d[d]       Enable internal debugging features.
  -f          Flush output for debugging.
  -Ipath      Add 'path' to the #include search path.
  -g          Compile with debug on.
  -m[m]       Compile only if out of date (make option)
  -n          Don't do anything but tell us what you would do.
  -o file     Name of compiled output file.
  -e file     Error output file.
  -A file     Autoload macro file.
  -p cpp      Specify name of C pre-processor to use.
  -q          Dont print filenames during compilation.
  -S          Dump symbol table.
  -Uvar       Undefine var as in #undef.
  -UUNUSED    Remove internal UNUSED definition.
  -V          Print version of grunch.
  -w          Enable warnings.
  -v          Verbose.
  -warn_errors Treat warnings as errors.
  -wproto     Disable prototype checks.
  -stages     Watch compiler passes.
  +struct    Pretty print structure offsets for easy parsing.
  -h          Command line help.

Variables:
  GRCPP=<cpp>  Preprocessor override (grcpp).
  GRARG=<args>   Preprocessor argument override.
  GRPATH=<path>  GRIEF include path.

```

Compatibility

Compatibility with the original Borland Brief and the current CRISP™ Edit implementation; when known at the time of writing, details of possible porting issues have been documented.

GRIEF and CRISP™ have both evolved independently and at this time no explicit testing has been performed to verify the compatibility between either the legacy Borland™ Brief macro language or the current commercial CRISP™ Edit macro language.

Future effort may be directed towards a formal statement of compatibility if the need is warranted, possibility including the development of a compatibility layer, yet I would guess except in cases where the more enhanced features are leveraged most macros should run directly with only minor changes.

Any assistance and effort you can contribute towards confirmed the current level of cross compatibility would be greatly appreciated and excepted.

\$Id: tutorial.txt,v 1.7 2014/11/27 18:08:41 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Language Specification

The syntax utilised by the **Grief** macro language allows one to create macros using commands written in a C based programming language.

In addition, **Grief** contains some constructs from other environments including C++.

Summary

Language Specification

The syntax utilised by the **Grief** macro language allows one to create macros using commands written in a C based programming language.

Lexical elements

Source is broken down into a number of tokens classes, identifiers, expression operators, and other separators.

Notation

The syntax is specified using Backus-Naur Form (BNF).

Source Code

Macro source code is ASCII text encoded; note UTF-8 is a planned feature.

Comments

Comments serve as a form of in-code documentation.

Statements

Grief statements are constructed in the same manner as the statements in the C language.

Braces

Curly braces are used to define the beginning and end of functions definitions, to group multiple statements together within a single function, for example.

Line Numbers

When reporting errors and warnings, a **Grief** compiler uses a source-code location that includes a file name and line number.

Tokens

Tokens form the vocabulary of the **Grief** language.

Identifiers

An identifier is used to give a name to an object.

Keywords

A keyword is a reserved identifier used by the language to describe a special feature.

Punctuators

The punctuation and special characters in the C character set have various uses, from organizing program text to defining the tasks that the compiler or the compiled program carries out.

Literals

A literal is the source code representation of a value of a primitive type, the String type , or the list type.

Integer Literals

The most commonly used type is the integer.

String Literals

A *string literal* is a sequence of characters from the source character set enclosed in double quotation marks ("").

Character Literals

A *character literal* is a single character from the source character set enclosed in single quotation marks ('').

Escape Sequences

In order to encode the character codes of non-printing characters, Unicode character codes within non-Unicode source, allow references to platform specific control characters and the allow literal delimiters within literals, these special symbols need to be denoted with an escape mechanism.

Floating Point Literals

A floating-point number is a number which may contain a decimal point and digits following the decimal point.

Types

Each object, reference, and function in **Grief** is associated with a type, which is defined at the point of declaration and cannot change.

Integer Types

An integer is just a number.

Float Types

Floating point number, which internal is represented using a double-precision float.

String Types

A string is a sequence of printable characters.

Character Types

There is no specific character type, instead an <integer type> can be used to handle character data.

List Types

A list is a collection of objects of any type.

Polymorphic Types

A polymorphic type is one in which the type of the variable stored can be changed.

Enumerated Types

At times it is desirable to have a list of constant values representing different items, and the exact values are not relevant nor can code easy to neither read nor understand.

Expressions

An expression is a sequence of operators and operands that describes how to,

Operators

An operator is used to describe an operation applied to one or several objects.

Operator Precedence

The following is a table that lists the precedence and associativity of all the operators in the **Grief** language.

Array Subscripting

Array Subscripting Operations are executed by the following operators.

Function Calls

Function calls executed by the following operators

Increment and Decrement Operators

Increment and Decrement operations are executed by the following operators.

Unary Arithmetic Operators

Unary Arithmetic operations are executed by the following operators.

Arithmetic Operators

Arithmetic operations are executed by the following operators.

Bitwise Shift Operators

Bitwise Shift operations are executed by the following operators

Relational Operators

Relational operations are executed by the following operators

Equality Operators.

Equality operations are executed by the following operators

Bitwise Logical Operators.

Bitwise Logical operations are executed by the following operators

Logical Operators	<i>Logical</i> operations are executed by the following operators
Conditional Operator	Inline <i>Conditional</i> operation are executed by the following operators
Assignment Operators	<i>Assignment</i> operations are executed by the following operators
Comma Operator	The <i>Comma</i> Operator.
Declarations	Variables and functions are declared in the same way in Grief as they are defined in C.
Storage Class	A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.
Function Declarations	A function is a group of statements that together perform a task.
Parameter List	Parameters are optional is that the list can be given as either <i>void</i> or empty, meaning the function takes no parameters, or a comma-separated list of declarations of the objects, including both type and parameter name (identifier).
Lazy Evaluation	To fully understand the Grief calling convention, examples of parameter implementation are required.
Function Prototypes	Function prototypes provide the compiler with type information about a function without providing any code.
Scope	Variables and functions can be used only in certain regions of a program.
Scope Rules	The four kinds of scope.
Modules	Grief provides a mechanism for alternative namespaces, being an abstract container providing context for the items, to protect modules from accessing on each other's variables.
Statements	A statement describes what actions are to be performed.
Compound Statements	A compound statement is a set of statements grouped together inside braces.
Expression Statement	A statement that is an expression is evaluated as a void expression for its side effects, such as the assigning of a value with the assignment operator.
Selection Statements	A selection statement evaluates an expression, called the controlling expression, then based on the result selects from a set of statements are then executed.
Iteration Statements	Iteration statements control looping.
Jump Statements	A jump statement causes execution to continue at a specific place in a program, without executing any other intervening statements.
if statement	if-else selection clause.
switch statement	switch selection clause.
while statement	while iteration clause.
do-while statement	do-while iteration clause.
for statement	for iteration clause.
continue statement	continue clause.
break statement	break clause.
return statement	return clause.
returns statement	returns clause.

Lexical elements

Source is broken down into a number of tokens classes, identifiers, expression operators, and other separators.

In general blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens. At least one of these characters is required to separate otherwise adjacent identifiers, constants, and certain operator pairs.

If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

Notation

The syntax is specified using Backus-Naur Form (BNF).

Each BNF is a set of derivation rules, written as

BNF specification

```
<symbol>:
    <expression>      // comment
    | <expression2>    // second choice or branch.
    ;
                // terminator, optional.
```

where *<symbol>* is a non-terminal, and the *<expression>* consists of one or more sequences of symbols; more sequences are separated by the vertical bar, |, indicating a choice, the whole being a possible substitution for the symbol on the left.

The :: (sometime only :) means that the symbol on the left must be replaced with the expression on the right, with ; terminating the current sequence.

Comments within the expressions are started with the sequence "://" and stop at the end of the line; and do not form

part of the expression being described.

White space, formed from spaces, horizontal tabs, carriage returns, and newlines, should be ignored except when stated explicitly as either a literal or terminator, for example `` or new-line.

Syntax	Description
: (double colon)	Definition
(vertical bar)	or
{...}*	0 or more
{...}+	1 or more
[....]	optional
(...)	selection
...	literal character or string
...	terminal or non-terminal
//	end-of-line comment

Backus Naur Form

Within the right expressions a grouped set of tokens enclosed in round brackets, indicating one or alternative selections, with only one of the available at any time, for example.

```
a_or_b_plus_word:: ('a', 'b') word ;
```

for expressions which only represent a single token the bracket shall be omitted, as such.

```
letter::  
    'a' .. 'f', 'A' .. 'F'  
;
```

Optional items enclosed in square brackets, for example

```
optionalword:: [word] ;
```

Items repeating 0 or more times are enclosed in curly brackets, for example

```
one_or_more_letters::  
    letter {letter}  
;
```

and the annotated short-hand form

```
one_or_more_letters::  
    {letter}+  
;  
none_or_more_letters::  
    {letter}*  
;
```

standard patterns only using the bases operators : and |. The follow are there longer forms.

```
optionalword::  
    word  
    |           // empty choice.  
    ;
```

```

word:::
    letter_list
;

letter_list:::
    letter      // recursive list definition.
    | letter_list letter
;

```

```

a_or_b_plus_wor:::
    a_or_b word
;

a_or_b:::
    'a'
    | 'b'
;

```

Source Code

Macro source code is ASCII text encoded; note UTF-8 is a planned feature.

Each code point is distinct; for instance, upper and lower case letters are different characters.

Like C, the **Grief** Macro language is case sensitive.

Comments

Comments serve as a form of in-code documentation. When inserted into the source code, they are effectively ignored by the preprocessor and compiler; they are solely intended to be used as notes by the developers that maintain the source code.

There are two forms of comments, "Multi-line" and "Single-Line" comments.

Syntax

```

/* comment */          (1)
// comment\n          (2)

```

Multi-line comments

General comments start with the character sequence "/*" and continue through the character sequence "*/". A general comment containing one or more newlines acts like a newline, otherwise it acts like a space.

These are often known as "C-style" or "multi-line" comments.

Single-line comments

The second form are line comments, which start with the character sequence "//" and stop at the end of the line, however, multiple line comments can be placed together to form multi-line comments. A line comment acts like a newline.

These are often known as "C++-style" or "single-line" comments.

Comments are recognized anywhere in a program, except inside a character constant or strings.

Grammar rules employed imply the following

- Comments do not nest, that is a comment opens /* is matched with the next */ encountered; within the comment body any additional /* or // tokens are simply treated as part of the comment text and have no special meaning.
- /*/* and */ have no special meaning inside // comments.
- // has no special meaning in either single-line or multi-line comments.

Examples

The following code fragment, using a mix of block and line comments;

```
/* Insert the list of string */
for (i = 0; i < scount; ++i) {           // loop through list
    insert( slist[s] );                  /* close the file */
}
```

is equivalent to,

```
for (s = 0; s < scount; ++s) {
    insert( slist[s] );
}
```

Comments have several uses, which includes the documentation of your source. Another use can be to temporarily remove a section of code during testing or debugging of macros, as example:

```
/* -- disable
for( i = 0; i < fcount; ++i) {
    fclose( flist[i] );
}
*/
```

Statements

Grief statements are constructed in the same manner as the statements in the C language.

The semicolon (;) represents a statement terminator. The semicolon should be used at the end of all complete statements.

Syntax

```
<expression>;
```

The expression shall be none or more statements, constructed of identifiers, literals, keywords and operators.

Note that the expression maybe empty, also referred as a null expression, which can be used as a no-op or no-operation place holder, as such;

```
;
```

Braces

Curly braces are used to define the beginning and end of functions definitions, to group multiple statements together within a single function, for example.

Block statement

```
{
    <expression>;
    <expression>;
}
```

Secondary braces are used to delimit data contained within initialisation lists.

Line Numbers

When reporting errors and warnings, a **Grief** compiler uses a source-code location that includes a file name and line number. **Grief** numbers the lines in a compilation unit starting from one. The end of a line is marked by a newline character.

To facilitate interoperability with other tools, a line directive can be used to associate source code lines to a location in a different file.

Line directive form

```
# number "file"
```

A line directive must be on a line of its own and must start with the # character; number is a decimal number in the file name *file*. One or more space or tab characters must be used to delimit the three tokens of a line directive.

The syntax of the line directive is the one used by the C preprocessor `cpp`. The line directive associates the following line in the source code with line *line* within the specified source *file*.

Line numbering continues from there on linearly and thus all subsequent lines are considered to be from *file*.

Tokens

Tokens form the vocabulary of the **Grief** language.

There are several classes

- Identifiers
- Keywords
- Literals
- Operators
- Punctuators

Tokens are only processes as full words. As the macro source is scanned, tokens are extracted in such a way that the longest possible token from the character sequence is selected. For example, `extern` would be parsed as a single identifier rather than as the keyword `extern` followed by the identifier `a`.

White space, formed from spaces, horizontal tabs, carriage returns, and newlines, are ignored except when they separate tokens that would otherwise combine into a single token.

Identifiers

An identifier is used to give a name to an object. It begins with a letter, and is followed by none or more letters or digits.

An identifier may have up to 255 characters.

Syntax

```
identifier::=
    identifier-letter { identifier-letter | identifier-digit }

identifier-letter::=
    (a' .. 'z', 'A' .. 'Z', '_')

identifier-digit::=
    ('0' .. '9')
```

Case sensitive

Within **Grief** identifiers are case sensitive, so that `Add`, `add` and `ADD` are all distinct identifiers.

Uniqueness and Scope

Although identifier names are arbitrary within the above syntax, errors shall result if the same name is used for more than one identifier within the same scope and sharing the same name-space. Duplicate names are legal for different name spaces regardless of scope. The scope rules are covered later (See: [Scope](#)).

Keywords

A keyword is a reserved identifier used by the language to describe a special feature. It is used in declarations to describe the basic type of an object, or in a function body to describe the statements executed. A keyword name cannot be used as an object name.

Upper and lower case letters are considered different, as such all keywords are case sensitive.

Based on the current C language specifications, the following keywords are reserved and may not be used as identifiers.

This list includes all keywords used and reserved for future use.

C89 keywords

auto	double	into	strict
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C99 keywords

_Bool	_Imaginary	inline	restrict
_Complex			

C11 keywords

_Alignas	_Atomic	_Noreturn	_Thread_local
_Alignof	_Generic	_Static_assert	

Grief specific keywords

array	foreach	list	string
declare	global	replacement	

Reserved future/experimental keywords

_command	delete	hash	throw
catch	finally	new	try

Implementation Notes

Note that the **Grief** compiler is based upon a C11 grammar which may successfully compile in some cases which are not explicitly supported resulting in unexpected execution; please consult the *Macro compatibility* Section for specific details.

Punctuators

The punctuation and special characters in the C character set have various uses, from organizing program text to defining the tasks that the compiler or the compiled program carries out.

Punctuators Characters:

[]	()	{ }	*	,	:	=	;	...	#
-----	-----	-----	---	---	---	---	---	-----	---

Note that some sequences are used as operators and as punctuation, such as *, =, :, # and ,.

They do not specify an operation to be performed. Some punctuators symbols are also operators see [Operators](#). The compiler determines their use from context

Finally several punctuators have to be used by pairs, such as "()", "[]" and "{ }".

Literals

A literal is the source code representation of a value of a primitive type, the String type , or the list type.

Syntax

```
literal::=
    integer-literal
    | floating-point-literal
    | boolean-literal
    | character-literal
    | string-literal
    | null-literal
;
```

Integer Literals

The most commonly used type is the integer. Integers are used for storing most numbers that do not require a decimal point, such as counters, sizes and indices into arrays.

An integer literal may be expressed in decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2).

Unlike a C value, an integer literal is always signed 32-bit values, and represent value in the range.

```
-2,147,483,648 to 2,147,483,647
```

An integer literal is a sequence of digits representing an integer constant.

Decimal

Decimal constants from (-2, 147, 483, 648) to (2, 147, 483, 647) are allowed. Constants exceeding this limit are truncated. Decimal constants must not use an initial zero. An integer constant that has an initial zero is interpreted as an octal constant, as above.

Octal

All constants with an initial zero are taken to be octal. If an octal constant contains the illegal digits 8 or 9, an error is reported. Octal constants exceeding 037777777777 are truncated.

Hexadecimal

All constants starting with 0x (or 0X) are taken to be hexadecimal. Hexadecimal constants exceeding 0xFFFFFFFF are truncated.

Syntax

```
integer-literal::=
    | decimal-integer-literal
    | hex-integer-literal
    | octal-integer-literal
    | binary-integer-literal
;

decimal-integer-literal::=
    decimal-numeral [{integer-type-suffix}]

hex-integer-literal::=
    hexnumeral [{integer-typesuffix}]

octal-integer-literal::=
    octalnumeral [{integer-type-suffix}]

binary-integer-literal::=
    binarynumeral [{integer-type-suffix}]

integer-type-suffix::=
    'l', 'L', 'u', 'U'
```

As denoted above, an integer literal may be notated in several ways; the notation determines the base of the literal. and whether it is signed or unsigned.

Rules

1. A literal starting with 0x or 0X is in hexadecimal notation (base 16).
2. A literal starting with 0 is in octal notation (base 8).
3. A literal starting any of the digits 1 through 9 and ending in the letter u or U is in decimal notation (base 10) and is unsigned. Also, a literal 0u or 0U is in decimal notation.
4. A literal starting with any of the digits 1 through 9 and ending in a digit is in decimal notation (base 10).
5. A literal starting with a minus sign, followed by any of the digits 1 through 9, and ending in a digit, is in decimal

notation (base 10).

Suffixes

For portability with the C language, suffixes are permitted; these currently have little effect as the underlying integer storage and associated representations is fixed yet may play more of a role in the future.

The suffix *L* (or *l*) attached to any constant forces the constant to be represented as a *long*, with *LL* (*ll*) forcing the constant to be represented as a *long long*.

Similarly the suffix *U* (or *u*) forces the constant to be *unsigned*.

It is *unsigned long* if the value of the number itself is greater than decimal 65,535, regardless of which base is used.

You can mix both *L* and *U* suffixes on the same constant in any order or case.

String Literals

A *string literal* is a sequence of characters from the source character set enclosed in double quotation marks ("").

String literals are used to represent a sequence of characters which, taken together, form a null-terminated string.

There are a number of string modifiers, allowing the specification of wide-string literals and raw-string literals.

All escape codes listed in the Escape Sequences table (See: [Escape Sequences](#)) are valid in string literals.

To represent a double quotation mark in a string literal, use the escape sequence "\\". The single quotation mark ('') can be represented without an escape sequence. The backslash (\) must be followed with a second backslash (\\) when it appears within a string. When a backslash appears at the end of a line, it is always interpreted as a line-continuation character.

Standard Strings

```
"string"
```

Wide Strings

```
L"wide-string content"
```

Raw Strings

```
R"raw-string content"
`raw string content`
```

Raw strings provide a method of specifying that a literal is to be processed without any language-specific interpretation, either by marking with a prefix, or by allowing in special sections. This avoids the need for escaping, and can yield more legible strings.

Raw strings are particularly useful when dealing with regular expressions and path, avoid the need to escape any embedded backslashes.

Example, escaped and raw pathnames

```
"The path is C:\\Foo\\\\Bar\\\"
R"The path is C:\\Foo\\Bar\\"
```

Character Literals

A *character literal* is a single character from the source character set enclosed in single quotation marks ('').

A character literal is a value of type *int*.

There are a number of character modifiers, allowing the specification of wide-character literals and raw-character literals.

All escape codes listed in the Escape Sequences table (See: [Escape Sequences](#)) are valid in character literals.

Standard Character

```
'a'
```

wide Character

```
L'\u1234'
```

Raw Character

```
R'\'
```

Escape Sequences

In order to encode the character codes of non-printing characters, Unicode character codes within non-Unicode source, allow references to platform specific control characters and the allow literal delimiters within literals, these special symbols need to be denoted with an escape mechanism.

Character combinations consisting of a backslash (\) followed by a letter or by a combination of digits are called "escape sequences."

Syntax

```

escape-value::=
  '\` escape-sequence
  | '\\' '\\'

escape-sequence::=
  unicode-escape
  | hex-escape
  | octal-escape
  | decimal-escape
  | control-escape
  | binary-escape

unicode-escape::=
  'u' '{' {digit}+ '}'
  | 'U' {digit}+
  | 'u' {digit}+           // unrestricted length
  | 'U' {digit}+           // 4 digit form
  | 'u' {digit}+           // 8 digit form

hex-escape::=
  'x' '{' {hex-digit}+ '}'
  | 'x' {hex-digit}+       // unrestricted length

decimal-escape::=
  'd' '{' {decimal-digit}+ '}'
  | decimal-nonzero {decimal-digits}*

octal-escape::=
  'o' '{' {octal-digit}+ '}'
  | '0' {octal-digit}*     // unrestricted length

control-escape::=
  'c' control-letter

binary-escape::=
  'b' '{' {binary-digit}+ '}'

hex-digit::=
  '0' .. '9', 'a' .. 'f', 'A' .. 'F'

decimal-digit::=
  '0' .. '9'

decimal-nonzero::=
  '1' .. '9'

octal-digit::=
  '0' .. '7'

control-letter::=
  'A' .. 'Z'

binary-digit::=
  '0', '1'

```

An escape sequence is regarded as a single character and is therefore valid as a character constant and within string literals.

In several cases escape sequence cannot be avoided. To represent a newline, single quotation mark ('') within character constants, double quotation mark ("") within string literal and a backslash (\) within either, you must use an escape sequence. These are in addition to all non-ASCII or non-printable character codes.

The following table lists the escape sequences and what they represent. A backslash followed by one or more digits or letters is interpreted according to the following table:

Escape Sequence Interpretation

1	\a	Alert (Bell).
2	\b	Backspace.
3	\e	Escape.
4	\f	Form feed.
5	\n	Newline.
6	\r	Carriage return .
7	\t	Horizontal tab.
8	\\\	Backslash.
9	\'	Single quote.
10	\"	Double quote.
11	\?	Question mark.
12	\xhh	The value of the hexdigit sequence, which must contain at least one and at most two hexdigits (0..f).
13	\x{h..}	Is an unrestricted hexadecimal value, which may contain one or more hex digits enclosed within brackets.
14	\o[0..]	The value of the octdigit sequence, which must contain at least one and at most three octal digits (0..7)
15	\o{0..}	An unrestricted length octal value, which may contain one or more octal digits enclosed within brackets.
16	\u####	16 bit Unicode character value, values must be complete; disallowing nul's, surrogates and reserved constants.
17	\U#####	32 bit Unicode character value, values must be complete; disallowing nul's, surrogates and reserved constants.

Octal and Hexadecimal Character Specifications

The sequence \ooo means you can specify any character in the ASCII character set as a three-digit octal character code. The numerical value of the octal integer specifies the value of the desired character or wide character.

Similarly, the sequence \xhh allows you to specify any ASCII character as a hexadecimal character code. For example, you can give the ASCII backspace character as the normal C escape sequence (\b), or you can code it as \010 (octal) or \x008 (hexadecimal).

You can use only the digits 0 through 7 in an octal escape sequence. Octal escape sequences can never be longer than three digits and are terminated by the first character that is not an octal digit. Although you do not need to use all three digits, you must use at least one. For example, the octal representation is \10 for the ASCII backspace character and \101 for the letter A, as given in an ASCII chart.

Similarly, you must use at least one digit for a hexadecimal escape sequence, but you can omit the second and third digits. Therefore you could specify the hexadecimal escape sequence for the backspace character as either \x8, \x08, or \x008.

The value of the octal or hexadecimal escape sequence should be in the range of representable values (1 .. 255) for a character constant and (1 .. 3^32) for a wide-character constant.

Note that the question mark proceeded by a backslash (\?) specifies a literal question mark, which under C has the intended use to guard against being misinterpreted as a trigraph; **Grief** does not support trigraph, as such its use is optional.

Examples

```
'\a', '\r',
'\x0', '\x12ab', '\x{1234abcd}',
'\x0', '\o10', '\o{1234567}',
```

Note, If a backslash precedes a character that does not appear in the table, the compiler handles the undefined

character as the character itself. For example, \c is treated as a c.

Floating Point Literals

A floating-point number is a number which may contain a decimal point and digits following the decimal point. The range of floating-point numbers is usually considerably larger than that of integers, but the efficiency of integers is usually much greater. Integers are always exact quantities, whereas floating-point numbers sometimes suffer from round-off error and loss of precision.

A floating-point literal is a decimal representation of a floating-point constant. It has an integer part, a decimal point, a fractional part, and an exponent part. The integer and fractional part comprise decimal digits; the exponent part is an e or E followed by an optionally signed decimal exponent. One of the integer part or the fractional part may be elided; one of the decimal point or the exponent may be elided.

Syntax

```

float_lit      :: decimals '. [decimals] [exponent]
                | decimals exponent
                | '.' decimals [exponent] .

decimals       :: decimal_digit {decimal_digit} .

exponent       :: ('e' | 'E') ['+' | '-' ] decimals .

```

Types

Each object, reference, and function in **Grief** is associated with a type, which is defined at the point of declaration and cannot change.

The type of an entity defines its possible values, possible operations, and their meaning.

Types in **Grief** are nominal: two types may define exact same range of values and allowed operations, but if they are named differently, objects of those types are generally not compatible and must be converted during comparison and assignment operations.

Basic Types

- Integer Types
- Float Types
- String Types
- Character Types
- List Types
- Polymorphic Types
- Enumerated Types

Declarations

Within a **Grief** macro is a set of tokens defining objects or variables, and functions to operate on these variables.

The **Grief** macro language uses a declaration to associate a type to a name. A type may be a simple type or a complex type.

A simple type is numerical, and may be integer or real.

Parameter Coercion

Grief provides automatic conversion between string and number values at run time.

Any arithmetic operation applied to a string tries to convert this string to a number, following the usual conversion rules.

Conversely, whenever a number is used where a string is expected, the number is converted to a string, in a reasonable format.

Integer Types

An integer is just a number.

Integers can reliably be from -2147483648 to +2147483647.

An integer is true if it is not zero.

Integer typed declarations use the *int* keyword.

Examples

```
int a;
int b, c;
int d = 1;
```

Primitives

The following primitives can act on integer data types.

- `atoi`
- `is_integer`
- `sprintf`
- `strtol`

Float Types

Floating point number, which internal is represented using a double-precision float.

Floating point number typed declarations use the either the `float` or the `double` keyword.

Examples

```
float a;
float b, c;
float d = 1.0;
```

A floating-point number is a number which may contain a decimal point and digits following the decimal point. The range of floating-point numbers is usually considerably larger than that of integers, but the efficiency of integers is usually much greater.

Integers are always exact quantities, whereas floating-point numbers sometimes suffer from round-off error and loss of precision.

Grief allows both `float` and `double` yet both refer to the same internal type, normally corresponding to a 64-bit quantity.

	Lower	Upper	Precision
<code>float</code>	2.2E-308	1.7E+308	15
<code>double</code>	2.2E-308	1.7E+308	15

Note, floating point constants compiled into macros are **not** currently stored in a machine independent manner, as such compiled macros may not be portable to different machines.

Future, floats shall be manage within objects using an IEEE float point representation allowing macros to be machine independent.

Primitives

The following primitives can act on float data types.

- `strtod`
- `strtod`
- `acos`
- `asin`
- `atan`
- `atan2`
- `ceil`
- `cos`
- `cosh`
- `exp`
- `fabs`
- `floor`
- `fmod`
- `frexp`
- `ldexp`
- `log`

- `log10`
- `modf`
- `pow`
- `sin`
- `sinh`
- `sqrt`
- `tan`
- `tanh`

String Types

A string is a sequence of printable characters.

A string is true if it is not empty.

String typed declarations use the `string` keyword.

Examples

```
string a;
string b, c;
string d = "1.0";
```

Any character may be used in a string, except for the “NUL” character, though some will need to be escaped (See: [Escape Sequences](#)).

The length of a string is only limited by available script memory.

Strings can be concatenated (joined together) using the `+` operator. Note that concatenation cannot be done when defining a string as a global variable, as `+` is treated as an executable instruction, it's not handled by the compiler. Concatenation can only be done within executable code.

Primitives

The following primitives can act on string data types.

- `characterat`
- `compress`
- `format`
- `index`
- `is_string`
- `lower`
- `Itrim`
- `rindex`
- `rtrim`
- `split`
- `sprintf`
- `sscanf`
- `strcasestr`
- `strcasecmp`
- `strfilecmp`
- `strlen`
- `strpbrk`
- `strpop`
- `strstr`
- `strtod`
- `substr`
- `tokenize`
- `trim`
- `upper`

Character Types

There is no specific character type, instead an `<integer type>` can be used to handle character data.

Any character constant may be used in a integer, including the “NUL” character, though some will need to be escaped (See: [Escape Sequences](#)).

As character types use an integer as their underlying storage, declarations use the *int* keyword.

Examples

```
int a = 'a';
int backspace = '\b';
```

Primitives

The following primitives can act on character data types.

- `characterat`
- `isdigit`
- `isalnum`
- `isalpha`
- `isascii`
- `isblank`
- `iscntrl`
- `isdigit`
- `isgraph`
- `islower`
- `isprint`
- `ispunct`
- `isspace`
- `isupper`
- `isword`
- `iscsym`
- `isxdigit`

List Types

A list is a collection of objects of any type.

Lists are useful for manipulating several objects at once.

A list is true if it is not empty.

List typed declarations use the *list* keyword.

Examples

```
list a;
list b, c;
list d = {1, 2};
```

You can initialise a list value by enclosing a comma-separated series of expressions in curly brackets.

Examples

```
list d = {1, 2};
```

To retrieve an element of a list, you can use a list selection expression.

Syntax

```
list[element]
```

Examples

```
x = d[1];
```

Primitives

The following primitives can act on list data types.

- `arg_list`
- `car`
- `cdr`
- `delete_nth`
- `get_nth`
- `is_list`
- `length_of_list`
- `list_each`
- `list_extract`
- `make_list`
- `nth`
- `pop`
- `push`
- `put_nth`
- `quote_list`
- `search_list`
- `shift`
- `sort_list`
- `splice`

Polymorphic Types

A polymorphic type is one in which the type of the variable stored can be changed.

The `declare` keyword is used to create a polymorphic variable.

Examples

```
declare a;
declare b, c;
```

These are normally used as function parameters when it is not known until run-time what the actual type will be, or for looking at elements in a list.

Null Type

On declaration `declared` variables are assigned as type of undefined.

The Null type has exactly one value, called null.

The actual type of a polymorphic variable is set upon assignment and remains that type until the next assignment.

Examples

```
declare a;           // upon declare, is the 'null' type.
a = 0;              // integer constant assignment, type 'int'
a = 1.0;            // float constant assignment, type 'float'
```

Primitives

The following primitives can act on polymorphic types.

- `cvt_to_object`
- `is_float`
- `is_integer`
- `is_list`
- `is_null`
- `is_string`
- `is_type`

Enumerated Types

At times it is desirable to have a list of constant values representing different items, and the exact values are not relevant nor can code easily be neither read nor understand. They may need to be unique or may have duplicates. For example, a set of actions, colours or keys might be represented in such a list.

An enumerated type allows the creation of a list of items.

Enumeration Form

```
enumeration      : enum identifier
| enum { enumeration-constant-list }
| enum identifier { enumeration-constant-list }
;
```

Integer Enumerations

An enumerated type is a set of identifiers that correspond to constants of type integer constant expression.

By default, the first enumerator has a value of 0, and each successive enumerator is one larger than the value of the previous one, unless you explicitly specify a value for a particular enumerator. Enumerators need not have unique values within an enumeration. The name of each enumerator is treated as a constant and must be unique within the scope where the enum is defined.

For example, the four suits in a deck of playing cards may be four enumerators named CLUB, DIAMOND, HEART, SPADE, as follows;

```
enum suits { CLUB, DIAMOND, HEART, SPADE };
```

An enumerated type may be given an optional tag (name) with which it may be identified elsewhere in the program. In the example above, the tag of the enumerated type is *suits*, which becomes a new type. If no tag is given, then only those objects listed following the definition of the type may have the enumerated type.

Enumeration constants may be given a specific value by specifying a *tag* = followed by the value.

Examples

```
enum suits { CLUB = 1, DIAMOND = 2, HEART = 3, SPADE = 4 };
```

creates the constants CLUB, DIAMOND, HEART and SPADE with values 1, 2 and 4 respectively.

```
enum fruits { WHITE = 1, RED, GREEN = 6, BLUE, BLACK = 0 };
```

creates constants with values 1, 2, 6, 7 and 0.

String Enumerations

Grief also supports enumerations which are maybe assigned string-literals. Yet unlike integer enumeration lists, once a string is assigned all following enumerated values must be explicitly stated as it is not possible to automatically assign the next value in the sequence.

Expressions

An expression is a sequence of operators and operands that describes how to,

- calculate a value (e.g. addition)
- create side-effects (e.g. assignment, increment) or both.

The order of execution of the expression is usually determined by a mixture of,

1. parentheses (), which indicate to the compiler the desired grouping of operations,
2. the precedence of operators, which describes the relative priority of operators in the absence of parentheses,
3. the common algebraic ordering,
4. the associativity of operators.

In most other cases, the order of execution is determined by the compiler and may not be relied upon. Exceptions to this rule are described in the relevant section. Most users will find that the order of execution is well-defined and intuitive. However, when in doubt, use parentheses. The table below summarizes the levels of precedence in expressions.

Operators

An operator is used to describe an operation applied to one or several objects. It is mainly meaningful in expressions, but also in declarations. It is generally a short sequence using non alphanumeric characters.

Grief supports a rich set of operators, which are symbols used within an expression to specify the manipulations to be performed while evaluating that expression.

Available operators

- Array Subscripting
- Function Calls
- Increment and Decrement Operators
- Unary Arithmetic Operators
- Arithmetic Operators
- Bitwise Shift Operators
- Relational Operators
- Equality Operators
- Bitwise Logical Operators
- Relational Operators
- Logical Operators
- Conditional Operators
- Assignment Operators
- Comma Operator

Scalar Types

Most operators act on expressions and/or scalar types.

A scalar (or base type) is a single unit of data. Scalar data types are single-valued data types, that can be used for individual variables, constants, etc.

The following types are

- `int`
- `float` or `double`
- `string`
- `bool`

`String` is a bit of a hybrid. A string is made up of multiple characters, that can be manipulated individually. It still counts as a scalar type, though, since a string can be treated as a single data value.

Operator Precedence

The following is a table that lists the precedence and associativity of all the operators in the **Grief** language.

Operators are listed top to bottom, in descending precedence. Descending precedence refers to the priority of evaluation. Considering an expression, an operator which is listed on some row will be evaluated prior to any operator that is listed on a row further below it. Operators that are in the same cell are evaluated with the same precedence, in the given direction.

Operators	Association
::	None
() [] -> .	L -> R
! ~ ++ -- -	R -> L
* / %	L -> R
+	L -> R
<< >>	L -> R
< <= > >=	L -> R
== !=	L -> R
&	L -> R
^	L -> R
	L -> R
&&	L -> R
	L -> R
?:	R -> L
= += -= *= /= %= >>= <<= &= ^= = <=>	R -> L
,	L -> R

Array Subscripting

Array Subscripting Operations are executed by the following operators.

[]

General form

array[index]

where array must have the type *list* or *array*, and index must have an integral type. The result has type "type".

Note that index is scaled automatically to account for the size of the elements of array.

Function Calls

Function calls executed by the following operators

()

Syntax

```
function-expression:
    function-name (argument-list)

argument-list:
    one or more <expression> separated by commas
```

A *function-name* followed by a set of parentheses(' ,) containing zero or more comma-separated expressions is a *function-expression*.

The function-name denotes the function to be called, and must be known function. The simplest form of this expression is an identifier which is the name of a function. For example, `function()` calls the function *function*.

```

1  function()
2
3  function1(1)
4
5  function2(1, 2)
```

Increment and Decrement Operators

Increment and Decrement operations are executed by the following operators.

```
++, --
```

Syntax

```
unary-expression:
    '++' expression
    '--' expression
    | expression '++'
    | expression '--'
```

The operand of the increment and decrement operators must be a modifiable value, generally a *int* or *float* data types.

They are two forms, either prefix or postfix.

- prefix** **Prefix Increment/Decrement**, The operand is incremented or decremented by 1, with the result of the operation returned.
- postfix** **Postfix Increment/Decrement**, The effect of the operation is that the operand is incremented or decremented by 1, with the original value prior to the operation returned. In other words, the original value of the operand is used in the expression, and then it is incremented or decremented.

Unary Arithmetic Operators

Unary Arithmetic operations are executed by the following operators.

```
+, -, ~, !
```

Syntax

```
unary-expression:
    '+' expression
    | '-' expression
    | '~' expression
    | '!' expression
```

- '+' **Unary positive**, simply returns the value of its operand. The type of its operand must be an arithmetic type (character, integer or floating-point). Integral promotion is performed on the operand, and the result has the promoted type.
- '-' **Unary minus**, is the negation or negative operator. The type of its operand must be an arithmetic type (character, integer or floating-point). The result is the negative of the operand. Integral promotion is performed on the operand, and the result has the promoted type. The expression `-obj` is equivalent to `(0-obj)`.
- '~' **Bitwise complement**, 1's complement or bitwise not operator. The type of the operand must be an integral type, and integral promotion is performed on the operand. The type of the result is the type of the promoted operand. Each bit of the result is the complement of the corresponding bit in the operand, effectively turning 0 bits to 1, and 1 bits to 0. The `!` symbol is the logical not operator. Its operand must be a scalar type (not a structure, union or array). The result type is *int*. If the operand has the value zero, then the result value is 1. If the operand has some other value, then the result is 0.

' !' **Not**; Its operand must be a numeric type. The result type is int. If the operand has the value zero, then the result value is 1. If the operand has some other value, then the result is 0.

Arithmetic Operators

Arithmetic operations are executed by the following operators.

* , / , % , + , -

Syntax

```
arithmetic-expression:
    expression '*' expression
    | expression '/' expression
    | expression '%' expression
    | expression '+' expression
    | expression '-' expression
```

- ' *' **Multiplication**, yields the product of its operands. The operands must have arithmetic types.
- ' /' **Division**, yields the quotient from the division of the first operand by the second operand. The operands must have numeric types.
- ' %' **Modulus**, yields the remainder from the division of the first operand by the second operand. The operands must have numeric types.
- ' +' **Addition**, yields the sum of its operands resulting from the addition of the first operand with the second.
- ' -' **Subtraction**, yields the difference resulting from the subtraction of the second operand from the first.

Bitwise Shift Operators

Bitwise Shift operations are executed by the following operators

<< , >>

Syntax

```
bitwise-shift:
    expression '<<' expression
    | expression '>>' expression
    ;
```

- ' <<' **Left-shift Operator**; Both operands must have an integral type, and the integral promotions are performed on them. The type of the result is the type of the promoted left operand.
- ' >>' **Right-shift Operator**; Both operands must have an integral type, and the integral promotions are performed on them. The type of the result is the type of the promoted left operand.

Relational Operators

Relational operations are executed by the following operators

< , > , <= , =>

Syntax

```

relational-expression:
    expression '<'     expression
    | expression '>'     expression
    | expression '<='    expression
    | expression '>='    expression
    | expression '<=>'  expression

```

- '<' **Less than**, yields the value 1 if the relation is true, and 0 if the relation is false. The result type is int.
- '>' **Greater than**, yields the value 1 if the relation is true, and 0 if the relation is false. The result type is int.
- '<=' **Less than or equal to**, yields the value 1 if the relation is true, and 0 if the relation is false. The result type is int.
- '>=' **Greater than or equal to**, yields the value 1 if the relation is true, and 0 if the relation is false. The result type is int.
- '<=>' **Comparison**. yields the value -1 if the first expression is less than the second, 0 if the equals, and 1 the greater than. The result type is int.

Equality Operators.

Equality operations are executed by the following operators

```

==, !=

```

Syntax

```

equality-expression:
    expression '==' expression
    | expression '!= expression

```

- '==' **Equals**, yields the value 1 if the relation is true, and 0 if the relation is false. The result type is int
- '!=' **Not equals**. yields the value 1 if the relation is true, and 0 if the relation is false. The result type is int

Bitwise Logical Operators.

Bitwise Logical operations are executed by the following operators

```

~, &, |, ^

```

- '~' **Bitwise complement**, 1's complement or bitwise not operator. The type of the operand must be an integral type, and integral promotion is performed on the operand. The type of the result is the type of the promoted operand. Each bit of the result is the complement of the corresponding bit in the operand, effectively turning 0 bits to 1, and 1 bit to 0.
- '&' **Bitwise AND operator**, The result is the bitwise AND of the two operands. That is, the bit in the result is set if and only if each of the corresponding bits in the operands are set.
- '|' **Bitwise inclusive OR operator**, The result is the bitwise inclusive OR of the two operands. That is, the bit in the result is set if at least one of the corresponding bits in the operands is set.
- '^^' **Bitwise exclusive OR operator**, The result is the bitwise exclusive OR of the two operands. That is, the bit in the result is set if and only if exactly one of the corresponding bits in the operands are set.

Logical Operators

Logical operations are executed by the following operators

```

&&, ||

```

Syntax

```
logicalexpression:
    '&&' expression
    | '||' expression
```

'`&&`' **Logical AND operator**, Each of the operands must have scalar type. If both of the operands are not equal to zero, then the result is 1. Otherwise, the result is zero. The result type is int.

'`||`' **Logical OR operator**, Each of the operands must have scalar type. If one or both of the operands is not equal to zero, then the result is 1. Otherwise, the result is zero (both operands are zero). The result type is int.

Short Circuit Evaluation

Logical operators are executed using *short-circuit* semantics whereby the second argument is only executed or evaluated if the first argument does not suffice to determine the value of the expression:

- Logical ADD - If the first operand is zero, then the second operand is not evaluated. Any side effects that would have happened if the second operand had been executed do not happen. Any function calls encountered in the second operand do not take place.
- Logical OR - If the first operand is not zero, then the second operand is not evaluated. Any side effects that would have happened if the second operand had been executed do not happen. Any function calls encountered in the second operand shall not take place.

Conditional Operator

Inline *Conditional* operation are executed by the following operators

```
? :
```

Syntax

```
conditional-expression '?' expression ':' expression
```

The ? token separates the first two parts of a conditional operator, and the : token separates the second and third parts.

The first operand is evaluated. If its value is not equal to zero, then the second operand is evaluated and its value is the result. Otherwise, the third operand is evaluated and its value is the result. Whichever operand is evaluated, the other is not evaluated. Any side effects that might have happened during the evaluation of the other operand shall not happen.

Assignment Operators

Assignment operations are executed by the following operators

```
=
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
```

Assignment operators store a value in the object designated by the left operand.

There are two kinds of assignment operations;

- simple assignment, in which the value of the second operand is stored in the object specified by the first operand
- augmented assignment, in which an arithmetic, shift, or bitwise operation is performed prior to storing the result.

All assignment operators in the following table are augmented except the simple = operator;

' <code>=</code> '	Store the value of the second operand in the object specified by the first operand (simple assignment).
' <code>*=</code> '	Multiply the value of the first operand by the value of the second operand; store the result in the object specified by the first operand.
' <code>/=</code> '	Divide the value of the first operand by the value of the second operand; store the result in the object specified by the first operand.
' <code>%=</code> '	Take the modulus of the first operand specified by the value of the second operand; store the result in the object specified by the first operand.

- '`+=`' Add the value of the second operand to the value of the first operand; store the result in the object specified by the first operand.
- '`-=`' Subtract the value of the second operand from the value of the first operand; store the result in the object specified by the first operand.
- '`<<=`' Shift the value of the first operand left the number of bits specified by the value of the second operand; store the result in the object specified by the first operand.
- '`>>=`' Shift the value of the first operand right the number of bits specified by the value of the second operand; store the result in the object specified by the first operand.
- '`&=`' Obtain the bitwise AND of the first and second operands; store the result in the object specified by the first operand.
- '`^=`' Obtain the bitwise exclusive OR of the first and second operands; store the result in the object specified by the first operand.
- '`|=`' Obtain the bitwise inclusive OR of the first and second operands; store the result in the object specified by the first operand.

Comma Operator

The *Comma Operator*.

```
,
```

Syntax

```
expression ',' expression
```

At the lowest precedence, the comma operator evaluates the left operand as a void expression (it is evaluated and its result, if any, is discarded), and then evaluates the right operand. The result has the type and value of the second operand.

In contexts where the comma is also used as a separator (function argument lists and initialiser lists), a comma expression must be placed in parentheses.

Declarations

Variables and functions are declared in the same way in **Grief** as they are defined in C.

Storage Class

A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program. These specifiers precede the type that they modify.

There are following storage classes which can be used within a Grief macro.

auto

The **auto** storage class is the default storage class for all local variables.

static

The **static** storage class instructs the compiler to keep a local variable in existence during the lifetime of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

When applied to a local variable any initialisation shall occur upon the first execution of its parent function. This is implemented using the **first_time** primitive.

The **static** modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

When applied to a global variable any initialisation shall occur upon the macro loader executed an internally defined and managed function **_init()**.

extern

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use **extern** the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function which will be used in other files also, then **extern** will be used in another file to give reference of defined variable or function.

The `extern` modifier is most commonly used when there are two or more files sharing the same global variables or functions.

Grief has an additional usage related to dynamically scoped variables. As an aid to guard against the use of "dynamic scoping" as the result of a coding bug rather than by design the **Grief** Macro compiler enforces "static scoping" on all variable references. For an example (See: [Scope](#)).

replacement

The `replacement` keyword is used to explicitly declare overloaded interface, which is a macro that supersedes (or complements) another macro of the same name.

Function Declarations

A function is a group of statements that together perform a task. Every **Grief** macro shall have at least one function being its primary entry point, either as `main()` or the name of the functionality to implemented; the general convention is that entry point match the macro object name, allowing the autoload function to locate the macro at runtime..

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

General Form

```
[class] type
name( parameters )
{
    // body of the function
}
```

class **Storage class;** A functions storage class defines the scope (visibility) of the function. If omitted by functions are visible to all macros.

type **Return type;** A function must have a return a value. The *return type* is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the `return_type` is the keyword `void`.

name **Function name;** This is the actual name of the function. The function name and the parameter list together constitute the function signature.

parameters **Parameter list;** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function.

body **Function body;** The function body contains a collection of statements that define what the function does.

Note, care should be taken to manage the global function namespace and unless they are intended as macro entry points or as library functions to other macros, a macro function should be declared as `static`.

Global functions are managed within a namespace where the function name is assumed to be unique, with only the last loaded image of any given function name shall be remembered, overwriting any previous with the same name.

Main function

Every macro source may have a `main()` function; where you place it is a matter of preference. Like all functions, their location or order of function declarations contained the macro source has not effect of the execution order.

As such like C, some programmers place main at the beginning of the file, others at the very end. But regardless of its location, the following points about main always apply.

Parameter List

Parameters are optional is that the list can be given as either `void` or empty, meaning the function takes no parameters, or a comma-separated list of declarations of the objects, including both type and parameter name (identifier).

If multiple arguments of the same type are specified, the type of each argument must be given individually.

If the parameter-type-list ends with `...` then the function will accept a variable number of arguments; furthermore unlike C which requires at least one parameter before the `...`, within **Grief** `...` is permitted as the only parameter specification.

```
function(...)
```

Parameter Syntax

Components within the parameter-list should have one of the following forms.

```
type name
```

Argument is mandatory, and of type string and is referred to as name in the function.

```
~type name
```

Argument is optional (can be omitted in the call or passed as NULL), and is referred to as name in the function.

```
type name = constant-expression
```

Argument is optional. If omitted from the function call, then constant expression shall be used as a default value.

```
type &name
```

Argument is mandatory of the specified type and is referred to as name in the function as reference bound by the `ref_parm()` primitive.

```
~type
```

Argument is optional/unnamed and is not directly accessible in the defining function by name. Typically this is used for place-holder arguments.

The actual argument can be accessed by calling the `get_parm()` primitive.

Lazy Evaluation

To fully understand the Grief calling convention, examples of parameter implementation are required.

At the source level macros have a very similar look and feel in C functions, yet this can be little deceiving. All functions parameters are implemented using the set of primitives `get_parm`, `put_parm` and `ref_parm`, includes ones which are declared within parameter lists.

For example given the following function declaration, which takes three parameters, an *integer*, secondary a *string* and thirdly a *list*.

```
1 void
2 function(int i, string s, list l)
3 {
4 }
```

Internally the compiler implements the parameter list using the `get_parm` primitive, with the result being parameters are not directly evaluation upon the macro execution; only as a result of an explicit `get_parm` execution.

```
1 void
2 function()
3 {
4     int      i;
5     string   s;
6     list    l;
7
8     get_parm(0, i);      // 1st argument
9     get_parm(1, s);      // 2nd argument
10    get_parm(2, l);      // 3th argument
11
12    :
13 }
```

If not all parameters are named it shall then become the macros writers responsibility to retrieve the parameter

values, which does not necessary need to done in the order of there declaration nor at all if the parameters are not required.

For example, within the following function the second and third arguments are optionally evaluated, based upon the value of the first.

```

1 void
2 function(~int, ~string, ~list)
3 {
4     int      i;
5
6     get_parm(0, i);          // evaluate and retrieve 1st arg.
7
8     if (2 == i) {
9         string s;
10
11        get_parm(1, s);    // optional 2nd arg processing.
12
13        :
14
15    } else if (3 == i) {
16        list l;
17
18        get_parm(2, s);    // optional 3nd arg processing.
19
20        :
21    }
22 }
```

Lazy Evaluation

The side effect that parameters may or may not be evaluated in addition the order of evaluation is not always predefined can seem that macros have miss-behaving. In general care must be taken when calling non-built-in macros that non named arguments have no side effects. Consider the following macro

```

1 static void
2 printit(string str, ~int)
3 {
4     if (str != "") {
5         int pos;
6         get_parm(1, pos);
7         message(%d: %s\n", str, pos);
8     }
9 }
10
11 int
12 print_tokens(string str)
13 {
14     list tokens;
15     int len, i;
16
17     tokens = split(str, ",");
18     len = length_of_list(tokens);
19     while (len > 0) {
20         echoit(tokens[i++], --len);
21     }
22 }
```

In the call to the function *echoit*, the second parameter is specified as *++i*. This will not cause *i* to be incremented until it is *referenced* in the function *printit()*. This shall only occur upon the *get_parm()*.

```
get_parm(1, pos);
```

Note, as a result of lazy evaluation in the above example the second argument may not always be executed, in this case upon an empty string, resulting in the *--len* not occurred. Worst in this case as the associated variable *len* is a part of the loop expression which would in turn the loop to never exit.

Function Prototypes

Function prototypes provide the compiler with type information about a function without providing any code.

Grief removes the need for explicit prototypes for builtin function, yet user defined functions should be prototyped.

Both prototype forms allow compile time checks against the required function arguments and the supplied values.

The syntax for defining a function prototype is identical to defining a function except that a semicolon (;) is placed after the closing parentheses of the parameter list.

Within macro source the preprocessor constant **__PROTOTYPES__** signals that prototype checks are enabled and maybe used for conditional prototype definitions.

Unlike C++, default arguments in prototypes have no effect, instead they must be stated within the function declaration.

Examples

```

1 #if defined(__PROTOTYPES__)
2 extern int box(int lx, int by, int rx, int ty, ~string, ~string);
3 extern int beep(void);
4#endif

```

Scope

Variables and functions can be used only in certain regions of a program. This area is called the "scope" of the name. Scope determines the "lifetime" of a name that does not denote an object of static extent. Scope also determines the visibility of a name, when module constructors, and when variables local to the scope is initialized.

Grief supports the concepts of multiple storage classes for variables, supported thru the *extern* and *static* keywords plus the *module()* and *make_local_variable()* primitives.

Furthermore variable access utilises "dynamic scoping" at run-time. Dynamic scoping is similar to the scoping rules of *Lisp* rather than *C*.

Internally the **Grief** macro byte-code is a *Lisp* like interpreted language. As with *Lisp* it supports "dynamic scoping". Using this scoping rule, the interpreter first look for a local definition of a variable. If it is not found, it searches a number of other resources, including look up the calling stack for a definition (See: [Scope Rules](#)).

As an aid to guard against the use of "dynamic scoping" as the result of a coding bug rather than by design the **Grief** Macro compiler enforces "static scoping" on all variable references.

Like *C*, all Variables which are referenced must be visible to that block of code, either as a global or an explicit *extern* declaration. As such you should avoid the use of public *extern* declaration at a global level where possible.

```

1 void
2 func1()
3 {
4     int x = 99;           // locally defined type
5     func2();
6     message("x1 = %d", x);
7 }
8
9 static void
10 func()
11 {
12     extern int x;        // extern type
13     message("x2 = %d", x); // references 'x' within func1()
14     ++x;
15 }
16
17 output:
18     x2 = 99
19     x1 = 100

```

The above example shows "dynamic scoping" in action. Being a simple case, the full power of dynamic scoping is not truly visible. One design pattern for the use of dynamic scoping would be for a highly recursive set of macros, reducing the need to pass parameters between callers and allowing the call-chain to automatically act as a variable stack.

Scope Rules

The four kinds of scope.

Local scope

A name declared within a block is accessible only within that block and blocks enclosed by it, and only after the point of declaration.

The names of formal arguments to a function in the scope of the outermost block of the function have local scope, as if they had been declared inside the block enclosing the function body.

These go out of scope when the associated block is terminated.

Buffer scope

Variables which are declared and then acted upon using `make_local_variable` primitive.

Which go out of scope when the current buffer is changed.

Buffer-local variables are useful for saving state information on a per buffer basis.

File scope

Any name declared outside all blocks or functions has file scope. It is accessible anywhere in the translation unit after its declaration.

Names with file scope that do not declare static objects are often called global names.

Module scope

Global scope variables which are declared in objects which are associated using the `module` primitive.

Rules

When searching for a variable, **Grief** searches the symbol tables in the following order:

1. static variable definition in the current function.
2. buffer local variable.
3. local variables of a current block.
4. nested stack frames to the outermost function call “dynamic scope”.
5. global variable.

Example

As the result of these rules care should be taken when overloading symbol names.

Grief shall permit local variables, global variables and buffer variables to all be visible at once, in which case the variable at the highest level in the above list shall be the only one accessible.

A specific example it is possible to confuse **Grief** by declaring static variables inside local blocks (i.e. instead of at the start of a function) and an outer block define variables with the same name but with different attributes inside the nested block.

For example, the following code on the output of the int *variable* it shall have the value of 1 on the first execution then 2 on the each subsequent call:

```

1 void
2 myfunction(void)
3 {
4     int variable;
5
6     // On the first call shall access the local version of
7     // 'variable' as the static has been yet to be
8     // defined, but on each subsequent calls the static
9     // version of 'variable' is referenced.
10    //
11    variable = 2;
12
13    {
14        // On first execution its initial value shall be
15        // one and be defined at function NOT block scope
16        // like C/C++.
17        //
18        static int variable = 1;
19    }
20
21    // The variable with the highest level shall be
22    // accessed, being the function level static
23    // declaration.
24    //
25    message("%d", variable);
26}
27
28 output:
29    1
30    2

```

Grief provides a mechanism for alternative namespaces, being an abstract container providing context for the items, to protect modules from accessing on each other's variables. The purpose of this functionality is to provide data hiding within a set of common macro files (objects), by allowed separate function and variables namespaces, in effect reducing naming conflicts in unrelated objects (See: [Scope](#)).

The module statement declares the object as being in the given namespace. The scope of the module declaration is from the declaration itself and effects all current and future declarations within the associated object.

The intended usage of the module() primitive is the within the main function in all of the related objects.

Namespaces

The namespace specification should be a string containing a valid sequence of identifier symbols (e.g. [A-Za-z_][A-Za-z_0-9]* describes the set of valid identifiers).

Namespaces are in conjunction with static scoping of members (See: [static](#)). If you are writing a set of macros some of which are internal and some for external use, then you can use the `static` declaration specifier to restrict the visibility of a member variable or function.

However as static functions are hidden from usage outside their own macro file (or module), this can present a problem with functionality which involves the usage of call-backs (e.g. `assign_to_key`). In this case, the `::` (scope resolution) operator is used to qualify hidden names so that they can still be used.

Multiple objects can be contained within the same namespace. The module primitive allows you to refer to static functions defined in another macro file explicitly, using the `::` naming modifier, and also allows static macro functions to be accessed in call-backs. Upon a module being associated, it is possible to use the syntax "`<module-name>::<function>`" to reference functions in call-backs.

Example

```

1 void
2 main()
3 {
4     module("my_module");
5     assign_to_key("<Alt-D>", "my_module::doit");
6
7     /* which to has the identical behaviour as */
8     assign_to_key("<Alt-D>", "::doit");
9
10    /* and */
11    assign_to_key("<Alt-D>", inq_module() + "::doit");
12 }
13
14 static void
15 doit()
16 {
17     :
18 }
```

Statements

A statement describes what actions are to be performed. Statements may only be placed inside functions. Statements are executed in sequence, except where described below.

Syntax

```

statement:
    compound-statement
    | expression-statement
    | selection-statement
    | iteration-statement
    | jump-statement
```

Compound Statements

A compound statement is a set of statements grouped together inside braces. It may have its own declarations of objects, with or without initializations, and may or may not have any executable statements. A compound statement is also called a *block* or *block statement*.

Compound statement general format

```
{ declaration-list statement-list }
```

where declaration-list is a list of zero or more declarations of objects to be used in the block (See: [Declarations](#)).

statement-list is a list of zero or more statements to be executed when the block is entered.

Expression Statement

A statement that is an expression is evaluated as a void expression for its side effects, such as the assigning of a value with the assignment operator. The result of the expression is discarded. This discarding may be made explicit by casting the expression as a void.

Example

```
count = 3;
```

consists of the expression count = 3, which has the side effect of assigning the value 3 to the object count. The result of the expression is 3, with the type the same as the type of count. The result is not used any further.

Selection Statements

A selection statement evaluates an expression, called the controlling expression, then based on the result selects from a set of statements are then executed.

The general form of a typical selection structure is as follows:

□

There are two primary forms of Selection Statements,

- **if statement** - An if statement consists of a boolean expression followed by one or more statements, followed by an optional else statement, which executes when the boolean expression is false.
- **switch statement** - A switch statement allows a variable to be tested for equality against a list of values.

Iteration Statements

Iteration statements control looping.

An iteration or loop statement allows us to execute a statement or group of statements multiple times and the following is the general form of a loop statement:

□

There are three forms of iteration Statements,

- **while statement** - Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
- **do-while statement** - Like a while statement, except that it tests the condition at the end of the loop body
- **for statement** - Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

The controlling expression must have a scalar type. The loop body (often a compound statement or block) is executed repeatedly until the controlling expression is equal to zero.

Jump Statements

A jump statement causes execution to continue at a specific place in a program, without executing any other intervening statements.

There are three jump statements,

- **continue statement**
- **break statement**
- **return statement**
- **returns statement**

Note: C/C++ *goto* and *label* constructs are not supported.

if statement

if-else selection clause.

General Form

```
if ( expression ) statement
or
if ( expression ) statement else statement
```

In the first form, if expression is true (nonzero), statement is executed. If expression is false, statement is ignored.

In the second form, the *else* is executed if the controlling expression evaluates to zero. Each statement may be a compound statement. For example,

```
if (returncode <= 0) {
    message("error: %d", returncode);
    ret = FALSE;
} else {
    ret = TRUE;
}
```

Dangling Else

As in C and C++, the *if* statement suffers from what is generally referred to as the "dangling else problem".

Within an *if-else* construct a seemingly well-defined statement can become *ambiguous* as the result of an miss assumed association between an opening *if* and a trailing *else* when *if* are nested. This problem shall be illustrated by this misleadingly formatted example:

```
if (returncode <= 0)
    if (display_errors)
        message("error: %d", returncode);

else message("success");           // dangling "else"
```

The issue is that both the outer *if* statement and the inner *if* statement might conceivably own the *else* clause.

In this example, one might surmise that the programmer intended the *else* clause to belong to the outer *if* statement.

The **Grief** language, like C and C++, arbitrarily decree that an *else* clause belongs to the innermost *if* to which it might possibly belong.

A corrected implementation.

```
if (returncode <= 0) {
    if (display_errors) {
        message("error: %d", returncode);
    }
} else {
    message("success");
}
```

If-else Style

On the matter of macro coding style, this example illustrates why it is a sound idea to always use braces to explicitly state the subject of the control structures.

```
if (returncode <= 0) {
    if (display_errors) {
        message("error: %d", returncode);
    }
} else {
    message("success");
}
```

where all subjects of the control structures are contained within braces, leaving no doubt about the meaning. A dangling *else* cannot occur if braces are always used

switch statement

switch selection clause.

General Form

```
switch( expression ) statement
```

The switch and case statements help control complex conditional and branching operations. The switch statement transfers control to a statement within its body.

Usually a statement is a compound statement or block. Embedded within the statement are case labels and possibly a default label, of the following form:

```
case expression : statement
default : statement
```

Control passes to the statement whose *case* expression matches the value of *switch (expression)*. The *switch* statement can include any number of case instances, but no two case constants within the same switch statement can have the same value.

Execution of the statement body begins at the selected statement and proceeds until the end of the body or until a *break* statement transfers control out of the body.

The default statement is executed if no case constant-expression is equal to the value of switch expression. If the default statement is omitted, and no case match is found, none of the statements in the switch body are executed. There can be at most one default statement. The default statement need not come at the end; it can appear anywhere in the body of the switch statement.

The *default* label may appear at most once in any switch block.

A *case* or *default* label can only appear inside a switch statement.



The controlling expression and the expressions on each case label all must have integral type. Unlike c/C++, *case* expressions need not be constant, yet when constant no two of the case constant-expressions may be the same value.

Example

```
void
echonumber(int num)
{
    switch (num) {
        case 1:
        case 2:
        case 3:
            message( "less than 4" );
            break;
        case 5:
        case 7:
        case 9:
            message( "old" );
            break;
        case 4:
        case 6:
        case 8:
            message( "even" );
            break;
        default:
            message( "greater than 9" );
    }
}
```

Case Blocks

The statements associated with which label may contain their own block, allowing local declaration of variables and other resources, for example:

```

switch (num) {
case 5:
case 7:
case 9: {
    string msg1 = odd(num);
    message( msg1 );
}
break;
case 4:
case 6:
case 8: {
    string msg2 = even(num);
    message( msg2 );
}
break;
}

```

Case Flow

Unlike C/C++ and more similar to C#, there is no implicit fall-through behaviour between case blocks. That is, on the completion of the statements associated with the matching case or block of case labels, the switch statement shall be exited ignoring any following statements; in other words each set of statements end with an implied *break*.

This behaviour is unlike C/C++ which shall continue until either a *break* or the end of the *switch* statement is encountered, for example:

```

void
echonumber(int num)
{
    switch (num) {
        case 1:
        case 2:
        case 3:
            message( "less than 4" );
            // implied break;
        case 5:
        case 7:
        case 9:
            message( "old" );
            break; // explicit break;
        case 4:
        case 6:
        case 8:
            message( "even" );
            break;
        default:
            message( "greater than 9" );
            // implied break
    }
}

```

Note: at this time there is no means of implementing explicit drop-through within *switch* statements. A C# style *goto case* extension is a possible future language extension, though the use of sub-function's to handle common functionality shall generally remove the need.

Case Style

On the matter of macro coding style, there is no additional overhead including an explicit *break* as the end of each *case* block, as a reminder case statements do not drop through.

while statement

while iteration clause.

General Form

```
while ( expression ) statement
```

The evaluation of the controlling expression takes place before each execution of the loop body (statement). If the expression evaluates to zero the first time, the loop body is not executed at all.

The statement may be a compound statement.

Use of the *continue* within a *while* statement, the jumps to the next execution of while *expression*.

A **break** shall cause execution to exit the statement body, and continue at the statement(s) following the *while* body.

Example

```

1 int
2 polluser(int iterations, string match)
3 {
4     string str;
5
6     while (iterations-- > 0) {
7
8         if (get_parm(NULL, str, "value?") <= 0) {
9             break;           // error, exit
10        }
11
12        if (str == "") {
13            continue;       // empty reply, ignore
14        }
15
16        if (str == match) {
17            return 1;        // string match
18        }
19    }
20
21    return 0;           // no match
22 }
```

The *while* loop shall be executed while the expression (*iterations-- > 0*) is true, prompting the user for input.

- Upon an error the loop is exited using **break**.
- Empty prompt replies ignored using **continue** which moves into the next while expression.
- On a match, the loop and the function is exiting using **return**.

do-while statement

do-while iteration clause.

General Form

```
do statement while ( expression );
```

The evaluation of the controlling expression takes place after each execution of the loop body (statement); therefore, the body of the loop is always executed at least once. If the expression evaluates to zero the first time, the loop body is executed exactly once.

The statement may be a compound statement.

Use of the **continue** within a *do-while* statement, the jumps to the next execution of while expression.

A **break** shall cause execution to exit the statement body, and continue at the statement(s) following the *do-while* body.

for statement

for iteration clause.

General Form

```
for ( [initialization]; [conditional]; [post] ) statement
```

initialisation-expression is an optional initialization expression and may be omitted, in which case nothing is executed in its place.

condition-expression is the optional controlling expression, and specifies an evaluation to be made before each iteration of the loop body. If the expression evaluates to zero, the loop body is not executed, and control is passed to the statement following the loop body. If the *condition-expression* is omitted, then a non-zero (true) value is assumed in its place. In this case, the statements in the loop must cause an explicit break from the loop, using a **break** or **return**.

post-expression specifies the optional operation to be performed after each iteration. A common operation would be the incrementing of a counter. As the *post-expression* is optional it may be omitted, nothing is executed in its place.

The statement may be a compound statement.

Use of the **continue** within a **for** statement, the jumps to the next execution of the *post-expression*, followed by the next *conditional-expression* evaluation.

A **break** shall cause execution to exit the statement body, and continue at the statement(s) following the **for** body.

Example

```

1 int
2 polluser(int iterations, string match)
3 {
4     string str;
5     int i;
6
7     for (i = 0; i < iterations; ++i) {
8
9         if (get_parm(NULL, str, "value?") <= 0) {
10            break;           // error, exit
11        }
12
13        if (str == "") {
14            continue;        // empty reply, ignore
15        }
16
17        if (str == match) {
18            return 1;          // string match
19        }
20    }
21
22    return 0;              // no match
23 }
```

The **for** loop shall be executed whilst the expression ($i < \text{iterations}$) is true, with the initial value of i being set to one prior to the first conditional expression test, prompting the user for input.

- Upon an error the loop is exited using **break**.
- Empty prompt replies ignored using **continue** which jumps to the end of for statement executing the completion statement, and then moves upon the next for expression evaluation.
- On a match, the loop and the function is exiting using **return**.

Valid Forms

The following are all valid usage of the **for** loop, with one or all of the expression being optional, for example.

```
for (;;)
    statement;
```

All statements in the body of the loop will be executed until a **break** statement is executed which passes control outside of the loop, or a **return** statement is executed which exits the function. This is sometimes called loop forever

```
for ( ; i > 0, --i)
    statement;
```

The counter i is assumed to be already initialized, and the loop will continue until i is zero or below. After each iteration of the loop, i shall be decremented.

continue statement

continue clause.

General Form

```
continue;
```

A **continue** statement may only appear within a loop body, and causes a jump to the inner-most loop-continuation statement (the end of the loop body).

In a **while** loop, the jump is effectively back to the while.

In a **do** loop, the jump is effectively down to the while

In a **for** statement, the jump is effectively to the closing brace of the compound-statement that is the subject of the for loop. The third expression in the for statement, which is often an increment or decrement operation, is then executed before control is returned to the loop's conditional expression.

break statement

break clause.

General Form

```
break;
```

A *break* statement may only appear in an iteration body or a switch statement.

In a iteration construct, **for**, **do**, and **while**, a break will cause execution to continue at the statement following the loop body.

In a **switch** statement, a break will cause execution to continue at the statement following the switch. If the loop or switch that contains the break is enclosed inside another loop or switch, only the inner-most loop or switch is terminated.

return statement

return clause.

General Form

```
return [expression];
```

The *return* statement causes execution of the current function to be terminated, and control is passed to the caller. A function may contain any number of return statements.

If the function is declared with a return type of void then no return statement within that function may return a value.

If the function is declared as having a return type of other than void, then any return statement with an expression will evaluate the expression and convert it to the return type. That value will be the value returned by the function.

If a return is executed without an expression, and the caller uses the value returned by the function, the behaviour is undefined since no value was returned; generally the value of previous statement shall be returned to the caller, yet it is not portable and may change in future versions.

Reaching the closing brace } that terminates the function is equivalent to executing a return statement without an expression; which shall return in the value of the last statement being returned, yet it is not portable and may change in future versions.

returns statement

returns clause.

General Form

```
returns(expression);
```

This primitive is similar to the **return** statement, except it doesn't cause the current macro to terminate. It simply sets **Grief's** internal accumulator with the value of the expression.

This primitive is not strictly compatible with the **returns()** macro of BRIEF and is not recommended as statements following may have side effects, if any other statements follow the execution of **returns**, then the accumulator will be overwritten changing the returned value.

\$Id: language.txt,v 1.6 2014/10/31 01:09:05 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Macros

Grief is built on the concept of extensibility and programmability. Macros are so ingrained in the design of BRIEF that many of the standard editing functions are implemented in the macro language, rather than in compiled code.

By writing your own macros, you can tailor the standard editor in both small ways (such as setting start-up colors, tab settings, or window positions), and large ways (such as modifying existing commands or creating new commands of your own design).

Summary

Macros

Grief is built on the concept of extensibility and programmability.

Types of Macros

Commands	Prompt line and keyword command macros.
Startup and main	Every macro source may have a <code>main()</code> function; where you place it is a matter of preference.
File Extension Macros	File extension callbacks are executed whenever GriefEdit edits a file via the <code>edit_file</code> primitive.
Registered Macros	A registered macro is one that is executed when an associated event occurs.
Replacement Macros	A replacement macros is an overloaded function, whereby an existing macro or builtin can be replaced and extended, yet allowing the replacement macro to then execute the origin
Special Purpose Macros	Similar to registered macros, special purpose macros act a signal handlers.
Built-in Functions	A built-in function is a function that was implemented in the interpreter and was not written in the Grief language.

Macro Resources

Keyboards	Most of Grief's keyboard input is either via the command prompt <code>get_parm</code> or executed through the use of keyboard resources.
Buffers	Buffers are the basic editing construct utilised by Grief .
Buffer Identifiers	Buffers can are identified by one of two means, firstly by name and secondary its associated unique buffer handle or identifier.
Marked Regions	Many commands work on certain regions of text.
Scrap Buffer	The scrap is a special buffer used for moving or copying blocks of text.
BookMarks	A bookmark allows a record of the current buffer position to be saved for quick navigation to the same location at a later time.
Buffer List	Buffers are maintained in a circular list, referred to as the buffer list.
System Buffer	There are two primary categories of buffers, user and system buffers.
Scrap Buffers	Scrap buffers are buffers which have been assigned the job of cut and paste storage.
Buffer Content	The following primitives are used to manage buffer content.
Buffer Attributes	As buffers are the basic editing construct utilised by Grief , they also represent the primary user interface.
Encoding	Grief supports both explicit and auto-detection of most file encoding types including;
Hilite Regions	Similar to marked regions, buffers may have one or more special highlighting regions defined, allowing macros to show specific buffer content.
Syntax Highlighting	Syntax highlighting allow different colors and text styles to be given to dozens of different lexical sub-elements of syntax.
Process Buffers	An external, for example a command shell, may be attached to specific buffer providing a tty style view for the underlining process.
Dictionaries	A dictionary is a collections of associations.
Windows	Windows are used to create views into buffers.
Dialogs	A dialog is a small window resource that prompts the user to make a decision or enter additional information.
Regular Expressions	A regular expression is a pattern that the regular expression engine attempts to match in input text.

Types of Macros

A function can be called from the macro language. **Grief** has a number of different kinds of macros or functions;

- Commands
- Startup and main
- File Extension Macros
- Registered Macros
- Replacement Macros
- Special Purpose Macros
- Built-in Functions

The following sections describe each of the function types.

Commands

Prompt line and keyword command macros.

A command function is a macro which are assigned to a key or key sequence. As these are normally explicitly designed to be called directly by the user from the keyboard, these shall be referred to as *commands*.

There is no specific differences between command macros and other macros, as commands can be called also from other macros.

Note:

The `_command` keyword has been reserved for a possible extension to the macro language, whereby command macros must be explicitly declared, hiding all non command macros from direct user usage.

Startup and main

Every macro source may have a `main()` function; where you place it is a matter of preference.

```

1 static int buffer;
2
3 void
4 main()
5 {           // initial our private working buffer
6     buffer = create_buffer("--working-buffer--", NULL, 1);
7 }
```

Note:

As such like C, some programmers place `main` at the beginning of the file, others at the very end. Like all macros, their location of order of function declarations contained the macro source has not effect of the execution order.

File Extension Macros

File extension callbacks are executed whenever GriefEdit edits a file via the `edit_file` primitive.

It is provided to allow macros to hook buffer loads, for example to setup defaults tabs before any extension specific settings are applied.

Once executed if defined the extension specific handler shall be executed, which should be named as `_ext`. If not available the default extension handler `_default` shall be executed.

The extension case shall be preserved on case sensitive file-systems otherwise the extension is converted to lower case.

- `register_macro`
- `_extension`
- `_default`

Registered Macros

A registered macro is one that is executed when an associated event occurs.

Types of events which can be registered include;

- A character has been typed.
- A invalid key is typed.
- A new user buffer is created.
- The current buffer is changed.
- Exiting the editor.

These events are registered using the `register_macro()` primitive and can then be unregistered using the `unregister_macro()` primitive. One or more macros can be registered against any given event, in which case each macro shall be executed upon the event, in the same order as they were registered.

Registered macro primitive include;

- `register_macro`
- `unregister_macro`
- `call_registered_macro`

Replacement Macros

A replacement macros is an overloaded function, whereby an existing macro or builtin can be replaced and extended, yet allowing the replacement macro to then execute the origin

□

The `replacement` keyword is used to explicitly declare overloaded interfaces, which is a macro that supersedes (or complements) another macro of the same name.

Special Purpose Macros

Similar to registered macros, special purpose macros act as signal handlers.

These macros are executed upon a specific event occurring, allowing macros to track these events.

Special purpose macros include;

- `_startup_complete`
- `_prompt_begin`
- `_prompt_end`
- `_bad_key`
- `_invalid_key`
- `_extension`
- `_fatal_error`
- `_chg_properties`

Built-in Functions

A built-in function is a function that was implemented in the interpreter and was not written in the **Grief** language.

See the [Library Reference](#) for the list of built-in functions.

Macro Resources

- Keyboards
- Buffers
 - Buffer Identifiers
 - Buffer List
 - Marked Regions
 - System Buffers
 - Scrap Buffers
- Windows
- Dialogs
- Dictionaries
- Regular Expressions

Keyboards

Most of Grief's keyboard input is either via the command prompt `get_parm` or executed through the use of keyboard resources.

Keyboard resources map key strokes to actions allowing

- Assignment of any commands to a virtual key-code.
- Manage individual keyboards on a keyboard stack.
- Provide local keyboards for a buffer.
- Tie keyboards to macros.

Primitives

The following primitives are used to manage keyboards.

- `assign_to_key`
- `copy_keyboard`
- `inq_assignment`
- `inq_kbd_char`
- `inq_keyboard`
- `inq_local_keyboard`
- `keyboard_flush`
- `keyboard_pop`
- `keyboard_push`
- `keyboard_typeables`
- `pause`
- `process`
- `push_back`
- `read_char`
- `remember`
- `self_insert`
- `use_local_keyboard`

Command Prompt

Prompts are used to request direct user input, either indirectly then invoking a primitive or directly through the use of `get_parm`. The command prompt has a number of macro callbacks which are invoked during the prompt session, which may be replaced to modify and/or extend the prompt behaviour.

The following special macros `_prompt_begin`, `_bad_key` and `_prompt_end`:

- | | |
|----------------------------|---|
| <code>_prompt_begin</code> | This callback is invoked prior to prompt display. If the macro <code>_prompt_begin</code> has been loaded, it is executed with a string parameter containing the value of prompt to be presented on the prompt. |
| | This macro may set the default response, use the prompt as key to load previous values or other similar functions. |
| <code>_bad_key</code> | This callback may be invoked one or more times during the command line edit session upon an invalid/unknown key being used during the edit. The <code>read_char</code> primitive retrieves the associated bad key, which can be ignored, processed or replaced. |
| <code>_prompt_end</code> | This callback is invoked during the prompt completion process. The primitive <code>inq_cmd_line</code> retrieves the value entered during the edit session. |

Buffers

Buffers are the basic editing construct utilised by **Grief**. One buffer corresponds to one piece of text being edited. You can have several buffers open at once, but can edit only one at a time.

Each buffer may hold text only limited by the available system memory with as many buffers active at any given time in memory as desired, again only limited by system resources.

Several buffers can be visible at the same time when you're splitting your window, see [Windows](#).

Primitives

The following primitives are used to create and delete buffers

- `create_buffer`
- `attach_buffer`
- `delete_buffer`
- `edit_file`

Buffer Identifiers

Buffers can be identified by one of two means, firstly by name and secondary its associated unique buffer handle or identifier.

Primitives

The following primitives are used to select buffers

- `inq_buffer`
- `set_buffer`

Marked Regions

Many commands work on certain regions of text. A region is defined by two points, the first being the anchor and the other the cursor position. Each buffer may have their own region, yet commands generally only work on the

marked region within the current buffer.

Primitives

The following primitives are used to manage regions

- `inq_marked`
- `drop_anchor`
- `end_anchor`
- `raise_anchor`
- `swap_anchor`
- `delete_block`

Scrap Buffer

The scrap is a special buffer used for moving or copying blocks of text. A scrap buffer differs from a regular buffer because:

- Blocks of text can only be copied into it or pasted from it; the content cannot be modified by the user.
- It is automatically managed whenever the user cuts or copies text.

Primitives

The following primitives are used to manage the scrap buffer.

- `cut`
- `copy`
- `paste`
- `delete_block`
- `transfer`
- `inq_scrap`
- `set_scrap_info`

BookMarks

A bookmark allows a record of the current buffer position to be saved for quick navigation to the same location at a later time.

There is no visible indication of where bookmarks are set.

Primitives

- `drop_bookmark`
- `delete_bookmark`
- `goto_bookmark`
- `bookmark_list`

Buffer List

Buffers are maintained in a circular list, referred to as the buffer list. New buffers are automatically inserted into the list on creation. To locate a buffer that is not current, the `next_buffer` and `previous_buffer` primitives along with `set_buffer` can be used to manipulate the current buffer.

Primitives

- `inq_buffer`
- `next_buffer`
- `previous_buffer`

System Buffer

There are two primary categories of buffers, user and system buffers. System buffers are similar to user buffer except they are normally skipped by user level buffer functions.

Primitives

- `set_buffer_flags`
- `inq_system`

Scrap Buffers

Scrap buffers are buffers which have been assigned the job of cut and paste storage.

Primitives

The cut and paste working storage is a special scape buffer which is controlled using the following primitives.

- `inq_scrap`
- `set_scrap_info`
- `transfer`
- `cut`
- `get_region`
- `paste`

Buffer Content

Primitives

The following primitives are used to manage buffer content.

- `insert`
- `insertf`
- `insert_buffer`

Buffer Attributes

As buffers are the basic editing construct utilised by **Grief**, they also represent the primary user interface. Each buffer has many run-time attributes which effect the way content is presented to the user.

These attributes include.

- Built-in features.
- Display effects.
- Tabs and filling.

Primitives

The following primitives are used to manage buffer attributes.

Built-in features

- `inq_buffer_flags`
- `set_buffer_flags`

Display effects

- `set_attribute`
- `inq_attribute`

Tabs and filling

- `tabs`
- `inq_tabs`
- `set_ruler`
- `inq_ruler`
- `set_indent`
- `inq_indent`
- `set_margins`
- `inq_margins`

Encoding

Grief supports both explicit and auto-detection of most file encoding types including;

- UTF-8.
- UTF-16.
- UTF-32.
- Latin1 (ISO8859-1).
- Extended ASCII.
- ASCII.

Primitives

The following primitives are used to manage character maps and buffer encodings.

- `set_encoding`

- `inq_encoding`

Character Map

- `create_char_map`
- `set_buffer_cmap`
- `inq_char_map`

Line termination support

- `set_terminator`
- `inq_terminator`

Auto detection support

- `set_file_magic`
- `inq_file_magic`

Hilite Regions

Similar to marked regions, buffers may have one or more special highlighting regions defined, allowing macros to show specific buffer content.

Primitives

The following primitives are used to manage hiliting resources.

- `hilite_create`
- `hilite_destroy`

Syntax Highlighting

Syntax highlighting allow different colors and text styles to be given to dozens of different lexical sub-elements of syntax. These include keywords, comments, control-flow statements, variables, and other elements

There are several forms of highlighting engines.

- Simple tokeniser.
- DFA regular expression tokeniser.

Primitives

The following primitives are used to manage syntax resources.

- `attach_syntax`
- `create_syntax`
- `define_keywords`
- `syntax_token`
- `detach_syntax`
- `get_color_pair`
- `syntax_build`
- `syntax_rule`
- `inq_syntax`
- `set_color_pair`
- `set_syntax_flags`
- `syntax_column_ruler`

Process Buffers

An external, for example a command shell, may be attached to specific buffer providing a tty style view for the underlining process.

Primitives

The following primitives are used to manage processes.

- `connect`
- `disconnect`
- `insert_process`
- `set_process_position`
- `inq_process_position`
- `wait`
- `wait_for`

- [send_signal](#)

Dictionaries

A dictionary is a collections of associations. Dictionaries consist of pairs of keys and their corresponding values, both arbitrary data values. Dictionaries are also known as associative arrays or hash tables.

Dictionaries require more space than lists, yet allow fast searches. Only one association in a dictionary may have a given key, that item keys must be unique.

You can iterate over a dictionary's associations using the [dict_each](#) primitive; at each step of the statement, the loop index will contain a list [key, value] for the current association.

Primitives

The following primitives are used to manage dictionary resources; see the descriptions of each function for details.

- [create_dictionary](#)
- [delete_dictionary](#)
- [set_property](#)
- [get_property](#)
- [dict_delete](#)
- [dict_each](#)
- [dict_exists](#)
- [dict_keys](#)
- [dict_values](#)
- [dict_list](#)
- [dict_name](#)
- [list_of_dictionaries](#)

Windows

Windows are used to create views into buffers. Windows are either tiled or popup.

Tiled windows

Tiled windows are the normal user views into buffers, allowing direct user interaction so that they may edit the buffer content.

The Grief window manager organises the screen into mutually non-overlapping frames. Tiled windows are not permitted to be overlapped with other tiled windows, as such are sized and positioned to own a specified rectangled area of the display.

The following primitives are used to create and delete tiled windows.

- [create_tiled_window](#)
- [display_windows](#)
- [change_window](#)
- [move_edge](#)
- [create_edge](#)
- [delete_edge](#)
- [inq_top_left](#)
- [inq_window](#)
- [inq_window_info](#)
- [set_top_left](#)
- [set_window](#)
- [next_window](#)

Popup windows

Unlike Tiled windows, Popup Windows when displayed overlay any tiled windows positioned behind and maybe sized and positions anywhere within the visible arena.

Usually, popup windows are utilised to display temporary information, for example the results of a command. They are generally created dynamically and managed by macros and are destroyed on completion of that macro.

In addition popup windows stack, allowing multiple popups to represent nested structures; one example being the help system which allows the user to drill down thru command relationships and then backout in reverse.

Primitives

The following primitives are used to create and delete popup windows.

- [create_window](#)

- [delete_window](#)
- [inq_window](#)
- [set_window](#)

Like buffers each windows has a set of attributes which effect the presentation of connected buffer contentt, these are managed using the following primitives.

- [set_window_flags](#)
- [inq_window_flags](#)

Dialogs

A dialog is a small window resource that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

Primitives

The following primitives are used to manage dialogs.

- [dialog_create](#)
- [dialog_delete](#)
- [dialog_exit](#)
- [dialog_run](#)
- [inq_dialog](#)
- [widget_get](#)
- [widget_set](#)

Regular Expressions

A regular expression is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs.

Primitives

The following primitives are used to utilise regular expressions.

- [quote_regexp](#)
- [re_comp](#)
- [re_delete](#)
- [re_result](#)
- [re_search](#)
- [re_syntax](#)
- [re_translate](#)
- [search_case](#)

in addition the following are Brief compatible interfaces which also support regular expressions.

- [search_back](#)
- [search_fwd](#)
- [search_list](#)
- [search_string](#)
- [translate](#)

\$Id: macros.txt,v 1.4 2014/10/31 01:09:05 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Preprocessor

The C preprocessor is a text processor which operates on the C source before it is actually parsed by the compiler. It provides macro and conditional features very closed to the ones available with most of the existing assemblers.

The preprocessor modifies the C program source according to special directives found in the program itself. Preprocessor directives start with a sharp sign # when found as the first significant character of a line. Preprocessor directives are line based, and all the text of a directive must be placed on a single logical line. Several physical lines can be used if all of them but the last one end with the continuation character backslash \.

There are three basic kinds of directives: macro directives, conditional directives and control directives. The macro directives allow text sequences to be replaced by some other text sequences, depending on possible parameters. The conditional directives allow selective compiling of the code depending on conditions most of the time based on symbols defined by some macro directives.

The control directives allow passing of information to the compiler in order to configure or modify its behaviour.

Summary

Preprocessor

The C preprocessor is a text processor which operates on the C source before it is actually parsed by the compiler.

Implementation

Grief utilises *ucpp* as it's preprocessor; it is designed to be quick and light, but fully compliant to the ISO standard 9899:1999, also known as C99.

Directives

The preprocessing directives control the behaviour of the preprocessor.

Conditional Compilation

The preprocessor supports conditional compilation of parts of source file.

Condition evaluation

The preprocessor supports text macro replacement.

Replacment Macros

The following macro names are predefined in any translation unit.

File Inclusion

Message Generation

Source Information

Implementation

Grief utilises *ucpp* as it's preprocessor; it is designed to be quick and light, but fully compliant to the ISO standard 9899:1999, also known as C99.

ucpp is distributed under the following license. The current source is available on Google Code, version 1.3, at

<http://code.google.com/p/ucpp>

```

*
* (c) Thomas Pornin 2002
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the distribution.
* 4. The name of the authors may not be used to endorse or promote
* products derived from this software without specific prior written
* permission.
*
* THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
* IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
* OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
* BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
* OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
```

Directives

The preprocessing directives control the behaviour of the preprocessor. Each directive occupies one line and has

the following format:

```
# character
```

The current implementation follows the "ISO standard 9899:1999", also known as C99.

A preprocessing directive consists of a sequence of preprocessing tokens that begins with a # preprocessing token that is either the first character in the source file (optionally after white space containing no new-line characters) or that follows white space containing at least one new-line character, and is ended by the next new-line character. A new-line character ends the preprocessing directive even if it occurs within what would otherwise be an invocation of a function-like macro.

Preprocessing instruction (one of define, undef, include, if, ifdef, ifndef, else, elif, endif, line, error, warning, pragma) arguments (depends on the instruction) line break

The null directive (# followed by a line break) is allowed and has no effect.

The following directive classes are available

- Conditional Compilation
Compile of parts of source file (controlled by directive #if, #ifdef, #ifndef, #else, #elif and #endif).
- Replacement
Text macros while possibly concatenating or quoting identifiers (controlled by directives #define and #undef, and operators # and ##).
- Include
Inclusion of other files (controlled by directive #include).
- Message generation
Controlled by directive #warning, #error and #pragma.
- Pragmas
Special purpose controls.

Preprocessor Lexical grammar

```

preprocessing-file      : [group]
;
group:                : group-part
                     | group group-part
;
group-part            : if-section
                     | control-line
                     | text-line
                     | '#' non-directive
;
if-section            : if-group [elif-groups] [else-group] endif-line
if-group              : '#' 'if' constant-expression new-line [group]
                     | '#' 'ifdef' identifier new-line [group]
                     | '#' 'ifndef' identifier new-line [group]
;
elif-groups           : elif-group
                     | elif-groups elif-group
;
elif-group             : '#' 'elif' constant-expression new-line [group]
;
else-group            : '#' 'else' new-line [group]
;
endif-line             : '#' 'endif' new-line
;
control-line          : '#' 'include' pp-tokens new-line
                     | '#' 'define' identifier replacement-list new-line
                     | '#' 'define' identifier lparen [identifier-list] )
                     | replacement-list new-line
                     | '#' 'define' identifier lparen ... ) replacement-list new-line
                     | '#' 'define' identifier lparen identifier-list , ... )
                     | replacement-list new-line
                     | '#' 'undef' identifier new-line
                     | '#' 'line' pp-tokens new-line
                     | '#' 'error' [pp-tokens] new-line
                     | '#' 'pragma' [pp-tokens] new-line
                     | '#' new-line
;
text-line              : [pp-tokens] new-line
;
non-directive         : pp-tokens new-line
;
lparen                : '(' not immediately preceded by white-space
;
replacement-list       : [pp-tokens]
;
pp-tokens              : preprocessing-token
                     | pp-tokens preprocessing-token
;

```

Conditional Compilation

The preprocessor supports conditional compilation of parts of source file. This behaviour is controlled by `#if`, `#else`, `#elif`, `#ifdef`, `#ifndef` and `#endif` directives.

Syntax

```

#if expression
#endif expression
#ifndef expression
#endif expression
#else
#endif

```

The conditional preprocessing block starts with `#if`, `#ifdef` or `#ifndef` directive, then optionally includes any number of `#elif` directives, then optionally includes at most one `#else` directive and is terminated with `#endif` directive. Any

inner conditional preprocessing blocks are processed separately.

Each of `#if`, `#elif`, `#else`, `#ifdef` and `#ifndef` directives control code block until first `#elif`, `#else`, `#endif` directive not belonging to any inner conditional preprocessing blocks.

`#if`, `#ifdef` and `#ifndef` directives test the specified condition ((see [below](#))) and if it evaluates to true, compiles the controlled code block. In that case subsequent `#else` and `#elif` directives are ignored. Otherwise, if the specified condition evaluates false, the controlled code block is skipped and the subsequent `#else` or `#elif` directive (if any) is processed. In the former case, the code block controlled by the `#else` directive is unconditionally compiled. In the latter case, the `#elif` directive acts as if it was `#if` directive: checks for condition, compiles or skips the controlled code block based on the result, and in the latter case processes subsequent `#elif` and `#else` directives. The conditional preprocessing block is terminated by `#endif` directive.

Condition evaluation

```
#if expression
#elif expression
```

The expression is a constant expression, using only literals and identifiers, defined using `#define` directive. Any identifier, which is not literal, non defined using `#define` directive, evaluates to 0.

The expression may contain unary operators in form defined identifier or defined (identifier) which return 1 if the identifier was defined using `#define` directive and 0 otherwise. If the expression evaluates to nonzero value, the controlled code block is included and skipped otherwise. If any used identifier is not a constant, it is replaced with 0.

Syntax

```
#ifdef expression
#ifndef expression
```

Checks if the identifier was defined using `#define` directive.

`#ifdef identifier` is essentially equivalent to `#if defined(identifier)`.

`#ifndef identifier` is essentially equivalent to `#if !defined(identifier)`.

Replacement Macros

The preprocessor supports text macro replacement. Function-like text macro replacement is also supported.

#define directives

```
#define identifier replacement-list          (1)
#define identifier( parameters ) replacement-list (2)
```

The `#define` directives define the identifier as macro, that is instruct the compiler to replace all successive occurrences of identifier with replacement-list, which can be optionally additionally processed. If the identifier is already defined as any type of macro, the program is ill-formed.

Object-like macros Object-like macros replace every occurrence of defined identifier with replacement-list. Version (1) of the `#define` directive behaves exactly like that.

Function-like macros Function-like macros replace each occurrence of defined identifier with replacement-list, additionally taking a number of arguments, which then replace corresponding occurrences of any of the parameters in the replacement-list. The number of arguments must be the same as the number of arguments in macro definition (parameters) or the program is ill-formed. If the identifier is not in functional-notation, i.e. does not have parentheses after itself, it is not replaced at all.

Version (2) of the `#define` directive defines a simple function-like macro.

#undef directive

```
#undef identifier
```

The `#undef` directive undefines the identifier, that is cancels previous definition of the identifier by `#define` directive.

If the identifier does not have associated macro, the directive is ignored.

and ## operators

In function-like macros, a `#` operator before an identifier in the replacement-list runs the identifier through parameter

replacement and encloses the result in quotes, effectively creating a string literal. In addition, the preprocessor adds backslashes to escape the quotes surrounding embedded string literals, if any, and doubles the backslashes within the string as necessary.

All leading and trailing whitespace is removed, and any sequence of whitespace in the middle of the text (but not inside embedded string literals) is collapsed to a single space. This operation is called "stringification". If the result of stringification is not a valid string literal, the behaviour is undefined.

A `##` operator between any two successive identifiers in the replacement-list runs parameter replacement on the two identifiers and then concatenates the result. This operation is called "concatenation" or "token pasting". Only tokens that form a valid token together may be pasted: identifiers that form a longer identifier, digits that form a number, or operators + and = that form a +=. A comment cannot be created by pasting / and * because comments are removed from text before macro substitution is considered. If the result of concatenation is not a valid token, the behaviour is undefined.

Predefined Symbols

The following macro names are predefined in any translation unit.

The preprocessor defines a few symbols with a built-in behaviour. Those symbols cannot be undefined by a `#undef` directive and then cannot be redefined to any other behaviour.

<code>__FILE__</code>	expands to a text string containing the name of the file being compiled.
<code>__LINE__</code>	expands to a numerical value equal to the current line number in the current source file.
<code>__DATE__</code>	expands to a text string containing the date you compiled the program. The date format is mmm dd yyyy, where mmm is the month abbreviated name, dd is the day and yyyy the year.
<code>__TIME__</code>	expands to a text string containing the time you compiled the program. The time format is hh:mm:ss, where hh is the hours, mm the minutes and ss the seconds.
<code>__FUNCTION__</code>	current function name.

The values of these macros (except for `__FILE__` and `__LINE__`) remain constant throughout the translation unit. Attempts to redefine or undefine these managed macros results in undefined behaviour.

The value of `__FUNCTION__` changes dynamically based upon the current function scope.

The following additional macro names may be predefined by the implementations.

<code>__VERSION__</code>	expands to a text string containing the compiler version.
<code>__PROTOTYPES__</code>	defined when Grief prototype support is enabled.
<code>__TIMESTAMP__</code>	current unix time, represented by the number of seconds since 1/1/1970.

File Inclusion

Syntax

```
#include <filename>           (1)
#include "filename"           (2)
```

```
#include_next "filename"      (3)
```

Includes source file, identified by filename into the current source file at the line immediately after the directive.

The first version of the directive searches only standard include directories. The standard C++ library, as well as standard C library, is implicitly included in standard include directories. The standard include directories can be controlled by the user through compiler options.

The second version firstly searches the directory where the current file resides and, only if the file is not found, searches the standard include directories.

The third form, directive `#include_next` instructs the preprocessor to continue searching for the specified file name, and to include the subsequent instance encountered after the current directory. The syntax of the directive is similar to that of `#include`. The language feature is an extension to C and C++. It extends the techniques available to address the issue of duplicate file names among applications and shared libraries.

In all cases if the file is not found, program is ill-formed.

Message Generation

```
#error [error_message]
#warning [warning_message]
```

Message generation directives include the following.

The #error directive, which defines text for a compile-time error message. After encountering #error directive, diagnostic message *error_message* is shown and the program is rendered ill-formed (the compilation is stopped).

error_message can consist of several words not necessarily in quotes.

The #warning directive, which defines text for a compile-time warning message. After encountering #warning directive, a diagnostic message.

Like the #error directive, *warning_message* is shown. *warning_message* can consist of several words not necessarily in quotes.

Source Information

```
#line lineno          (1)
#line lineno "filename" (2)
```

The #line directive, which supplies a line number for compiler messages.

Changes the current preprocessor line number to *lineno*. Expansions of the macro LINE beyond this point will expand to *lineno* plus the number of actual source code lines encountered since.

The second form also changes the current preprocessor file name to *filename*. Expansions of the macro FILE from this point will produce *filename*.

Any preprocessing tokens (macro constants or expressions) are permitted as arguments to #line as long as they expand to a valid decimal integer optionally following a valid character string.

Notes, This directive is used by some automatic code generation tools which produce C++ source files from a file written in another language. In that case, #line directives may be inserted in the generated C++ file referencing line numbers and the file name of the original (human-editable) source file.

Pragmas

The #pragma preprocessor directive is used to change various options during the course of a compile.

```
#pragma token-string
```

The *token-string* is a series of characters that gives a specific compiler instruction and arguments, if any. The number sign (#) must be the first non-white-space character on the line containing the pragma; white-space characters can separate the number sign and the word pragma.

Following #pragma, write any text that the translator can parse as preprocessing tokens. The argument to #pragma is subject to macro expansion.

If the compiler encounters a pragma it does not recognize, a warning is issued, yet the compilation shall continue.

Grief offers the following pragma's.

Message

```
#pragma message (message-string)
```

A typical use of the message pragma is to display informational messages at compile time.

The message-string parameter can be a macro that expands to a string literal, and you can concatenate such macros with string literals in any combination.

Example

```
#pragma message ("Compiling " __FILE__ ", at " __TIME__)
#if defined(TEST)
#pragma message("Notice: TEST enable")
#endif
```

Warning

```
#pragma warning (once|default|disable|enable|error : <list>
#pragma warning (level : <level>
#pragma warning (push)
```

The `pragma warning(push)` stores the current warning state for all warnings.

```
#pragma warning (pop)
```

The `pragma warning(pop)` pops the last warning state pushed onto the stack. Any changes made to the warning state between push and pop are undone.

Diagnostic message control

```
#pragma enable_message message-number
```

The `enable_message` enables the specified warning message *message-number*.

```
#pragma disable_message message-number
```

The `disable_message` disables the specified warning message *message-number*.

Autoload Management

```
#pragma autoload ("function1" ...)
```

Autoload management, experimental feature at this time.

\$Id: preprocessor.txt,v 1.3 2014/10/31 01:09:05 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Debugging

Macro Errors

Grief run-time diagnostics using the *debug* primitive.

Example trace output

```
todo ...
```

These primitive allow visible inspection of specific macro resources.

- **vars** - Display global variables.
- **bvars** - Display all buffer variables.
- **mvars** - Display all modules variables.

The following diagnostics functions are provided for macro writers to echo trace information to either the command prompt or the trace log.

- **debug** - Debug trace.
- **error** - Issue an error message.
- **message** - Display a message on the command line.
- **dprintf** - Diagnostic log output.

Compiler Errors

The following Macro compiler options maybe of help then errors are encountered

```
-d[d]           Enable internal debugging features.  
-c             Leave temporary files.
```

Macro object decompiler allows

```
Grief macro decompiler.  
gm v2.6.1 compiled Jul  8 2013 01:27:27 (cm version 20)  
  
Usage: gm [options] <file> ...  
  
options:  
  -a          Print atom percentages.  
  -l          List macro expansions.  
  -L          Print detailed disassembly info.  
  -q          Quiet error messages.  
  -s          Print size of .cm file only.  
  -o file    Name of compiled output file.  
  -h          Command line help.
```

```
$Id: debugging.txt,v 1.3 2014/10/31 01:09:05 ayoung Exp $
```

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Library Reference

This section contains the description for the all the **Grief** macro primitives. At last count over 600 primitives including operators are published by the macro library.

Summary

Library Reference

This section contains the description for the all the **Grief** macro primitives.

[Return Value](#)

The return value for most builtin functions which return an integer have the following

[Function Reference](#) Builtin constant and function reference.

Return Value

The return value for most builtin functions which return an integer have the following

- Less then zero (generally -1) the function did not complete successfully, and an error message was displayed.
- Zero, the call failed yet no error message was displayed.
- One or greater, the function execution was successful.

System Call Macros

Primitives which are based on underlying POSIX system calls shall set the global variable `errno` to the result of the underlying system call upon a -1 return.

For example, if the `remove()` function fails, then the global variable `errno` will be set to the value set by the underlying system call. The primitive `strerror()` maybe used to decode the error code. For a complete list (See: `errno`) and `<grief.h>`.

Within their descriptions the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in *RFC 2119*.

Function Reference

Builtin constant and function reference.

Macro	Description
<code>!</code>	Not operator
<code>!=</code>	Non-equality operator
<code>%</code>	Modulus operator
<code>%=</code>	Modulus assignment operator
<code>&</code>	Bitwise AND operator
<code>&&</code>	Logical AND operator
<code>&=</code>	Bitwise AND assignment operator
<code>*</code>	Multiplication operator
<code>*=</code>	Multiplication assignment operator
<code>+</code>	Addition operator
<code>++</code>	Prefix Increment
<code>+=</code>	Addition assignment operator
<code>-</code>	Subtraction operator
<code>--</code>	Prefix Decrement
<code>-=</code>	Subtraction
<code>/</code>	Division operator
<code>/=</code>	Division assignment operator
<code><</code>	Less than comparison
<code><<</code>	Left-shift operator
<code><<=</code>	Left-shift assignment operator
<code><=</code>	Less than or equal comparison
<code><></code>	Comparison operator
<code>=</code>	Assignment operator
<code>==</code>	Equality operator
<code>></code>	Greater than comparison
<code>>=</code>	Greater than or equal comparison
<code>>></code>	Right-shift operator
<code>>>=</code>	Right-shift assignment operator

Macro	Description
Signals	undocumented
^	Bitwise exclusive OR operator
^=	Bitwise exclusive OR assignment operator
_breaksw	Switch break statement
_lexicalblock	Lexical scope
_regress_op	Regression operations
_regress_replacement	Replacment regression testing
_bad_key	Command prompt unknown key callback
_chg_properties	Property change event
_default	Default extension handler
_extension	Buffer load handler
_fatal_error	Fatal condition callback
_init	Internal macro initialisation
_invalid_key	Invalid key event
_prompt_begin	Command prompt session begin callback
_prompt_end	Command prompt session end callback
_startup_complete	Startup event callback
abort	Abnormal process
above	Greater than comparison
above_eq	Greater than or equal comparison
abs	Absolute value
access	Test file accessibility
acos	Arc cosine
arg_list	Argument list
array	Declare a array symbol
asin	Arc sine
assign_to_key	Assign command to key or key sequence
atan	Arctangent
atan2	Arctangent division
atoi	Convert string to a decimal number
attach_buffer	Attach a buffer to a window
attach_syntax	Attach a syntax to a buffer
autoload	Register location of one or more macros
backspace	Delete character to the left of the cursor
basename	Return the last component of a pathname
beep	Issue a beep sound
beginning_of_line	Goto beginning of line
below	Less than comparison
below_eq	Less than or equal comparison
bless	Associate an object with a class/module
bookmark_list	Retrieve existing bookmark list
bool	Declare a boolean symbol
borders	Set window border status
break	break statement
call_registered_macro	Invoke registered macro callbacks
car	Retrieve the first list element
case	Switch case statement
<cast_float>	undocumented
<cast_int>	undocumented
catch	Catch statement
cd	Change directory
cdr	Retrieve all secondary list elements
ceil	Round up to integral value
cftime	Format time and date
change_window	Selects a new window
change_window_pos	Modify window coordinates/size
characterat	Retrieve the character value within a string
chdir	Change directory
chmod	Change mode
chown	Change owner
close_window	Close specified the window
color	Set the basic colors
color_index	Border color background color index
command_list	Retrieve list of built-in and active macros

Macro	Description
compare	Comparison
compare_files	Binary file compare
compress	Compress repeated instances of white-space characters
connect	Attach a process to a process
const	Define a variable as being constant
continue	Loop continuation
copy	Copy marked area to scrap
copy_ea_info	Copy file extended information
copy_keyboard	Copy a keyboard
copy_screen	Copy the current screen
cos	Cosine
cosh	Hyperbolic cosine
create_buffer	Create and load a buffer
create_char_map	Create a display character-map
create_dictionary	Create a dictionary
create_edge	Create an edge, splitting the window
create_menu_window	Create the menu window
create_nested_buffer	Create or reference a buffer
create_syntax	Syntax table creation
create_tiled_window	Creates a tiled window
create_window	Create a popup window
ctype	undocumented
cursor	Control cursor display
cut	Cut marked area to scrap
cvt_to_object	Convert string to object
date	Get current system date
debug	Control runtime debug and tracing
debug_support	Internal debugger functionality
declare	Declare a polymorphic symbol
define_keywords	Add keywords to a syntax dictionary
del	Delete a file
delete_block	Deleted selected region
delete_bookmark	Delete a bookmark
delete_buffer	Delete a buffer
delete_char	Delete character
delete_dictionary	Destroy a dictionary
delete_edge	Delete an edge, combining a split window
delete_line	Delete current line
delete_macro	Delete a macro from memory
delete_nth	Remove one or more elements from a list
delete_to_eol	Delete to end-of-line
delete_window	Delete a window
detach_syntax	Detach a syntax from a buffer
dialog_create	Build a dialog resource
dialog_delete	Delete a dialog resource
dialog_exit	Exit a dialog resource
dialog_run	Execute a dialog resource
dict_clear	Clear a dictionary
dict_delete	Remove a dictionary item
dict_each	Iterator a dictionary
dict_exists	Dictionary item existence check
dict_keys	Iterator dictionary keys
dict_list	Retrieve dictionary items
dict_name	Retrieve a dictionary name
dict_values	Iterator dictionary values
<diff_buffers>	undocumented
<diff_lines>	undocumented
diff_strings	Compare to strings
dirname	Report the parent directory name of a file pathname
disconnect	Disconnect a buffer from a process
display_mode	Set/retrieve display control flags
display_windows	Control window display
distance_to_indent	Calculate distance to next indent

Macro	Description
distance_to_tab	Calculate distance to next tab
do	do statement
dos	Create a sub-shell
double	Declare a double float symbol
down	Move position down one line
dprintf	Formatted diagnostics output
drop_anchor	Start marking a selection
drop_bookmark	Create or update a bookmark
echo_line	Set echo line flags
edit_file	Edit a file
edit_file2	Extended file edit
ega	Terminal line display
else	else statement
end_anchor	Set the end of the anchor
end_of_buffer	Move cursor to end of current buffer
end_of_line	Goto end of line
end_of_window	Goto end of the current window
error	Issue an error message on the command line
execute_macro	Invokes a command by name
exist	Check file existence
exit	Exit current process level
exp	Exponential function
expandpath	Expand the path
extern	Declare an external variable
fabs	Floating-point absolute value
fclose	Close a stream
feof	Test end-of-file indicator on a stream
ferror	Test error indicator on a stream
fflush	Flush a stream
file_canon	Canonicalize a path
file_glob	Return names of files that match patterns
file_match	File match utility
file_pattern	Directory file pattern
filename_match	Perform file pattern matching
filename.realpath	Return a resolved pathname
finally	Finally statement
find_file	Read next directory entry
find_file2	Extended read next directory entry
find_line_flags	Locate next line with specific flags
find_macro	Determine path of a macro object
find_marker	Locate next marker
ioctl	File miscellaneous control
first_time	Determine a macros initialisation status
firstof	Leftmost character search
float	Declare a float symbol
flock	File lock operations
floor	Round down to integral value
fmktemp	Make a unique filename
fmod	Floating-point remainder
fopen	Open a stream
for	for statement
foreach	Container iterator
format	Formatted printing
fread	Read from a stream
frexp	Extract mantissa and exponent
fseek	Reposition a file-position indicator in a stream
fstat	Stream status information
fstype	File system type
ftell	Report the file-position indicator of a stream
ftest	Test file type
<ftp_chdir>	undocumented
<ftp_close>	undocumented
<ftp_connect>	undocumented
<ftp_connection_list>	undocumented

Macro	Description
<ftp_create>	undocumented
<ftp_directory>	undocumented
<ftp_error>	undocumented
<ftp_find_connection>	undocumented
<ftp_get_file>	undocumented
<ftp_getcwd>	undocumented
<ftp_mkdir>	undocumented
<ftp_protocol>	undocumented
<ftp_put_file>	undocumented
<ftp_register>	undocumented
<ftp_remove>	undocumented
<ftp_rename>	undocumented
<ftp_set_options>	undocumented
<ftp_sitename>	undocumented
<ftp_stat>	undocumented
<ftp_timeout>	undocumented
ftruncate	Truncate the specified file
fwrite	Write to a stream
get_color	Retrieve screen colors
get_color_pair	Retrieve the specific color
get_mouse_pos	Retrieve last mouse action
get_nth	Retrieve a list element
get_parm	Retrieve the value of a macro parameter
get_property	Retrieve a dictionary item
get_region	Retrieve marked region content
<get_system_resources>	undocumented
get_term_characters	Retrieve terminal special characters
get_term_feature	Get value of a terminal feature
get_term_features	Retrieve terminate features
get_term_keyboard	Retrieve the terminal keyboard definitions
getenv	Retrieve an environment variable
geteuid	Retrieve the effective user identifier
getopt	Get options
getpid	Retrieve the process identifier
getsubopt	Parse suboption arguments from a string
getuid	Retrieve the user identifier
getwd	Get current working directory
glob	Generate pathnames matching a pattern
global	Declare a global variable
gmtime	Convert a time value to UTC time components
goto_bookmark	Seek a bookmark
goto_line	Move to a particular line
goto_old_line	Move to line before buffer modification
grief_version	GRIEF version
hilite_create	Create a hilite resource
hilite_delete	Delete a hilite resource
hilite_destroy	Destroy hilite resources
if	if statement
index	Search string for a leftmost sub-string or character
<iniclose>	undocumented
<inielexport>	undocumented
<inifirst>	undocumented
<ininext>	undocumented
<inioopen>	undocumented
<iniproperties>	undocumented
<inipush>	undocumented
<iniquery>	undocumented
<iniremove>	undocumented
input_mode	Effect of certain system keys
inq_assignment	Get key assignment for function
inq_attribute	Retrieve the current attributes
inq_backup	Get the backup creation mode
inq_backup_option	Get backup options

Macro	Description
inq_borders	Retrieve the border status
inq_brief_level	Retrieve the editor nesting level
inq_btn2_action	Retrieve the second button action
inq_buffer	Retrieve a buffer identifier
inq_buffer_flags	Retrieve buffer flags
inq_buffer_title	Retrieve a buffer title
inq_buffer_type	Retrieve buffer type
inq_byte_pos	Get current position in buffer stream
inq_called	Get the name of the calling macro
inq_char_map	Retrieve the character-map
inq_char_timeout	Get the escape delay
inq_clock	Retrieve the user identifier
inq_cmd_line	Retrieve the command line message
inq_color	Retrieve the basic colors
inq_command	Retrieve name of last keyboard command
inq_connection	Connection information
inq_ctrl_state	Retrieve the state of a window control
inq_debug	Retrieve the current debug setting
inq_dialog	Retrieve the current dialog resource
inq_display_mode	Inquire display control flags
inq_echo_format	Retrieve the echo-line format
inq_echo_line	Retrieve the echo-line flags
inq_encoding	Retrieve a buffers character encoding
inq_environment	Retrieve an environment variable
inq_feature	Retrieve an editor feature
inq_file_change	Determine state of underlying file
inq_file_magic	Retrieve the file type detection rules
inq_font	Inquire the current window fonts
inq_hilite	Retrieve a hilite definition
inq_home	Retrieve the user home directory
inq_hostname	Retrieve the local host name
inq_idle_default	Retrieve idle interval
inq_idle_time	Retrieve keyboard idle time
inq_indent	Get current indentation settings
inq_kbd_char	Peek at the keyboard
inq_kbd_flags	Get keyboard key flags
inq_kbd_name	Retrieve the assigned keyboard name
inq_keyboard	Retrieve the keyboard identifier
inq_keystroke_macro	Retrieve the current keystroke macro
inq_keystroke_status	Determine keystroke macro status
inq_line_col	Retrieve the echo_line content
inq_line_flags	Retrieve a lines associated flags
inq_line_length	Determine the longest line length
inq_lines	Retrieve the line count
inq_local_keyboard	Retrieve local keyboard identifier
inq_macro	Determine whether a macro or primitive exists
inq_macro_history	Retrieve macro execution history
inq_margins	Retrieve buffer formatting margins
inq_mark_size	Retrieve size of marked region
inq_marked	Determine the current marked region
inq_marked_size	Size the marked region
inq_message	Retrieve the prompt line
inq_mode	Returns the overstrike mode
inq_modified	Determine a buffers modification status
inq_module	Retrieve the current module
inq_mouse_action	Retrieve the keyboard mouse handler
inq_mouse_type	Retrieve the button type
inq_msg_level	Get the message level
inq_names	Retrieve associated buffer names
inq_position	Retrieve current buffer position
inq_process_position	Get position of process buffer
inq_profile	Retrieve profile directory
inq_prompt	Retrieve the prompt status

Macro	Description
inq_remember_buffer	Determine the keystroke buffer name
inq_ruler	Retrieves the ruler specification
inq_scrap	Obtain the scrap buffer identifier
inq_screen_size	Determine screen dimensions
inq_symbol	Determine if the symbol exists
inq_syntax	Retrieve the syntax identifier
inq_system	Determine if buffer is a system buffer
inq_tab	Derive the tab increment
inq_tabs	Retrieves the buffer tab specification
inq_terminator	Retrieve a buffers line terminator
inq_time	Retrieve the last modification time
inq_tmpdir	Get the temporary-file directory
inq_top_left	Retrieve window view port coordinates
<inq_unicode_version>	undocumented
inq_username	Retrieve the user name
inq_vfs_mounts	Retrieve list of mounts
inq_views	Determine window count
inq_window	Retrieve the current window
inq_window_buf	Retrieve the associated buffer
inq_window_color	Retrieve the window attribute
inq_window_flags	Retrieve window flags
inq_window_info	Retrieve the current window information
inq_window_infox	Retrieve information about a window
inq_window_priority	Retrieve the windows display priority
inq_window_size	Retrieve the window size
insert	Insert string into current buffer
insert_buffer	Insert format text into a buffer
insert_mode	Set the buffer insert/overstrike mode
insert_process	Send string to a attached process
insertf	Insert a formatted string
<inside_region>	undocumented
int	Declare an integer symbol
int_to_key	Convert an keycode to mnemonic key string
is_array	Determine whether an array type
is_float	Determine whether a float type
is_integer	Determine whether an integer type
is_list	Determine whether a list type
is_null	Determine whether a NULL type
is_string	Determine whether a string type
is_type	Determine whether an explicit type
isalnum	Alphanumeric character predicate
isalpha	Alpha character predicate
isascii	ASCII character predicate
isblank	Blank character predicate
isclose	Test for floating point equality
iscntrl	Control character predicate
iscsym	A symbol character predicate
isdigit	Numeric character predicate
isfinite	Test for finite value
isgold	Alphanumeric character predicate
isgraph	Graphic character predicate
isinf	Test for infinity
islower	Lowercase character predicate
isnan	Test for a NaN
isprint	A printable character predicate
ispunct	Punctuation character predicate
isspace	Space character predicate
isupper	Uppercase character predicate
isword	Word character predicate
isxdigit	Hexadecimal character predicate
itoa	Convert an integer into a string
key_list	Retrieve keyboard bindings
key_to_int	Convert key name to a code
keyboard_flush	Flush the keyboard buffer

Macro	Description
keyboard_pop	Pop a keyboard from the keyboard stack
keyboard_push	Push a keyboard onto the keyboard stack
keyboard_typeables	Assign self_insert to all typeable keys
lastof	Rightmost character search
lsexp	Multiply by a power of two
left	Move position left one character
length_of_list	Determine list element count
link	Link a file
list	Declare a list symbol
list_each	Iterator through the list elements
list_extract	Extract list elements
list_of_dictionaries	List of created dictionaries
list_reset	Reset the list iterator
load_keystroke_macro	Load a recorded macro from a file
load_macro	Load a macro object
localtime	Convert a time value to local time components
log	Natural logarithm
log10	Base 10 logarithm function
lower	Convert string or character to lowercase
lstat	Get symbolic link status
ltrim	Chomp characters from the front of a string
macro	Define a macro body
macro_list	Retrieve list of current macros
main	Macro entry point
make_list	Build an evaluated list
make_local_variable	Make a buffer local variable
mark	Toggle the anchor status
mark_line	Create a line marker
message	Display a message on the command line
mkdir	Create a directory
mktemp	Make a temporary filename
mode_string	Conversion stat mode to a string representation
modf	Decompose a floating-point number
module	Assign a module identifier
move_abs	Move to an absolute location in the buffer
move_edge	Modify a window
move_rel	Move to a relative location in the buffer
next_buffer	Identifier of the next buffer
next_char	Move to the next character
next_window	Obtain the next window identifier
nothing	Noop
nth	Extract the indexed list item
output_file	Change the output file-name
page_down	Move position down a page
page_up	Move position up a page
parse_filename	Parse a file into its components
paste	Insert scrap buffer at cursor location
pause	Pause keystroke definition
pause_on_error	Set the error display mode
pause_on_message	Set the message display mode
perror	Print error
playback	Replay a keystroke macro
pop	Pop the last element
post++	Postfix Increment
post--	Postfix Decrement
pow	Raise to power
prev_char	Move to the previous character
previous_buffer	Identifier of the previous buffer
print	Print formatted string to stdout
printf	Print formatted string to stdout
process	Invoke a Grief engine
process_mouse	Process mouse event
profile	Profiling support

Macro	Description
push	Add an element onto a list
push_back	Push back a character into the keyboard
put_nth	Modify a list element
put_parm	Assign an argument value
putenv	Set an environment variable
quote_list	Build an unevaluated list
quote_regexp	Quote regexp special characters
raise_anchor	Raise the last dropped mark
rand	Generate a random number
re_comp	Compile a regular expression
re_delete	Delete a compiled expression
re_result	Retrieve search captures
re_search	Search for a string
re_syntax	Set the regular expression mode
re_translate	Search and replace
read	Read characters from the buffer
read_char	Read next key from the keyboard
read_ea	Read file extended information
read_file	Read into the current buffer
readlink	Read the contents of a symbolic link
realpath	Resolve a pathname
redo	Redo an undo operation
redraw	Redraw screen
ref_parm	Create a reference parameter
refresh	Update the display
register	Define a variable as being a register
register_macro	Register a callback procedure
reload_buffer	Reload the buffer content
remember	Start remembering keystrokes
remove	Remove a file
rename	Rename a file
replacement	Overload an existing macro definition
require	Enforce the use of an external module
reregister_macro	Register a unique callback procedure
restore_position	Restore a previously saved position
return	Return from a macro
returns	Return an expression from a macro
right	Move position right one character
rindex	Search string for a rightmost sub-string or character
rmdir	Remove directory
rtrim	Chomp characters from the end of a string
save_keystroke_macro	Save the current keystroke macro
save_position	Saves current cursor/buffer state
screen_dump	Dump an image of the screen
search_back	Backwards buffer search
search_case	Set the search pattern case mode
search_fwd	Buffer search
search_list	Search list contents
search_string	Searches for a pattern in a string
searchpath	Searches for a given file in a specified path
self_insert	Insert a character as if it was typed
send_signal	Send signal to a process buffer
set_attribute	Set the color attributes
set_backup	Set backup creation mode
set_backup_option	Set backup options
set_binary_size	Set binary chunk size
set_btn2_action	Set the second button action
set_buffer	Set the current buffer
set_buffer_cmap	Set a buffers character-map
set_buffer_flags	Set buffer flags
set_buffer_title	Set a buffers title
set_buffer_type	Set the buffer storage type
set_calling_name	Set the name of the calling macro

Macro	Description
set_char_timeout	Set the escape delay
set_color	Set screen colors
set_color_pair	Set a specific color
set_ctrl_state	Set the state of a window control
set_ea	Set file extended information
set_echo_format	Set the echo line format
set_encoding	Set a buffers character encoding
set_feature	Config an editor feature
set_file_magic	Define the file type detection rules
set_font	Set the current window fonts
set_idle_default	Set idle interval
set_indent	Set the buffers default indentation
set_kbd_name	Set the keyboard name
set_line_flags	Associate line flags
set_macro_history	Set the macro execution history
set_margins	Set buffer formatting margins
set_mouse_action	Set keyboard mouse handler
set_mouse_type	Sets the mouse type
set_msg_level	Set level of informational messages
set_process_position	Set process insertion position
set_property	Set a dictionary item
set_ruler	Configure the buffer ruler
set_scrap_info	Set the scrap buffer details
set_syntax_flags	Set syntax flags
set_tab	Derive the buffer tab stops
set_term_characters	Set terminal special characters
set_term_feature	Set a terminal attribute
set_term_features	Define terminal attributes
set_term_keyboard	Define terminal keyboard definitions
set_terminator	Set a buffers line terminator
set_top_left	Manages window view port coordinates
<set_unicode_version>	undocumented
set_window	Set the active window
set_window_cmap	Set a windows character-map
set_window_flags	Set window flags
set_window_priority	Set the window display priority
set_wm_name	Set the window and/or icon name
shell	Spawn a sub-shell process
shift	Shift the first list element
sin	Sine function
sinh	Hyperbolic sine function
sleep	Suspend execution for an interval of time
sort_buffer	Sort buffer content
sort_list	Sort list
spell_buffer	Spell the specified buffer
spell_control	Spell control
spell_dictionary	Spell dictionary modifications
spell_distance	Edit distance
spell_string	Spell the specified word or line
spell_suggest	Suggest spelling of the the specified word
splice	Splice a list, removing and/or adding elements
split	Split a string into tokens
split_arguments	Argument split
sprintf	Formatted printing to a string
sqrt	Square root function
rand	Seed the random number generator
sscanf	Read formatted data from string
stat	Obtain file information
static	Define a function or module scope
strcasecmp	String case insensitive compare
strcasestr	Locate first occurrence of a case insensitive
strcmp	String compare
strerror	String error
strfilecmp	Filename comparison

Macro	Description
strftime	Format time and date
string	Declare a string symbol
string_count	Count occurrences of characters in a string
strlen	String length
strnlen	String length limited to an explicit maximum
strpbrk	Search a string for any of a set of characters
strupr	Pop the leading character(s)
strrstr	Locate last occurrence of a sub-string
strsignal	Return string describing signal
strstr	Locate first occurrence of a sub-string
strtod	String to double
strtof	String to float
strtol	Convert a string into its numeric value
strverscmp	Version string compare
substr	Extract a sub-string
suspend	Suspend current process
swap_anchor	Swaps the mark with the current position
switch	Switch statement
symlink	Create a symbolic link
syntax_build	Build a syntax hilitng engine
syntax_column_ruler	Column syntax coloriser
<syntax_find>	undocumented
syntax_rule	Define a syntax hilite rule
syntax_token	Define a syntax token
tabs	Set buffer tab stops
tagdb_close	Tag database close
tagdb_open	Tag database open
tagdb_search	Tag database search
tan	Tangent function
tanh	Hyperbolic tangent function
throw	Throw an exception
time	Get the current system time
tokenize	Tokenize a string into token elements
top_of_buffer	Move cursor to start of current buffer
top_of_window	Goto top of the current window
transfer	Buffer to buffer transfer
translate	Buffer search and replace
translate_pos	Convert window coordinates
trim	Chomp characters from a string
try	Try statement
typeof	Determine the symbol type
umask	Set and get the file mode creation mask
uname	Retrieve system information
undo	Undo previous edit operations
unlink	Unlink a file
unregister_macro	Remove a registered macro
unshift	Shift the first list element
up	Move position up one line
upper	Convert string or character to uppercase
use_local_keyboard	Associate a keyboard with a buffer
use_tab_char	Configure use of hard/soft tabs
version	Version information
vfs_mount	Mount a virtual file-system
vfs_unmount	Unmount a virtual file-system
view_screen	View the content of underlying screen
wait	Wait for attached process to terminate
wait_for	Wait for process output
watch	Watch a symbol
<wcharacterat>	undocumented
<wcwidth>	undocumented
<wfirstof>	undocumented
while	while statement
widget_get	Retrieve a widget attribute

Macro	Description
<code>widget_set</code>	Set a widget attribute
<code><windex></code>	undocumented
<code>window_color</code>	Set the window attribute
<code><wlastof></code>	undocumented
<code><wlower></code>	undocumented
<code><windex></code>	undocumented
<code>write_block</code>	Write selected region
<code>write_buffer</code>	Write to buffer content
<code><wstrcasecmp></code>	undocumented
<code><wstrcmp></code>	undocumented
<code><wstrlen></code>	undocumented
<code><wstrnlen></code>	undocumented
<code><wstrpbrk></code>	undocumented
<code><w strrstr></code>	undocumented
<code><wstrstr></code>	undocumented
<code><wsubstr></code>	undocumented
<code><wupper></code>	undocumented
<code> </code>	Bitwise OR operator
<code> =</code>	Bitwise OR assignment operator
<code> </code>	Logical OR operator
<code>~</code>	Bitwise complement

Macro Primitives

Constant	Description
<code>BPACKAGES</code>	BRIEF packages default
<code>GRBACKUP</code>	Backup path
<code>GRDICTIONARIES</code>	Dictionary locales
<code>GRDICTIONARY</code>	Dictionary search path
<code>GRFILE</code>	Default empty file name
<code>GRFLAGS</code>	Default command line arguments
<code>GRHELP</code>	Help search path
<code>GRINIT_FILE</code>	GRIEF initialisation name
<code>GRKBDPATH</code>	Keyboard library search path
<code>GRLEVEL</code>	GRIEF Nesting level
<code>GRLOG_FILE</code>	GRIEF diagnostics log file name
<code>GRPATH</code>	Macro object search path
<code>GRPROFILE</code>	Profile directory override
<code>GRRESTORE_FILE</code>	GRIEF restore file name
<code>GRSTATE_DB</code>	GRIEF state database name
<code>GRSTATE_FILE</code>	GRIEF state file name
<code>GRTEMPLATE</code>	Source template search path
<code>GRTERM</code>	Terminal override
<code>GRTERMCAP</code>	Terminal capability database
<code>GRTMP</code>	Temporary dictionary
<code>GRVERSIONMAJOR</code>	GRIEF major version
<code>GRVERSIONMINOR</code>	GRIEF minor version
<code>GRVERSIONS</code>	Backup versions
<code>errno</code>	Last system errno number

Macro Constants

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Arithmetic Operators

Summary

Arithmetic Operators

Macros

above	Greater than comparison.
above_eq	Greater than or equal comparison.
abs	Absolute value.
acos	Arc cosine.
asin	Arc sine.
atan	Arctangent.
atan2	Arctangent division.
below	Less than comparison.
below_eq	Less than or equal comparison.
ceil	Round up to integral value.
compare	Comparison.
cos	Cosine.
cosh	Hyperbolic cosine.
cvt_to_object	Convert string to object.
exp	Exponential function.
fabs	Floating-point absolute value.
floor	Round down to integral value.
fmod	Floating-point remainder.
frexp	Extract mantissa and exponent.
isclose	Test for floating point equality.
isfinite	Test for finite value.
isinf	Test for infinity.
isnan	Test for a NaN.
ldexp	Multiply by a power of two.
log	Natural logarithm.
log10	Base 10 logarithm function.
modf	Decompose a floating-point number.
pow	Raise to power.
sin	Sine function.
sinh	Hyperbolic sine function.
sqrt	Square root function.
tan	Tangent function.
tanh	Hyperbolic tangent function.

Functions

!	Not operator.
!=	Non-equality operator.
%	Modulus operator.
%=	Modulus assignment operator.
&	Bitwise AND operator.
&&	Logical AND operator.
&=	Bitwise AND assignment operator.
*	Multiplication operator.
*=	Multiplication assignment operator.
+	Addition operator.
++	Prefix Increment.
+=	Addition assignment operator.
-	Subtraction operator.
--	Prefix Decrement.
-=	Subtraction.
/	Division operator.
/=	Division assignment operator.
<	Less than comparison.
<<	Left-shift operator.
<<=	Left-shift assignment operator.
<=	Less than or equal comparison.
<=>	Comparison operator.
=	Assignment operator.
==	Equality operator.
>	Greater than comparison.
>=	Greater than or equal comparison.
>>	Right-shift operator.
>>=	Right-shift assignment operator.
^	Bitwise exclusive OR operator.
^=	Bitwise exclusive OR assignment operator.
post++	Postfix Increment.

<code>post--</code>	Postfix Decrement.
<code> </code>	Bitwise OR operator.
<code> =</code>	Bitwise OR assignment operator.
<code> </code>	Logical OR operator.
<code>~</code>	Bitwise complement.

Macros

above

```
int above(declare expr1,
          declare expr2 )
```

Greater than comparison.

Description

The `above()` primitive is the functional form of the `>` operator.

The function yields the comparison of the two arguments `expr1` and `expr2` which both must be of equivalent types, either numeric or string; which are arithmetic and lexicographically comparisons respectively.

Parameters

`expr1` Left expression, numeric or string.
`expr2` Right expression, numeric or string.

Returns

The `above()` primitive yields the value 1 if `expr1` is greater then `expr2`, and 0 if the relation is false.

Limitations

Lists can not be compared.

See Also

[Operators, >](#)

above_eq

```
int above_eq(declare expr1,
              declare expr2 )
```

Greater than or equal comparison.

Description

The `above_eq()` primitive is the functional form of the `>=` operator.

The function yields the comparison of the two arguments `expr1` and `expr2` which both must be of equivalent types, either numeric or string; which are arithmetic and lexicographically comparisons respectively.

Parameters

`expr1` Left expression, numeric or string.
`expr2` Right expression, numeric or string.

Returns

The `above_eq()` primitive yields the value 1 if `expr1` is greater then or equals `expr2`, and 0 if the relation is false.

See Also

[Operators, >=](#)

abs

```
int abs(int val)
```

Absolute value.

Description

The `abs()` primitive computes the absolute value of an integer.

Returns

Returns the absolute value.

See Also

[fabs](#)

acos

```
float acos(float x)
```

Arc cosine.

Description

The *acos()* primitive shall compute the principal value of the arc cosine of the argument *x*. The value of *x* should be in the range [-1,1].

Returns

The *acos()* primitive on successful completion shall return the arc cosine of *x*, in the range [0,pi] radians.

Portability

Result on error are system dependent.

See Also

[cos](#), [isnan](#)

asin

```
float asin(float x)
```

Arc sine.

Description

The *asin()* primitive shall compute the principal value of the arc sine of the argument *x*. The value of *x* should be in the range [-1,1].

Returns

Upon successful completion shall return the arc sine of *x*, in the range [-pi/2,pi/2] radians.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[isnan](#), [sin](#)

atan

```
float atan(float x)
```

Arctangent.

Description

The *atan()* primitive calculates the arctangent of *x*.

Returns

Returns a value in the range -pi/2 to pi/2 radians.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[atan2](#)

atan2

```
float atan2(float y,
           float x )
```

Arctangent division.

Description

The *atan2()* primitive calculate the arctangent (inverse tangent) of the operand division *y/x*.

Returns

The *atan2()* primitive returns a value in the range -pi to pi radians. If both arguments of *atan2()* are zero, the function sets *errno* to EDOM, and returns 0. If the correct value would cause underflow, zero is returned and the value ERANGE is stored in *errno*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[atan](#)

below

```
int below(declare expr1,
          declare expr2 )
```

Less than comparison.

Description

The *below()* primitive is the functional form of the < operator.

The function yields the comparison of the two arguments *expr1* and *expr2* which both must be of equivalent types, either numeric or string; which are arithmetic and lexicographically comparisons respectively.

Parameters

expr1 Left expression, numeric or string.
expr2 Right expression, numeric or string.

Returns

The *below()* primitive yields the value 1 if *expr1* is less than *expr2*, and 0 if the relation is false.

Limitations

Lists can not be compared.

See Also

[Operators](#), <

below_eq

```
int below_eq(declare expr1,
              declare expr2 )
```

Less than or equal comparison.

Description

The *below_eq()* primitive is the functional form of the <= operator.

The function yields the comparison of the two arguments *expr1* and *expr2* which both must be of equivalent types, either numeric or string; which are arithmetic and lexicographically comparisons respectively.

Parameters

expr1 Left expression, numeric or string.
expr2 Right expression, numeric or string.

Returns

The *below_eq()* primitive yields the value 1 if *expr1* is less then or equals *expr2*, and 0 if the relation is false.

Limitations

Lists can not be compared.

See Also

[Operators, >=](#)

ceil

```
float ceil(float x)
```

Round up to integral value.

Description

The `ceil()` primitive computes the smallest integer that is greater than or equal to x .

Returns

Returns the calculated value as a double, float, or long double value.

If there is an overflow, the function sets `errno` to `ERANGE` and returns `HUGE_VAL`.

Portability

Result on error are system dependent; where supported `errno` shall be set to a non-zero manifest constant describing the error.

See Also

[floor](#), [fabs](#)

compare

```
int compare(expr1,  
           expr2 )
```

Comparison.

Description

The `compare()` primitive is the functional form of the `<=>` operator.

The function yields the comparison of the two arguments `expr1` and `expr2` which both must be of equivalent types, either numeric or string; which are arithmetic and lexicographically comparisons respectively.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The `compare()` primitive yields the value -1 if the first expression is less than the second, 0 if they equals, and 1 if the greater than; which are arithmetic and lexicographically comparisons respectively.

Limitations

Lists can not be compared.

See Also

[Operators](#), [declare](#)

cos

```
float cos(float x)
```

Cosine.

Description

The `cos()` primitive calculates the cosine of x . The value x is expressed in radians.

Returns

Returns the calculated value.

If x is outside prescribed limits, the value is not calculated. Instead, the function returns 0 and sets the `errno` to `ERANGE`.

If the correct value would cause an underflow, zero is returned and the value `ERANGE` is stored in `errno`.

Portability

Result on error are system dependent; where supported `errno` shall be set to a non-zero manifest constant describing the error.

See Also[sin](#), [atan](#), [atan2](#)**cosh**

```
float cosh(float x)
```

Hyperbolic cosine.

Description

The `cosh()` primitive calculates the hyperbolic cosine of *x*. The value *x* is expressed in radians.

Returns

Returns the calculated value.

If the result overflows, the function returns `+HUGE_VAL` and sets *errno* to `ERANGE`.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also[cos](#), [sin](#)**cvt_to_object**

```
declare cvt_to_object(string value,
                      [int &length])
```

Convert string to object.

Description

The `cvt_to_object()` primitive converts the initial portion of the string specified string *value* into an underlying `int`, `float` or `string` object.

Firstly, the input string is divided into three parts

- An initial, possibly empty, sequence of white-space characters.
- Either:
 - a) A subject sequence interpreted as string of characters enclosed by either single('') or double("") quotes.
 - b) A subject sequence interpreted as a integer constant represented in some radix determined by the value of base.
 - c) A subject sequence interpreted as a floating-point constant or representing *infinity* or *Nan*.
- A final string of one or more unrecognised characters, including the terminating null byte of the input string.

Then they shall attempt to convert the subject sequence to either a string, integer-constant or float-point constant, and return the result.

Integer Values

If the value of base is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a + or - sign.

A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits.

An octal constant consists of the prefix *O* optionally followed by a sequence of the digits *O* to *7* only.

A binary constant consists of the prefix *Ob* or *OB* followed by a sequence of the decimal digits *O* or *1*.

A hexadecimal constant consists of the prefix *0x* or *0X* followed by a sequence of the decimal digits *a* (or *A*) to *f* (or *F*) with values 10 to 15 respectively.

Floating Point Values

A floating-point number should have the form "SI.FESX" containing at least a fractional value or exponent, where

- **S** is the sign; may be "+", "-", or omitted.
- **I** is the integer part of the mantissa.
- **F** is the fractional part of the mantissa prefixed with a dot.
- **X** is the exponent.

Either *I* or *F* may be omitted, or both. The decimal point isn't necessary unless *F* is present.

A character sequence *INF* or *INFINITY*, ignoring case, shall be interpreted as an infinity; if representable.

A character sequence *NAN* shall be interpreted as a quiet NaN, ignoring case, shall be interpreted as an Quiet Not-A-Number; if representable.

Example

```

int
parsetoken(string line)
{
    declare x;
    int len;

    x = cvt_to_object(line, len);
    switch (typeof(x)) {
        case "integer": // integer constant.
            break;
        case "float": // floating point constant.
            break;
        case "string": // string constant.
            break;
        default:
            // error condition
            break;
    }
    return len;
}

```

Parameters

value String value to be parsed.

length Optional integer variable reference, if specified shall be populated with the number of characters consumed within the source string, including all consumed leading characters, for example white-space. Upon a parse error the populated length shall be 0.

Returns

The `cvt_to_object()` primitive returns the value of the parsed object converted to the most suitable type, either as an `int`, `float` or `string` type.

Otherwise if the string cannot be parsed NULL is returned.

Portability

n/a

See Also

`atoi`, `strtod`

exp

```
float exp(float x)
```

Exponential function.

Description

The `exp()` primitive calculates the exponent of `x`, defined as e^{**x} , where `e` equals `2.17128128....`

Returns

If successful, the function returns the calculated value.

If an overflow occurs, the function returns **HUGE_VAL**. If an underflow occurs, it returns `0`. Both overflow and underflow set `errno` to **ERANGE**.

Portability

Result on error are system dependent; where supported `errno` shall be set to a non-zero manifest constant describing the error.

See Also

`frexp`, `log`, `log10`

fabs

```
float fabs(float x)
```

Floating-point absolute value.

Description

The *fabs()* primitive calculates the absolute value of a floating-point argument *x*.

Returns

Returns the absolute value of the float input.

Portability

n/a

See Also

[ceil](#), [floor](#), [abs](#)

floor

```
float floor(float x)
```

Round down to integral value.

Description

The *floor()* primitive calculates the largest integer that is less than or equal to *x*.

Returns

Returns the calculated integral value expressed as a double, float, or long double value. The result cannot have a range error.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[ceil](#), [fmod](#), [modf](#), [fabs](#)

fmod

```
float fmod(float x,
           float y )
```

Floating-point remainder.

Description

The *fmod()* primitive Calculates the floating-point remainder of *x/y*. The absolute value of the result is always less than the absolute value of *y*. The result will have the same sign as *x*.

Returns

If successful, the function returns the floating-point remainder of *x/y*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

Dependent on the implementation if *y* is 0 one of two may occur,

- *fmod* returns 0;
- the function sets *errno* to **EDOM** and returns NaN.

No other errors will occur.

See Also

[ceil](#), [floor](#), [modf](#)

frexp

```
float frexp(float num,
            int   &exp )
```

Extract mantissa and exponent.

Description

The *frexp()* primitive breaks a floating-point number *num* into a normalised fraction and an integral power of 2. It stores the integer exponent in the int object *exp*.

An application wishing to check for error situations should set *errno* to 0 before calling *frexp()*. If *errno* is non-zero

on return, or the return value is NaN, an error has occurred.

Returns

The `frexp()` primitive returns the value x , such that x is a double with magnitude in the interval [0.5, 1) or 0, and `num` equals x times 2 raised to the power `exp`.

If `num` is 0, both parts of the result are 0.

Portability

Result on error are system dependent; where supported `errno` shall be set to a non-zero manifest constant describing the error.

See Also

`isnan`, `ldexp`, `modf`

isclose

```
int isclose(float v1,
           float v2,
           float ~rel_tol,
           float ~abs_tol )
```

Test for floating point equality.

Description

The `isclose()` primitive determines whether two floating point numbers are close in value.

Parameters

<code>v1</code>	First string.
<code>v2</code>	Second value to compare against.
<code>rel_tol</code>	Optional value, used for relative tolerance.
<code>abs_tol</code>	Optional value, used for the minimum absolute tolerance.

Returns

The `isclose()` primitive returns a non-zero value if `v1` is close in value to `v2`, otherwise zero or -1 on error.

Portability

A **GriefEdit** extension.

Result on error are system dependent; where supported `errno` shall be set to a non-zero manifest constant describing the error.

isfinite

```
int isfinite(float val)
```

Test for finite value.

Description

The `isfinite()` primitive shall determine whether its argument has a finite value (zero, subnormal, or normal, and not infinite or NaN).

Returns

The `isfinite()` primitive shall return a non-zero value if and only if its argument has a finite value.

Portability

A **GriefEdit** extension.

isinf

```
int isinf(float val)
```

Test for infinity.

Description

The `isinf()` primitive shall determine whether its argument has a finite value (zero, subnormal, or normal, and not infinite or NaN).

Returns

The *isnan()* primitive shall return a non-zero value if and only if its argument has an infinite value.

Portability

A **GriefEdit** extension.

isnan

```
int isnan(float val)
```

Test for a NaN.

Description

The *isnan()* primitive tests the specified floating-point value *val*, returning a nonzero value if *val* is a not a number (NaN).

A **NaN** is generated when the result of a floating-point operation cannot be represented in Institute of Electrical and Electronics Engineers (IEEE) format.

On systems not supporting NaN values, *isnan()* always returns 0.

Returns

Non-zero if **NaN**, otherwise 0.

Portability

A **GriefEdit** extension.

ldexp

```
float ldexp(float val,
           int   exp )
```

Multiply by a power of two.

Description

The *ldexp()* primitive calculates the value of $x \cdot (2^{\text{exp}})$.

This is *x* multiplied by 2 to the power of *exp*.

Returns

Returns the calculated value.

Otherwise, if the correct calculated value is outside the range of representable values, **-/+HUGE_VAL** is returned, according to the sign of the value. The value **ERANGE** is stored in *errno* to indicate that the result was out of range.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[modf](#)

log

```
float log(float x)
```

Natural logarithm.

Description

The *log()* primitive calculates the natural logarithm (base e) of *x*, for *x* greater than 0.

Returns

Returns the computed value.

If *x* is negative, the function sets *errno* to **EDOM** and returns **-HUGE_VAL**. If *x* is 0.0, the function returns **-HUGE_VAL** and sets *errno* to **ERANGE**.

If the correct value would cause an underflow, 0 is returned and the value **ERANGE** is stored in *errno*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also[log10](#), [exp](#)**log10**

```
float log10(float val)
```

Base 10 logarithm function.

Description

The *log10()* primitive calculates the base 10 logarithm of the positive value of *x*.

Returns

Returns the computed value.

If *x* is negative, the function sets *errno* to **EDOM** and returns **-HUGE_VAL**. If *x* is 0.0, the function returns **-HUGE_VAL** and sets *errno* to **ERANGE**.

If the correct value would cause an underflow, 0 is returned and the value **ERANGE** is stored in *errno*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also[log](#), [pow](#)**modf**

```
float modf(float num,
           float &mod )
```

Decompose a floating-point number.

Description

The *modf()* primitive breaks the argument *num* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a double in the object *mod*.

An application wishing to check for error situations should set *errno* to 0 before calling *modf()*. If *errno* is non-zero on return, or the return value is NaN, an error has occurred.

Returns

Upon successful completion, *modf()* returns the signed fractional part of *num*.

If *num* is NaN, NaN is returned, *errno* may be set to **EDOM**.

If the correct value would cause underflow, 0 is returned and *errno* may be set to **ERANGE**.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also[frexp](#), [isnan](#), [ldexp](#)**pow**

```
float pow(float x,
          float y )
```

Raise to power.

Description

The *pow()* primitive returns *x* raised to the power of *y*.

Returns

Returns the computed value.

If *y* is 0, the function returns 1.

If *x* is negative and *y* is non-integral, the function sets *errno* to **EDOM** and returns **-HUGE_VAL**. If the correct value is outside the range of representable values, **+HUGE_VAL** is returned according to the sign of the value, and

the value of **ERANGE** is stored in *errno*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[exp](#), [log](#)

sin

```
float sin(float val)
```

Sine function.

Description

The *sin()* primitive computes the sine of the argument 'val', measured in radians.

Returns

Upon successful completion, shall return the sine of *x*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[asin](#), [isnan](#)

sinh

```
float sinh(float val)
```

Hyperbolic sine function.

Description

The *sinh()* primitive computes the hyperbolic sine of the argument *val*.

Returns

Upon successful completion, shall return the hyperbolic sine of *x*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[sin](#)

sqrt

```
float sqrt(float val)
```

Square root function.

Description

The *sqrt()* primitive computes the square root of the argument *val*.

Returns

Upon successful completion, shall return the square root of *val*.

Portability

Result on error are system dependent; where supported *errno* shall be set to a non-zero manifest constant describing the error.

See Also

[isnan](#)

tan

```
float tan(float val)
```

Tangent function.

Description

The `tan()` primitive computes the tangent of the argument `val`, measured in radians.

Returns

Upon successful completion, shall return the tangent of `val`.

Portability

Result on error are system dependent; where supported `errno` shall be set to a non-zero manifest constant describing the error.

See Also

[atan](#), [isnan](#)

tanh

```
float tanh(float val)
```

Hyperbolic tangent function.

Description

The `tanh()` primitive computes the hyperbolic tangent of the argument `val`.

Returns

Upon successful completion, shall return the hyperbolic tangent of `x`.

Portability

Result on error are system dependent; where supported `errno` shall be set to a non-zero manifest constant describing the error.

See Also

[tan](#)

Functions

!

```
!expr
```

Not operator.

Description

The `!` operator yields the logical not of the expression `expr`.

If the operand has the value zero, then the result value is 1.

If the operand has some other value, then the result is 0.

Parameters

`expr` Expression.

Returns

Returns the logical not of the expression `expr`. `expr` must evaluate to an integer expression.

See Also

[Operators](#), [declare](#)

!=

```
expr1 != expr2
```

Non-equality operator.

Description

The `!=` operator compares for inequality, yielding the value `1` if the relation is true, and `0` if the relation is false. The result type is *int*.

Parameters

- `expr1` Left expression.
- `expr2` Right expression.

Returns

The accumulator shall be set to the result of the comparison.

Limitations

Lists can not be compared.

See Also

[Operators](#), [declare](#)

%

```
expr1 % expr2
```

Modulus operator.

Description

The `%` operator yields the remainder from the division of the first operand by the second operand. The operands of must have numeric types.

Parameters

- `expr1` Left expression.
- `expr2` Right expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

%=

```
var %= expr1
```

Modulus assignment operator.

Description

The `%=` operator takes the modulus of the first operand `var` specified by the value of the second operand `expr1`, storing the result in the object specified by the first operand `var`.

Parameters

- `var` A numeric scalar type.
- `expr1` A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

&

```
expr1 & expr2
```

Bitwise AND operator.

Description

The `&` operator yields the result is the **bitwise AND** of the two operands `expr1` and `expr2`. That is, the bit in the result is set if and only if each of the corresponding bits in the operands are set.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

&&

```
expr1 && expr2
```

Logical AND operator.

Description

The `&&` operator performs a logic AND between the two expressions `expr1` and `expr2`.

Each of the operands must have scalar type. If both of the operands are not equal to zero, then the result is 1. Otherwise, the result is zero. The result type is int.

Short Circuit Evaluation

Logical operators are executed using *short-circuit* semantics whereby the second argument is only executed or evaluated if the first argument does not suffice to determine the value of the expression:

- Logical ADD - If the first operand is zero, then the second operand is not evaluated. Any side effects that would have happened if the second operand had been executed do not happen. Any function calls encountered in the second operand do not take place.
- Logical OR - If the first operand is not zero, then the second operand is not evaluated. Any side effects that would have happened if the second operand had been executed do not happen. Any function calls encountered in the second operand shall not take place.

See Also

[Logical Operators](#)

&=

```
var &= expr1
```

Bitwise AND assignment operator.

Description

The `&=` operator obtains the bitwise AND of the first operand `var` and second operand `expr1`, storing the result in the object specified by the first operand `var`.

Parameters

`var` A numeric scalar type.
`expr1` A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

```
expr1 * expr2
```

Multiplication operator.

Description

The `*` operator yields the product of its operands `expr1` and `expr2`. The operands must have numeric types.

Parameters

`expr1` Left numeric expression.
`expr2` Right numeric expression.

Returns

The accumulator shall be set to the result of the computation.

Notes

If either number is NaN, the result is NaN. Multiplication of Infinity by zero gives a result of NaN, while multiplying Infinity by any non-zero number gives a result of Infinity.

See Also

[Operators](#), [declare](#), [isnan](#), [isfinite](#)

***=**

```
var *= expr1
```

Multiplication assignment operator.

Description

The *= operator multiplies the value of the first operand *var* by the value of the second operand *expr1*; store the result in the object specified by the first operand *var*.

Parameters

var Any scalar type.

expr1 A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

+

```
expr1 + expr2
```

Addition operator.

Description

The + operator yields the sum of its operands resulting from the addition of the first operand with the second.

If *expr1* or *expr2* is a list, then a new list is returned which is the concatenation of *expr1* and *expr2*.

If *expr1* or *expr2* is a string, then the other operand is converted to a string, and a string is returned.

If either operand is a floating point number then the other the result is the sum of the two expressions.

Otherwise the integer value of the sum of the two expressions is returned.

Parameters

expr1 Left expression.

expr2 Right expression.

Returns

The accumulator shall be set to the result of the computation.

A list value if either operand is a list; a string if either operand is a string; a float if either operand is a float; otherwise an integer value.

See Also

[Operators](#), [declare](#)

++

```
++expr
```

Prefix Increment.

Description

The ++ operator increases the operand *expr* by 1, with the result of the operation returned.

Parameters

`expr` An expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

+=

```
var += expr1
```

Addition assignment operator.

Description

The `+=` operator adds the value of the second operand to the value of the first operand `var`, storing the result in the object specified by the first operand `var`.

Parameters

`var` Any scalar type.

`expr1` An expression resulting in value of the same type as the scalar type `var`.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

-

```
expr1 - expr2
```

Subtraction operator.

Description

The `-` operator yields the difference resulting from the subtraction of the second operand from the first.

Parameters

`expr1` Left expression.

`expr2` Right expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

--

```
--expr
```

Prefix Decrement.

Description

The `--` operator decreases the operand `expr` by 1, with the result of the operation returned.

Parameters

`expr` An expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

- =

```
var -= expr1
```

Subtraction.

Description

The `-=` operator subtracts the value of the second operand *expr1* from the value of the first operand *var*, storing the result in the object specified by the first operand *var*.

Parameters

var A numeric scalar type.

expr1 A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

/

```
expr1 / expr2
```

Division operator.

Description

The `/` operator yields the quotient from the division of the first operand by the second operand. The operands must have numeric types.

Parameters

expr1 Left numeric expression.

expr2 Right numeric expression.

Returns

The accumulator shall be set to the result of the computation.

Notes

If *number1* is a finite, non-zero number, and *number2* is zero, the result of the division is *Infinity* if the *number1* is positive, and *-Infinity* if negative. If both *number1* and *number2* are zero, the result is *NaN*.

See Also

[Operators](#), [declare](#), [isnan](#), [isfinite](#)

/=

```
var /= expr1
```

Division assignment operator.

Description

The `/=` operator divides the value of the first operand *var* by the value of the second operand *expr1*, storing the result in the object specified by the first operand *var*.

Parameters

var A numeric scalar type.

expr1 A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

<

```
expr1 < expr2
```

Less than comparison.

Description

The `<` operator performs less-than comparisons between the operands `expr1` and `expr2`.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The accumulator shall be set to the result of the comparison.

See Also

[Operators](#), [declare](#)

```
<<
```

```
expr1 << expr2
```

Left-shift operator.

Description

The `<<` operator shift the value of the first operand `expr1` left the number of bits specified by the value of the second operand `expr1`.

Both operands must have an integral type, and the integral promotions are performed on them. The type of the result is the type of the promoted left operand.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

```
<<=
```

```
var <<= expr1
```

Left-shift assignment operator.

Description

The `<<=` operator shift the value of the first operand `var` left the number of bits specified by the value of the second operand `expr1`, storing the result in the object specified by the first operand `var`.

Parameters

`var` A numeric scalar type.
`expr1` A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

```
<=
```

```
expr1 <= expr2
```

Less than or equal comparison.

Description

The `<=` operator performs less-than comparisons between the operands `expr1` and `expr2`.

Parameters

- `expr1` Left expression.
- `expr2` Right expression.

Returns

The accumulator shall be set to the result of the comparison.

See Also

[Operators](#), [declare](#)

`<=>`

```
expr1 <=> expr2
```

Comparison operator.

Description

The `<=>` operator yields the value `-1` if the first expression is less than the second, `0` if the equals, and `1` the greater than; which are arithmetic and lexicographically comparisons respectively.

Parameters

- `expr1` Left expression.
- `expr2` Right expression.

Returns

The accumulator shall be set to the result of the comparison.

Limitations

Lists can not be compared.

See Also

[Operators](#), [declare](#)

=

```
var = expr1
```

Assignment operator.

Description

The `=` operator applies simple assignment, in which the value of the second operand `expr1` is stored in the object specified by the first operand `var`.

Parameters

- `var` Any scalar type.
- `expr1` An expression resulting in value of the same type as the scalar type `var`.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

`==`

```
expr1 == expr2
```

Equality operator.

Description

The `==` operator compares for equality, yielding the value `1` if the relation is true, and `0` if the relation is false. The result type is `int`.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The accumulator shall be set to the result of the comparison.

Limitations

Lists can not be compared.

See Also

[Operators](#), [declare](#)

>

`expr1 > expr2`

Greater than comparison.

Description

The `>` operator performs greater-than comparisons between the operands `expr1` and `expr2`.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The accumulator shall be set to the result of the comparison.

See Also

[Operators](#), [declare](#)

>=

`expr1 >= expr2`

Greater than or equal comparison.

Description

The `>` operator performs greater-than-or-equal comparisons between the operands `expr1` and `expr2`.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The accumulator shall be set to the result of the comparison.

See Also

[Operators](#), [declare](#)

>>

`expr1 >> expr2`

Right-shift operator.

Description

The `>>` operator shift the value of the first operand `expr1` right the number of bits specified by the value of the second operand `expr1`.

Both operands must have an integral type, and the integral promotions are performed on them. The type of the result is the type of the promoted left operand.

Parameters

`expr1` Left expression.
`expr2` Right expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

>>=

```
var >>= expr1
```

Right-shift assignment operator.

Description

The `>>=` operator shift the value of the first operand `var` right the number of bits specified by the value of the second operand `expr1`, storing the result in the object specified by the first operand `var`.

Parameters

`var` A numeric scalar type.

`expr1` A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

^

```
expr1 ^ expr2
```

Bitwise exclusive OR operator.

Description

The `^` operator yields the result is the bitwise exclusive OR of the two operands.

That is, the bit in the result is set if and only if exactly one of the corresponding bits in the operands are set.

Parameters

`expr1` Left expression.

`expr2` Right expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

^=

```
var ^= expr1
```

Bitwise exclusive OR assignment operator.

Description

The `^=` operator obtains the bitwise exclusive OR of the first operand `var` and second operand `expr1`, storing the result in the object specified by the first operand `var`.

Parameters

`var` A numeric scalar type.

`expr1` A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

post++

```
expr++
```

Postfix Increment.

Description

The *post++* operator increments the operand *expr* by 1, with the original value prior to the operation returned.

In other words, the original value of the operand is used in the expression, and then it is incremented.

Parameters

expr An expression.

Returns

The accumulator shall be set to orginal value of *expr* prior to being incremented.

See Also

[Operators](#), [declare](#)

post--

```
expr--
```

Postfix Decrement.

Description

The *post--* operator decrements the operand *expr* by 1, with the original value prior to the operation returned.

In other words, the original value of the operand is used in the expression, and then it is decremented.

Parameters

expr An expression.

Returns

The accumulator shall be set to orginal value of *expr* prior to being decremented.

See Also

[Operators](#), [declare](#)

|

```
expr1 | expr2
```

Bitwise OR operator.

Description

The **|** operator yields the result is the bitwise **inclusive OR** of the two operands *expr1* and *expr2*. That is, the bit in the result is set if at least one of the corresponding bits in the operands is set.

Parameters

expr1 Left expression.

expr2 Right expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

|=

```
var |= expr1
```

Bitwise OR assignment operator.

Description

The **|=** operator obtains the bitwise inclusive OR of the first operand *var* and second operand *var*, storing the result

in the object specified by the first operand *var*.

Parameters

var A numeric scalar type.
expr1 A numeric expression.

Returns

The accumulator shall be set to the result of the assignment.

See Also

[Operators](#), [declare](#)

||
expr1 || expr2

Logical OR operator.

Description

The **||** operator performs a logic OR between the two expressions *expr1* and *expr2*.

Each of the operands must have scalar type. If one or both of the operands is not equal to zero, then the result is 1. Otherwise, the result is zero (both operands are zero). The result type is int.

Short Circuit Evaluation

Logical operators are executed using *short-circuit* semantics whereby the second argument is only executed or evaluated if the first argument does not suffice to determine the value of the expression:

- Logical ADD - If the first operand is zero, then the second operand is not evaluated. Any side effects that would have happened if the second operand had been executed do not happen. Any function calls encountered in the second operand do not take place.
- Logical OR - If the first operand is not zero, then the second operand is not evaluated. Any side effects that would have happened if the second operand had been executed do not happen. Any function calls encountered in the second operand shall not take place.

See Also

[Logical Operators](#)

~
~expr

Bitwise complement.

Description

The **^** operator yields the bitwise complement 1's complement or bitwise not operator of *expr*.

The type of the operand must be an integral type, and integral promotion is performed on the operand. The type of the result is the type of the promoted operand.

Each bit of the result is the complement of the corresponding bit in the operand, effectively turning 0 bits to 1, and 1 bits to 0. The ! symbol is the logical not operator. Its operand must be a scalar type (not a structure, union or array). The result type is int.

If the operand has the value zero, then the result value is 1.

If the operand has some other value, then the result is 0.

Parameters

expr Expression.

Returns

The accumulator shall be set to the result of the computation.

See Also

[Operators](#), [declare](#)

\$Id: \$

Copyright © Adam Young All Rights Reserved.

Buffer Primitives

Summary

Buffer Primitives

Constants

`Buffer Flags` Buffer attribute constants.

Macros

<code>attach_buffer</code>	Attach a buffer to a window.
<code>create_buffer</code>	Create and load a buffer.
<code>create_nested_buffer</code>	Create or reference a buffer.
<code>delete_buffer</code>	Delete a buffer.
<code>delete_char</code>	Delete character.
<code>delete_line</code>	Delete current line.
<code>delete_to_eol</code>	Delete to end-of-line.
<code>find_line_flags</code>	Locate next line with specific flags.
<code>find_marker</code>	Locate next marker.
<code>goto_bookmark</code>	Seek a bookmark.
<code>inq_attribute</code>	Retrieve the current attributes.
<code>inq_buffer_flags</code>	Retrieve buffer flags.
<code>inq_buffer_title</code>	Retrieve a buffer title.
<code>inq_buffer_type</code>	Retrieve buffer type.
<code>inq_byte_pos</code>	Get current position in buffer stream.
<code>inq_encoding</code>	Retrieve a buffers character encoding.
<code>inq_file_change</code>	Determine state of underlying file.
<code>inq_indent</code>	Get current indentation settings.
<code>inq_line_flags</code>	Retrieve a lines associated flags.
<code>inq_line_length</code>	Determine the longest line length.
<code>inq_lines</code>	Retrieve the line count.
<code>inq_margins</code>	Retrieve buffer formatting margins.
<code>inq_modified</code>	Determine a buffers modification status.
<code>inq_names</code>	Retrieve associated buffer names.
<code>inq_position</code>	Retrieve current buffer position.
<code>inq_process_position</code>	Get position of process buffer.
<code>inq_ruler</code>	Retrieves the ruler specification.
<code>inq_system</code>	Determine if buffer is a system buffer.
<code>inq_tab</code>	Derive the tab increment.
<code>inq_tabs</code>	Retrieves the buffer tab specification.
<code>inq_terminator</code>	Retrieve a buffers line terminator.
<code>inq_time</code>	Retrieve the last modification time.
<code>mark_line</code>	Create a line marker.
<code>mode_string</code>	Conversion stat mode to a string representation.
<code>next_buffer</code>	Identifier of the next buffer.
<code>previous_buffer</code>	Identifier of the previous buffer.
<code>print</code>	Print formatted string to stdout.
<code>set_attribute</code>	Set the color attributes.
<code>set_buffer</code>	Set the current buffer.
<code>set_buffer_flags</code>	Set buffer flags.
<code>set_buffer_title</code>	Set a buffers title.
<code>set_buffer_type</code>	Set the buffer storage type.
<code>set_encoding</code>	Set a buffers character encoding.
<code>set_indent</code>	Set the buffers default indentation.
<code>set_line_flags</code>	Associate line flags.
<code>set_margins</code>	Set buffer formatting margins.
<code>set_process_position</code>	Set process insertion position.
<code>set_ruler</code>	Configure the buffer ruler.
<code>set_tab</code>	Derive the buffer tab stops.
<code>set_terminator</code>	Set a buffers line terminator.
<code>sort_buffer</code>	Sort buffer content.
<code>tabs</code>	Set buffer tab stops.
<code>tagdb_close</code>	Tag database close.
<code>tagdb_open</code>	Tag database open.
<code>tagdb_search</code>	Tag database search.
<code>write_buffer</code>	Write to buffer content.

Constants

Buffer Flags

Buffer attribute constants.

Description

Buffer flags are represented by bit-fields grouped in one of four sets. The following section describes the possible values for these flags:

First

First flag set, representing status and control primary buffer options.

Constant	Description
BF_CHANGED	Changed.
BF_BACKUP	Backup required on next write.
BF_RDONLY	Read-only.
BF_READ	Buffer content still to be read.
BF_EXEC	File is executable.
BF_PROCESS	Buffer has process attached.
BF_BINARY	Binary buffer.
BF_ANSI	If TRUE, ANSI-fication is done.
BF_TABS	Buffer inserts real-tabs.
BF_SYSBUF	Buffer is a system buffer.
BF_LOCK	File lock.
BF_NO_UNDO	Dont keep undo info.
BF_NEW_FILE	File is a new file, so write even if no changes.
BF_CR_MODE	Append <CR> to end of each line on output.
BF_SYNTAX	Enable syntax highlighting (unless ANSI).
BF_SYNTAX_MATCH	Hilite matching braces.
BF_MAN	If TRUE, man style \b is done.
BF_SPELL	Enable spell.
BF_FOLDING	Test folding/hiding.
BF_RULER	Display ruler.
BF_VOLATILE	Buffer is volatile.
BF_EOF_DISPLAY	Show <EOF> markers.
BF_HIDDEN	Hidden buffer, from buffer list
BF_AUTOREAD	Automatically re-read buffer, if underlying changes.
BF_AUTOWRITE	Automatically write buffer, if modified
BF_SCRAPBUF	Scrap buffer.
BF_DELAYED	Content load delayed until first reference.

Second

Second buffer set, controlling general UI formatting options.

Constant	Description
BF2_ATTRIBUTES	Character attributes, enables character cell coloring.
BF2_DIALOG	Dialog
BF2_CURSOR_ROW	Display cursor cross-hair.
BF2_CURSOR_COL	
BF2_TILDE_DISPLAY	
BF2_EOL_HILITE	Limit hilites to EOL.
BF2_LINE_NUMBERS	Line numbers
BF2_LINE_OLDNUMBERS	If has line numbers, display old lines
BF2_LINE_STATUS	Markup modified lines.
BF2_LINE_SYNTAX	Syntax preprocessor flags
BF2_TITLE_FULL	Label window using full path name
BF2_TITLE_SCROLL	Scroll title with window
BF2_TITLE_LEFT	Left justify title
BF2_TITLE_RIGHT	Right justify title
BF2_SUFFIX_RO	Read-only suffix on title
BF2_SUFFIX_MOD	Modified suffix on title
BF2_EOL_CURSOR	Limit cursor to EOL
BF2_EOF_CURSOR	Limit cursor to EOF
BF2_HILITERAL	Hilite literal characters
BF2_HIWHITESPACE	Hilite whitespace
BF2_HIMODIFIED	Hilite modified lines

Constant	Description
BF2_HIADDITIONAL	Hilite added lines
BF2_HSTATUSLINE	Status line

Third

Third flag set, controlling indirect buffer functionality which are generally implemented at a macro level.

Constant	Description
BF3_AUTOSAVE	Auto-save
BF3_AUTOINDENT	Auto-indent
BF3_AUTOWRAP	Auto-wrap
BF3_PASTE_MODE	Paste mode, disables a number of auto functions.

Fourth

Fourth flag set, controlling file conversion options.

Constant	Description
BF4_OCVT_TRIMWHITE	Output conversion, trim trailing whitespace

See Also

[inq_buffer_flags](#), [set_buffer_flags](#)

Macros

attach_buffer

```
void attach_buffer(int bufnum)
```

Attach a buffer to a window.

Description

The `attach_buffer()` primitive attaches the specified buffer to the current window, so that the window becomes a view port into the buffer content.

This interface is generally used in combination with `set_buffer`, `set_window` and/or buffer/window creation. Care should be taken to always have both the current buffer attached to the current window at end of any macros returning control backup, otherwise results are undefined.

For example, create a buffer and window and then associate the two.

```
int buf = create_buffer("buffer");
int win = create_window(20, 10, 60, 2);
attach_buffer(buf);
```

When the specified buffer is attached to the current window, the top title of the window is changed to reflect the buffer or file-name associated with the buffer.

Parameters

bufnum Buffer identifier to be attached.

Notes

A few events automatically affect the attached buffer.

- An explicit `attach_buffer()` is performed during `create_tiled_window` calls.
- Deleting an attached buffer, results in all associated windows being reassigned the top buffer.

Returns

nothing

Portability

n/a

See Also

[create_buffer](#), [create_window](#), [inq_buffer](#)

create_buffer

```
int create_buffer( string bufname,
                  [string filename],
                  [int sysflag = FALSE],
                  [int editflags = 0],
                  [string encoding = ""])
```

Create and load a buffer.

Description

The `create_buffer()` primitive creates a buffer containing the content of an optional underlying file *filename*, with the filename being a full or relative path name of a file which should be read into the buffer. If this parameter is omitted, then an initially empty buffer will be created.

The buffer name *bufname* should be unique, and if the buffer already exists a unique name shall be derived by prefixing its name with "[x]"; *x* being the next available unique sequence.

The optional arguments *sysflag*, *editflags* and *encoding* control a variety of the buffer features.

Callbacks

Upon buffer loads firstly the `_extension` callback is executed and either an extension specific callback of the form `_ext` or `_default` shall be executed.

In addition any related `register_macro` callbacks are executed.

Example

The following creates the buffer named "Read-Me" and populates it with the content of the file "readme.txt".

```
int buf;

if ((buf = create_buffer("Read-Me", "readme.txt")) >= 0) {
    attach_buffer(newbuf);
    return buf;
}
message("error loading buffer ...");
return -1;
```

Parameters

<code>bufname</code>	String containing the unique buffer name. The name is used as the buffer title, which is usually the same as the underlying filename yet it need not be; for example an abbreviated form. The buffer names does not affect the file that shall be loaded into the buffer.
<code>filename</code>	Optional string containing the file that the buffer should contain, if omitted an empty buffer is created.
<code>sysflag</code>	Optional integer boolean flag stating whether or not the buffer is a system buffer, FALSE for non-system otherwise TRUE for system; if omitted the buffer is assumed to be a non-system. See inq_system is more details on system buffers.
<code>editflags</code>	Optional buffer creation flags. These flags control the file mode, see the edit_file primitive for additional information describing this field.
<code>encoding</code>	Optional buffer encoding hint.

Returns

The `create_buffer()` primitive returns the buffer identifier associated with the newly created buffer, otherwise -1 if the buffer was not created.

Note the buffer does not become the current buffer until the `set_buffer` primitive is used against the returned buffer identifier.

Portability

n/a

See Also

[attach_buffer](#), [delete_buffer](#), [set_buffer](#), [create_nested_buffer](#), [set_buffer_title](#)

create_nested_buffer

```
int create_nested_buffer( string bufname,
                         [string filename],
                         [int sysflag],
                         [int editflags],
                         [string encoding] )
```

Create or reference a buffer.

Description

The `create_nested_buffer()` primitive is similar to the `create_buffer` primitive yet if the buffer already exists its reference counter is incremented.

For each *nested* buffer increment `delete_buffer` must be called, with the buffer only being removed upon the reference count being zero.

This primitive but is provided for convenience when temporary access to a buffer is required, see the `ff` macro for a working example.

Parameters

<code>bufname</code>	String containing the unique buffer name. The name is used as the buffer title, which is usually the same as the underlying filename yet it need not be; for example an abbreviated form. The buffer names does not affect the file that shall be loaded into the buffer.
<code>filename</code>	Optional string containing the file that the buffer should contain, if omitted an empty buffer is created.
<code>sysflag</code>	Optional integer boolean flag stating whether or not the buffer is a system buffer, FALSE for non-system otherwise TRUE for system; if omitted the buffer is assumed to be a non-system. See <code>inq_system</code> is more details on system buffers.
<code>editflags</code>	Optional buffer creation flags. These flags control the file mode, see the <code>edit_file</code> primitive for additional information describing this field.
<code>encoding</code>	Optional buffer encoding hint.

Returns

The `create_nested_buffer()` primitive returns the buffer identifier associated with the newly created buffer, otherwise -1 if the buffer was not created.

Portability

n/a

See Also

`attach_buffer`, `delete_buffer`, `set_buffer`, `create_buffer`, `set_buffer_title`

delete_buffer

```
void delete_buffer(int bufnum)
```

Delete a buffer.

Description

The `delete_buffer()` primitive deletes the specified buffer, the buffer contents and all associated resources are released.

Any changes made to the buffer since it was last written shall be lost, as such if required the content should be written using `write_buffer`.

In the case of a process buffer, the underlying sub-process is shutdown.

Once deleted, the associated buffer handle is invalid.

Parameters

`bufnum` Non-optional buffer number.

Returns

nothing

Portability

n/a

See Also

`create_buffer`, `create_nested_buffer`, `set_buffer`

delete_char

```
void delete_char([int num])
```

Delete character.

Description

The *delete_char()* primitive deletes one or more characters at the current cursor position.

This primitive is the default assignment for the <Delete> key on the keyboard.

Parameters

num Optional integer, if stated specifies the number of characters to be deleted, if omitted only a single character is removed.

Returns

nothing

Portability

n/a

See Also

[backspace](#), [delete_block](#), [delete_line](#)

delete_line

```
void delete_line()
```

Delete current line.

Description

The *delete_line()* primitive deletes the current line, placing the cursor at the same column on the following line.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[delete_char](#), [delete_to_eol](#)

delete_to_eol

```
void delete_to_eol()
```

Delete to end-of-line.

Description

The *delete_to_eol()* primitive deletes from the current cursor position to the end-of-line, not including the newline. The cursor position remains unchanged.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[delete_char](#), [delete_line](#)

find_line_flags

```
int find_line_flags(      [int bufnum],
                        [int lineno],
                        int mode,
                        int and_mask,
                        [int or_value],
                        [int value]      )
```

Locate next line with specific flags.

Description

The *find_line_flag()* primitive positions the cursor at the next line which matches the specified flag value.

bufnum and *lineno* allow explicit buffer and line number references to be stated, otherwise if omitted the current buffer and/or associated line number shall be used.

flags defines direction and type of equivalence test to be utilised. *and_mask* and *or_value* parameterise the equivalence expression.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

lineno Starting line number.

mode Search mode flags.

LF_FORWARD Forward search (default).

LF_BACKWARDS Backwards search.

LF_MATCH_EQ Absolute value match against the line flags AND'ed against *and_mask* and then OR'ed against *or_value*.
ie. $((\text{flags} \& \text{and_mask}) | \text{or_value}) == \text{value}$

LF_MATCH_ANY Match line flags were any flags contained within the *and_mask* are set,
ie. $((\text{flags} \& \text{and_mask}) != 0)$.

and_mask AND mask.
or_value OR value during LF_MATCH_EQ operations.
value Value being matched during LF_MATCH_EQ operations.

Returns

The *find_line_flag()* primitive returns the matched line number, 0 if not suitable match was found, otherwise -1 on error, for example invalid parameters.

See Also

[find_marker](#)

```
int find_marker([int marker = L_MARKED])
```

Locate next marker.

Description

The *find_marker()* primitive positions the cursor at the next line which has a user defined marker enabled. On successful completion the marker is removed.

marker is the optional marker against which to search, by default L_MARKED. Only L_MARKED or one of the L_USERx definitions maybe be specified, if other bits or more then one user bit is stated the search shall not succeed.

Returns

The *find_marker()* primitive returns 1 on success and 0 if no additional markers exist.

See Also

[mark_line](#), [find_line_flags](#)

Compatibility

The *marker* parameter is a **GriefEdit** extension.

goto_bookmark

```
int goto_bookmark( int bookid      = NULL,
                   [int &bufnum],
                   [int &line],
                   [int &column]     )
```

Seek a bookmark.

Description

The *goto_bookmark()* primitive changes the current buffer and cursor location to the values associated with the named bookmark.

bookid is the unique identifier or name associated with the bookmark; any valid integer may be used as the identifier. If omitted the user shall be prompted for the bookmark identifier.

Go to bookmark:

If any of the arguments *bufnum*, *line* or *column* are specified, these are modified to contain the related bookmark values without effecting the current buffer or location. If all are omitted the bookmark is applied, updating the buffer and/or cursor location as required.

Parameters

bookid Bookmark identifier.
bufnum Optional integer reference, if specified shall be populated with the associate buffer number.
line Optional integer reference, if specified shall be populated with the buffer line number.
column Optional integer reference, if specified shall be populated with the buffer column number.

Returns

The *goto_bookmark()* primitive returns a non-zero value and populate any of the supplied arguments *bufnum*, *line* and *col*. Otherwise on error returns zero and a related error shall be displayed.

No such bookmark

goto_bookmark: No such buffer

Examples

The following two examples deal with the bookmark labelled as 9.

Retrieves the bookmark definition and echos the details to the user.

```
int buf, line, column;
goto_bookmark(9, buf, line, column);
message("bookmark: buf=%d, %d/%d", buf, line, column);
```

Applies the bookmark definition.

```
goto_bookmark(9);
```

Portability

n/a

See Also

[bookmark_list](#), [delete_bookmark](#), [drop_bookmark](#)

inq_attribute

```
int inq_attribute( [int &normal],
                   [int bufnum] )
```

Retrieve the current attributes.

Description

The *inq_attribute()* primitive retrieves the text and optionally the normal attribute for the specified buffer *bufnum*.

Parameters

normal Optional integer reference, if stated is populated with the clear/normal attribute value.
bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The *inq_attribute()* primitive returns the current text attribute, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[set_attribute](#)

inq_buffer_flags

```
int inq_buffer_flags( [int bufnum],
                      [string flag|int set = 1],
                      [string ~flags]
                    )
```

Retrieve buffer flags.

Description

The *inq_buffer_flags()* primitive retrieves one of the set of flags associated with the specific buffers, see [Buffer Flags](#).

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.
 flag/set Optional internal set identifier, if omitted the primary set(1) shall be referenced.
 ... Optional string of comma separated flag names.

Flags

The following table summaries the existing flags, for additional on a specific flag consult the [set_buffer_flags](#) primitive.

Returns

The *inq_buffer_flags()* primitive returns the value associated with the selected set of flags.

Portability

The string flag parameter variant is a GRIEF extension.

Many of the flags are GRIEF specific; CRISP™ has a similar primitive yet as the two were developed independently features differ.

See Also

[set_buffer_flags](#)

inq_buffer_title

```
string inq_buffer_title([int bufnum])
```

Retrieve a buffer title.

Description

The *inq_buffer_title()* primitive retrieves the title associated with the specified buffer.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

String containing the current buffer title.

Portability

n/a

See Also

[create_buffer](#), [set_buffer_title](#)

inq_buffer_type

```
int inq_buffer_type( [int bufnum],
                     [string &desc],
                     [string &encoding])
```

Retrieve buffer type.

Description

The *inq_buffer_type()* primitive retrieves the buffer type of the buffer *bufnum*.

Parameters

<i>bufnum</i>	Optional buffer number, if omitted the current buffer shall be referenced.
<i>desc</i>	Optional string variable reference to be populated with the buffer encoding description.
<i>encoding</i>	Optional string variable reference to be populated with the buffers character encoding name.

Returns

The *inq_buffer_type()* primitive returns on the following manifest constants representing the base encoding of the referenced buffer.

Buffer Types

The following manifest constants define the available Buffer Types.

Constant	Description
BFTYP_UNKNOWN	Unknown buffer type.
BFTYP_UNIX	Unix, LF line termination.
BFTYP_DOS	DOS, CF/LF line termination.
BFTYP_MAC	Old style MAX, CR termination.
BFTYP_BINARY	Binary.
BFTYP_ANSI	ANSI.
BFTYP_EBCDIC	EBCDIC.
BFTYP_UTF8	UTF8.
BFTYP_UTF16	UTF16/USC2.
BFTYP_UTF32	UTF32/USC4.
BFTYP_UTFEBCDIC	UTF8/EBCDIC.
BFTYP_BOUC1	Binary Ordered Compression for Unicode.
BFTYP_SCSU	Standard Compression Scheme for Unicode.
BFTYP_UTF7	7-bit Unicode Transformation Format.
BFTYP_GB	GB.
BFTYP_BIG5	BIG5.
BFTYP_ISO2022	ISO-2022.
BFTYP_SBCS	Single Byte.
BFTYP_DBCS	Double Byte.
BFTYP_MBCS	Multi-Byte (Non Unicode).
BFTYP_OTHER	Other supported.
BFTYP_UNSUPPORTED	Known file-type, yet no internal support.

Portability

n/a

See Also

[set_buffer_type](#)

inq_byte_pos

```
int inq_byte_pos( [int bufnum],
                  [int line],
                  [int col],
                  [int flags] )
```

Get current position in buffer stream.

Description

The *inq_byte_pos()* primitive is reserved for future compatibility.

The *inq_byte_pos()* primitive calculates and returns the byte offset from the start of the specified buffer with the first byte within the underlying buffer being at offset 0.

This primitive is similar the native library function *tell*.

Parameters

- `bufnum` Optional buffer number, if omitted the current buffer shall be referenced.
- `line` Optional line number.
- `col` Optional column.
- `flags` Offset origin flag, omitted when 0x00.

Returns

The `inq_byte_pos()` primitive returns the current value of the file-position indicator for the associated buffer measured in bytes from the beginning of the file. Otherwise, it returns -1 on error.

Portability

n/a

See Also

[inq_terminator](#)

inq_encoding

```
string inq_encoding([int bufnum])
```

Retrieve a buffers character encoding.

Description

The `inq_encoding()` primitive retrieves the character encoding associated with the referenced buffer. See [set_encoding](#) for possible encodings.

Parameters

- `bufnum` Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The `inq_encoding()` primitive returns the associated encoding.

Portability

A **GriefEdit** extension.

See Also

[set_encoding](#)

inq_file_change

```
int inq_file_change([int bufnum])
```

Determine state of underlying file.

Description

The `inq_file_change()` primitive determines the state of the file which underlies the specified buffer `bufnum`. This primitive checks whether the associated file has been modified or deleted.

Parameters

- `bufnum` Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The `inq_file_change()` primitive returns the reason code for the file state change.

- 0 No change.
- 1 File status detected; possible in-place changes.
- 2 File modified, size differences detected.
- 3 Underlying file does not exist (i.e. has been deleted).
- 1 Unknown error, the cause of the error condition can be derived from the system return code (See: [errno](#)).

Portability

A **GriefEdit** extension.

See Also

[inq_modified](#), [edit_file](#)

inq_indent

```
int inq_indent([int bufnum])
```

Get current indentation settings.

Description

The *inq_indent()* primitive retrieves the current buffer indentation of the specified buffer *bufnum*.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

Returns the non-zero indentation value if indentation is active, otherwise 0.

Portability

A **GriefEdit** extension.

See Also

[set_indent](#)

inq_line_flags

```
int inq_line_flags([int bufnum],
                  [int lineno],
                  [int& iflags])
```

Retrieve a lines associated flags.

Description

The *inq_line_flags()* primitive retrieves the flags associated with the specified line.

The line flags was a set of 32 bit values, with the upper 16 bits being defined for **GriefEdit** usage and lower 16 bits for user/macro usage.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

lineno Line number within the selected buffer, otherwise if omitted the current line is referenced.

iflags Optional storage for the associated internal flags.

Returns

Associated line flags.

Compatibility

Internal flags are an **GriefEdit** extension.

See Also

[set_line_flags](#), [find_line_flags](#)

inq_line_length

```
int inq_line_length([int bufnum])
```

Determine the longest line length.

Description

The *inq_line_length()* primitive determines the length of the longest line within the specified buffer *bufnum*.

The calculate line length corresponds to the logical column position at the end of the line, taking into account any tabs and control characters.

If the designated buffer contains a marked region, then only the lines within the marked region are including within the result.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The *inq_line_length()* primitive returns the longest line within the referenced buffer or region, otherwise -1 on error.

Portability

Unlike BRIEF the longest current line is returned. BRIEF returned the upper global line length rounded up to the next multiple of 16, for example 202 would have been rounded to 208, not a buffer specific value.

See Also

[inq_lines](#)

inq_lines

```
int inq_lines([int bufnum])
```

Retrieve the line count.

Description

The *inq_lines()* primitive returns the current line number of the specified buffer *bufnum*.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The *inq_lines()* primitive returns the line count within the referenced buffer, otherwise -1 on error.

Portability

n/a

See Also

[inq_line_length](#)

inq_margins

```
int inq_margins( [int bufnum],
                  [int &left],
                  [int &right],
                  [int &style],
                  [int &colorcolumn],
                  [int global = TRUE])
```

Retrieve buffer formatting margins.

Description

The *inq_margins()* primitive retrieves one or more of the specified buffers *bufnum* current margins.

Parameters

<i>bufnum</i>	Optional buffer number, if omitted the current buffer shall be referenced. A negative <i>bufnum</i> (e.g. -1) shall retrieve the global margin parameters, which are applied when no buffer specific margin has been set.
<i>left</i>	Optional left margin.
<i>right</i>	Optional right margin.
<i>style</i>	Optional justification style.
<i>colorcolumn</i>	Optional colour column.
<i>global</i>	Optional integer flag, if given as FALSE when retrieving buffer margins the global settings shall not be applied when no buffer specific value is available.

Returns

The *inq_margins()* primitive returns 0 on success, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[set_margins](#)

inq_modified

```
int inq_modified([int bufnum])
```

Determine a buffers modification status.

Description

The *inq_modified()* primitive determine whether the specified buffer *bufnum* has been modified.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Example

The following echos to the command prompt the current buffers modification status.

```
message("Buffer has %sbeen modified.",
       inq_modified() ? "" : "not ");
```

Returns

The *inq_modified()* primitive returns the modification status, **true** when modified otherwise **false** if the buffer has no changes since loading or the last save.

Portability

n/a

See Also

[inq_time](#), [inq_system](#)

inq_names

```
int inq_names( [string fullname],
               [string ext],
               [string bufname],
               [int bufnum]      )
```

Retrieve associated buffer names.

Description

The *inq_names()* primitive retrieves the file and/or buffer names associated with the specified buffer *bufnum*.

Parameters

fullname Optional string variable reference, if specified shall be populated with the full path name of the underlying file, that is used on [write_buffer](#) calls.
ext Optional string variable reference, if specified shall be populated the file extension taken from the full path.
bufname Optional string variable reference, if specified shall be populated with buffer name which is used as the buffer title, see [set_buffer_title](#); which is usually the basename, i.e. full path without the path.
bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The *inq_names()* primitive returns 0 on success, otherwise -1 on error.

Portability

n/a

See Also

[create_buffer](#), [set_buffer_title](#)

inq_position

```
int inq_position([int &line],
                 [int &col]   )
```

Retrieve current buffer position.

Description

The *inq_position()* primitive retrieves the current cursor position in the current buffer.

Parameters

line Optional integer variable when supplied shall be populated with the current buffer line.
col Optional integer variable when supplied shall be populated with the current buffer column.

Returns

The *inq_position()* primitive returns 0 if the current position is not past the end of the buffer. Otherwise, the number of lines between the end of the buffer and the current position.

Portability

n/a

See Also

[move_abs](#), [move_rel](#)

inq_process_position

```
int inq_process_position([int &line],
                        [int &column])
```

Get position of process buffer.

Description

The *inq_process_position()* primitive retrieves the current cursor position for the underlying process, this primitive is similar to *inq_position*.

The process position is used for output from the process; rather than inserting the output from the process where the users cursor is, a separate cursor is maintained instead.

This permits the user to move around the buffer whilst the process is generating output without the process output being sprinkled through the buffer.

Parameters

line Optional integer variable to be populated with the cursor line.
column Optional integer variable to be populated with the cursor row.

Returns

The *inq_process_position()* primitive returns 0 on sucess, otherwise -1 if the current buffer is not attached to a process.

Portability

n/a

See Also

[set_process_position](#), [connect](#)

inq_ruler

```
string|list inq_ruler([int bufnum],
                      [int min_count],
                      [int aslist = FALSE])
```

Retrieves the ruler specification.

Description

The *inq_ruler()* primitive retrieves the effective indentation specification of the current buffer.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.
min_count Optional integer, allows the specification of the minimum number of tab points which shall be presented within the returned specification.
aslist Optional integer boolean flags, if **TRUE** the tab specification is returned in the form of a list of integers, otherwise by default as a string specification.

Returns

The *inq_ruler()* primitive either returns a space separated string or an integer list containing the current indentation specification; both are suitable for use by [set_ruler](#).

Portability

A **GriefEdit** extension.

See Also

[set_ruler](#)

inq_system

```
int inq_system([int bufnum])
```

Determine if buffer is a system buffer.

Description

The *inq_system()* primitive determines whether the specified buffer *bufnum* is marked as a system buffer.

System buffers do not appear in buffer lists, are not editable by users and are handled specially by many macros.
System buffer are generally utilised by macros for internal work pads.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The *inq_system()* primitive returns non-zero if the associated buffer is a system buffer, otherwise 0 if the buffer is a normal buffer.

Portability

n/a

See Also

[create_buffer](#), [set_buffer_flags](#), [inq_modified](#)

inq_tab

```
int inq_tab([int bufnum])
```

Derive the tab increment.

Description

The *inq_tab()* primitive derives the tab increment in force at the current cursor position.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

Returns the current tab increment, otherwise the default of 8 if none is active.

Portability

An **GriefEdit** extension.

See Also

[tabs](#), [set_indent](#), [set_ruler](#)

inq_tabs

```
string|list inq_tabs([int bufnum],
                     [int min_count],
                     [int aslist = FALSE]))
```

Retrieves the buffer tab specification.

Description

The *inq_tabs()* primitive retrieves the effective tabs specification of the current buffer.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

min_count Optional integer, allows the specification of the minimum number of tab points which shall be presented within the returned specification.

aslist Optional integer boolean flags, if **TRUE** the tab specification is returned in the form of a list of integers, otherwise by default as a string specification.

Returns

The *inq_tabs()* primitive either returns a space separated string or an integer list containing the current tabs specification; both are suitable for use by [tabs](#).

Portability

A **GriefEdit** extension.

See Also

[tabs](#), [set_indent](#), [distance_to_tab](#), [distance_to_indent](#)

inq_terminator

```
int inq_terminator([int bufnum],
                   [string &term])
```

Retrieve a buffers line terminator.

Description

The *inq_terminator()* primitive retrieves the line terminator of the specified buffer *bufnum*.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.
term Optional string variable reference, to be populated with the line terminator of referenced buffer.

Returns

The *inq_terminator()* primitive returns the line terminator type of the specified buffer (See: [set_terminator](#)), otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[set_terminator](#)

inq_time

```
int inq_time([int bufnum],
             [int &ctime] )
```

Retrieve the last modification time.

Description

The *inq_time()* primitive returns the time at which the last modification occurred of the specified buffer *bufnum*, represented by the number seconds since the beginning of the current edit session.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.
ctime Optional integer value reference, if specified shall be populated with the associated system time, being the number of second since 1970/01/01 UTC.

Returns

The *inq_time()* primitive returns the time in seconds of last modification, 0 if the buffer has not been modified during the current edit session, otherwise -1 if the buffer is invalid.

Portability

n/a

See Also

[inq_modified](#)

mark_line

```
int mark_line( int flag,
               [int toggleall],
               [int bufnum],
               [int lineno = 0],
               [int marker = L_MARKED])
```

Create a line marker.

Description

The *mark_line* primitive controls the value of the user defined line marker within the current buffer. This mark maybe be utilised by macro developers to maintain a collection of lines on which can then be queried using

`find_marker()` for additional processing.

The `flags` parameter controls the mark value; if non-zero `true` then the marker is set otherwise it is cleared. When `toggleall` is stated then the mark status of all lines is toggled, ignoring the `flag` specification.

`bufnum` and `lineno` allow explicit buffer and line number references to be stated, otherwise if omitted the current buffer and/or associated line number shall be used.

`marker` is the optional marker against which to search, by default `L_MARKED`. Only `L_MARKED` or one of the `L_USERx` definitions maybe be specified.

Note:

Markers are only a temporary resource which maybe cleared when line are modified, deleted etc.

Returns

The `mark_line` primitives returns 1 if the marker was already set, 0 if the marker was not set, otherwise -1 when beyond the end of the buffer.

Compatibility

The options `bufnum`, `lineno` and `marker` are **GriefEdit** extensions.

See Also

`find_marker`

mode_string

```
string mode_string([int mode],
                  [int format = 0],
                  [string path]      )
```

Conversion stat mode to a string representation.

Description

The `mode_string()` primitive decodes the specified `mode` into a human readable form using a style similar to `/s` long listing format output detailing type and permissions, for example

drwxr-xr-x-

Mode String

The decoded mode string consists of a ten character string, using the following format ((see [File Modes](#)))

<type> <owner> <group> <other> <sticky>

Type

The first character indicates the file `type` and is not related to permissions, when `format` is omitted or (0) shall be one of the following:

- 'd' Directory.
- 'c' Character-device.
- 'b' Block-device.
- 'l' Link.
- 'p' Fifo/pipe.
- 's' Sockets.
- 'n' Name.
- 'D' Door.
- '-' Normal.

The alternative format shall be used when `format` is given as a non-zero value. In addition if specified `source` shall be utilised to verify the status of the link.

- ' /' Directories.
- ' -' Character devices.
- ' +' Block devices.
- ' ~' Directory link.

'!' Broken link.
'@' Link.
'|' Fifo/pipe.
'=' Sockets.
'\$' Name/door.
'*' Executable.
'-' Normal (space).

Permissions

Following are three permission sets defining the *user*, *group* and *other* access rights.

Each of the three characters represent the read, write, and execute permissions for each of the groups in the order (rwx).

'r' Read permission.
'w' Write permission.
'x' Execute permission
'-' No associated permission read, write or execute.

Sticky

The trailing character details the one of two special attributes.

'S' S_ISUID is set.
'T' S_ISVTX is set.

Parameters

`mode` Optional mode specification, otherwise the associate mode of current buffer is decoded.
`format` Optional format, when stated and non-zero the <type> field is decoded using an alternative form.
`path` Optional source of the mode, is supplied shall be utilised to verify the status of links.

Returns

Returns the decoded mode string.

Portability

A **GriefEdit** extension.

See Also

[File Modes](#), [stat](#), [lstat](#)

next_buffer

```
int next_buffer( [int sysflag = 0],
                 [int previous],
                 [int tab] )
```

Identifier of the next buffer.

Description

The `next_buffer()` primitive retrieves the buffer identifier of the next buffer after the current buffer in the buffer list, optionally filtering system buffers.

The buffer list, which is maintained by the GRIEF kernel, is a circular list of all buffers. Upon the end of list being reached, the first buffer on the list is returned as the next.

Note:

The `next_buffer` primitive does not alter the current buffer, the `set_buffer` can be used to select the returned buffer.

Parameters

`sysflag` Optional system buffer filter selection. If either omitted or zero system buffers shall be filtered from the returned identifiers. Otherwise all buffers including system shall be returned.
`prev` Optional boolean flag, if stated as non-zero then the previous buffer in the buffer list is retrieved.
`tab` Reserved for future use; tab identifier.

Returns

The `next_buffer()` primitive returns the buffer identifier of the next or previous buffer.

Portability

n/a

See Also[previous_buffer](#), [set_buffer](#), [create_buffer](#), [inq_buffer](#), [inq_system](#)**previous_buffer**

```
int previous_buffer([int sysflag = 0],
                   [int tab] )
```

Identifier of the previous buffer.

Description

The *previous_buffer()* primitive retrieves the buffer identifier of the previous buffer after the current buffer in the buffer list, optionally filtering system buffers.

The buffer list, which is maintained by the GRIEF kernel, is a circular list of all buffers. Upon the beginning of list being reached, the last buffer on the list is returned as the previous.

Note:

The *previous_buffer* primitive does not alter the current buffer, the [set_buffer](#) can be used to select the returned buffer.

Parameters

sysflag Optional system buffer filter selection. If either omitted or zero system buffers shall be filtered from the returned identifiers. Otherwise all buffers including system shall be returned.
tab Reserved for future use; tab identifier.

Returns

The *previous_buffer()* primitive returns the buffer identifier of the previous buffer.

Portability

n/a

See Also[next_buffer](#), [set_buffer](#), [create_buffer](#), [inq_buffer](#), [inq_system](#)**print**

```
int print()
```

Print formatted string to stdout.

Description

The *print()* primitive sends the contents of the currently marked area to the printer.

Parameters

none

Returns

0 for success, -1 for printer busy, less than -1 for other printer errors or no marked block.

Portability

n/a

See Also[error](#), [message](#), [dprintf](#)**set_attribute**

```
int set_attribute( [int|string text],
                  [int|string normal],
                  [int bufnum] )
```

Set the color attributes.

Description

The `set_attribute()` primitive set the text and/or normal attributes for the specified buffer `bufnum`.

Parameters

`text` Optional text attribute either by value or name.
`normal` Optional clear/normal attribute either by value or name.
`bufnum` Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The `set_attribute()` primitive returns the previous text attribute, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

`inq_attribute`

set_buffer

```
int set_buffer(int bufnum)
```

Set the current buffer.

Description

The `set_buffer()` primitive makes the buffer identifier specified by `bufnum` the current buffer, without effecting the current window. The current is the one referenced by all buffer operations which are not given an explicit buffer identifier.

Generally `set_buffer` is utilised in one of two ways;

- Temporarily changing the buffer so to perform specific buffer processing, for example searching for text, and on completion the previous is then restored to the current.
- Changing the active buffer, which should also involve changing the current window using `set_window` or associating the new buffer with the current window using `attach_buffer`.

The `set_buffer()` primitive unlike `edit_file` does not cause any registered macros to be executed.

Warning:

The referenced buffer does not always need to be attached to a window nor the one currently associated with the current window `set_window`, yet upon macro exit the current buffer and current window **should** be attached otherwise the side-effects may be disastrous.

Parameters

`bufnum` Buffer identifier to be selected.

Returns

The `set_buffer()` primitive returns the identifier of the previous current buffer otherwise -1 if an invalid buffer identifier was stated.

On failure the following diagnostics message shall be echoed on the command prompt.

```
'set_buffer': no such buffer
```

Portability

n/a

See Also

`inq_buffer`, `create_buffer`, `next_buffer`, `previous_buffer`

set_buffer_flags

```
void set_buffer_flags( [int bufnum],
                      [string|int or_mask],
                      [string|int and_mask],
                      [int set = 1] )
```

Set buffer flags.

Description

The `set_buffer_flags()` primitive modifies the internal flags associated with the specified buffer, see [Buffer Flags](#).

If specified one or more flags shall be cleared using the `and_mask`, in addition one or more flags are set using the `or_mask`.

Each buffer maintains several sets of integer flags which can be modified. Against the selected flag `set` the optional `and_mask` (clear) and then the optional `or_mask` (set) is applied.

Parameters

<code>bufnum</code>	Optional buffer number, if omitted the current buffer shall be referenced.
<code>set_mask</code>	Optional mask of flags to set. May either be an integer of AND'ed together flag constants, or alternatively a string of comma separated flag names.
<code>clear_mask</code>	Optional mask of flags to clear. May either be an integer of AND'ed together flag constants, or alternatively a string of comma separated flag names.
<code>set</code>	Optional integer stating the flag set to be modified, if omitted defaults to the primary set(1).

Returns

`nothing`

Portability

The string mask variants and `set` parameter are GRIEF extension.

Many of the flags are GRIEF specific; *CRISP*™ has a similar primitive yet as the two were developed independently features differ.

See Also

[inq_buffer_flags](#)

set_buffer_title

```
int set_buffer_title( [int bufnum],
                      [string title])
```

Set a buffers title.

Description

The `set_buffer_title()` primitive sets the buffer title of the stated buffer otherwise the current buffer when omitted. The specified title is displayed on the top edge of the buffers associated window.

Parameters

<code>bufnum</code>	Optional buffer number, if omitted the current buffer shall be referenced.
<code>title</code>	Optional string value of the title to associate. If omitted specified, then the buffer title is remove with the buffers underlying filename being used.

Returns

The `set_buffer_title()` primitives return zero on success, otherwise -1 if the specified buffer does not exist.

Portability

n/a

See Also

[inq_buffer_title](#), [create_buffer](#)

set_buffer_type

```
int set_buffer_type( [int bufnum],
                     [int type = NULL],
                     [string encoding = NULL])
```

Set the buffer storage type.

Description

The `set_buffer_type()` primitive optionally set the buffer type and/or the character encoding associated with the specified buffer.

Note that the specified `encoding` has priority over the buffer type, in that an incompatible encoding with the stated `type` or pre-existing buffer type shall imply the default buffer type associated with the encoding. The `inq_buffer_type()` primitive should be used to determine the resulting buffer type on completion.

Parameters

bufnum	Optional buffer number, if omitted the current buffer shall be referenced.
type	Optional integer buffer type which states the basic buffer encoding include an implied line termination.
encoding	Optional string which sets the specific buffer encoding beyond the buffer type, for example the page code utilized by a BFTYPE_DOS buffer.

Buffer Types

The following manifest constants define the available Buffer Types.

Constant	Description
BFTYP_UNKNOWN	Unknown buffer type.
BFTYP_UNIX	Unix, LF line termination.
BFTYP_DOS	DOS, CF/LF line termination.
BFTYP_MAC	Old style MAX, CR termination.
BFTYP_BINARY	Binary.
BFTYP_ANSI	ANSI.
BFTYP_EBCDIC	EBCDIC.
BFTYP_UTF8	UTF8.
BFTYP_UTF16	UTF16/USC2.
BFTYP_UTF32	UTF32/USC4.
BFTYP_UTFEBCDIC	UTF8/EBCDIC.
BFTYP_BOCU1	Binary Ordered Compression for Unicode.
BFTYP_SCSU	Standard Compression Scheme for Unicode.
BFTYP_UTF7	7-bit Unicode Transformation Format.
BFTYP_GB	GB.
BFTYP_BIG5	BIG5.
BFTYP_ISO2022	ISO-2022.
BFTYP_SBCS	Single Byte.
BFTYP_DBCS	Double Byte.
BFTYP_MBCS	Multi-Byte (Non Unicode).
BFTYP_OTHER	Other supported.
BFTYP_UNSUPPORTED	Known file-type, yet no internal support.

Returns

The `set_buffer_type()` primitive returns the 0 on success , otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[inq_buffer_type](#)

set_encoding

```
int set_encoding( [string encoding = NULL],
                  [int bufnum = NULL] )
```

Set a buffers character encoding.

Description

The `set_encoding()` primitive sets or clears the character encoding associated with the referenced buffer.

The following table lists the character encodes which maybe available dependent on build options and system support.

Name	Buffer Type	Code Page	Description
US-ASCII	BFTYP_SBCS	646	ANSI/ASCII
ISO-8859-1	BFTYP_SBCS	28591	Western Europe
ISO-8859-2	BFTYP_SBCS	28592	Western and Central Europe
ISO-8859-3	BFTYP_SBCS	28593	Western Europe and South European (Turkish, Maltese plus Esperanto)
ISO-8859-4	BFTYP_SBCS	28594	Western Europe and Baltic countries (Lithuania, Estonia and Lapp)
ISO-8859-5	BFTYP_SBCS	28595	Cyrillic alphabet

Name	Buffer Type	Code Page	Description
ISO-8859-6	BFTYP_SBCS	28596	Arabic
ISO-8859-7	BFTYP_SBCS	28597	Greek
ISO-8859-8	BFTYP_SBCS	28598	Hebrew
ISO-8859-9	BFTYP_SBCS	28599	Western Europe with amended Turkish character set
ISO-8859-10	BFTYP_SBCS		Western Europe with rationalised character set for Nordic languages
ISO-8859-13	BFTYP_SBCS	28603	Baltic languages plus Polish
ISO-8859-14	BFTYP_SBCS		Celtic languages (Irish Gaelic, Scottish, Welsh)
ISO-8859-15	BFTYP_SBCS	28605	Euro sign and other rationalisations to ISO 8859-1
ISO-8859-16	BFTYP_SBCS		Central, Eastern and Southern European languages
CP037	BFTYP_EBCDIC	37	EBCDIC-US
CP038	BFTYP_EBCDIC	38	EBCDIC-INT
CP930	BFTYP_EBCDIC	930	
CP1047	BFTYP_EBCDIC	1047	
UTF-8	BFTYP_UTF8	65001	
UTF-16	BFTYP_UTF16		
UTF-16be	BFTYP_UTF16	1201	
UTF-16le	BFTYP_UTF16	1200	
UTF-32	BFTYP_UTF32		
UTF-32be	BFTYP_UTF32		
UTF-32le	BFTYP_UTF32		
BOCU-1	BFTYP_BOITU1		
SCSU	BFTYP_SCSU		
UTF-7	BFTYP_UTF7	65002	
UTF-2	BFTYP_UCS2		BFTYP_UTF16 aliases
UTF-2be	BFTYP_UCS2		
UTF-2le	BFTYP_UCS2		
UTF-4	BFTYP_UCS4		BFTYP_UTF32 aliases
UTF-4be	BFTYP_UCS4		
UTF-4le	BFTYP_UCS4		
cp437	BFTYP_SBCS	437	OEM/US, ASCII
cp737	BFTYP_SBCS	737	Greek, ISO-8859-7
cp775	BFTYP_SBCS	775	Baltic
cp850	BFTYP_SBCS	850	Like ISO-8859-4
cp852	BFTYP_SBCS	852	Like ISO-8859-1
cp855	BFTYP_SBCS	855	Like ISO-8859-2
cp857	BFTYP_SBCS	857	Like ISO-8859-5
cp860	BFTYP_SBCS	860	Like ISO-8859-9
cp861	BFTYP_SBCS	861	Like ISO-8859-1
cp862	BFTYP_SBCS	862	Like ISO-8859-1
cp863	BFTYP_SBCS	863	Like ISO-8859-8
cp865	BFTYP_SBCS	865	Like ISO-8859-1
cp866	BFTYP_SBCS	866	Like ISO-8859-5
cp869	BFTYP_SBCS	869	Greek, like ISO-8859-7
cp874	BFTYP_SBCS	874	Thai
cp1046	BFTYP_SBCS	1046	Arabic DOS code
windows-1250	BFTYP_SBCS	1250	Central European languages that use Latin script (Polish, Czech etc).
windows-1251	BFTYP_SBCS	1251	Cyrillic alphabets
windows-1252	BFTYP_SBCS	1252	Western languages
windows-1253	BFTYP_SBCS	1253	Greek
windows-1254	BFTYP_SBCS	1254	Turkish
windows-1255	BFTYP_SBCS	1255	Hebrew
windows-1256	BFTYP_SBCS	1256	Arabic

Name	Buffer Type	Code Page	Description
windows-1257	BFTYP_SBCS	1257	Baltic languages
windows-1258	BFTYP_SBCS	1258	Vietnamese
Mac-Arabic	BFTYP_SBCS		
Mac-Celtic	BFTYP_SBCS		
Mac-Centeuro	BFTYP_SBCS		
Mac-Croatian	BFTYP_SBCS		
Mac-Cyrillic	BFTYP_SBCS		
Mac-Devanaga	BFTYP_SBCS		
Mac-Dingbats	BFTYP_SBCS		
Mac-Farsi	BFTYP_SBCS		
Mac-Gaelic	BFTYP_SBCS		
Mac-Greek	BFTYP_SBCS		
Mac-Gujarati	BFTYP_SBCS		
Mac-Gurmukhi	BFTYP_SBCS		
Mac-Hebrew	BFTYP_SBCS		
Mac-Iceland	BFTYP_SBCS		
Mac-Inuit	BFTYP_SBCS		
Mac-Roman	BFTYP_SBCS		
Mac-Romanian	BFTYP_SBCS		
Mac-Thai	BFTYP_SBCS		
Mac-Turkish	BFTYP_SBCS		
cp10000	BFTYP_SBCS	10000	MacRoman
cp10006	BFTYP_SBCS	10006	MacGreek
cp10007	BFTYP_SBCS	10007	MacCyrillic
cp10029	BFTYP_SBCS	10029	MacLatin2
cp10079	BFTYP_SBCS	10079	MacIcelandic
cp10081	BFTYP_SBCS	10081	MacTurkish
KOI8-R	BFTYP_SBCS	20866	Russian, using cynrillic alphabet.
KOI8-U	BFTYP_SBCS	21866	Ukrainian, using cynrillic alphabet.
KOI8-T	BFTYP_SBCS		Ukrainian
PT154	BFTYP_SBCS		Ukrainian
KOI7	BFTYP_SBCS		Ukrainian
MIK	BFTYP_SBCS	0	Bulgarian
ISCII	BFTYP_SBCS		Indian Script Code for Information Interchange.
TSCII	BFTYP_SBCS		Tamil Script Code for Information Interchange.
VSCII	BFTYP_SBCS		Vietnamese Standard Code for Information Interchange.
DEC-MCS	BFTYP_SBCS	-2	
DEC-KANJI	BFTYP_SBCS	-2	
DEC-HANYU	BFTYP_SBCS	-2	
HP-Roman8	BFTYP_SBCS	-3	
HP-Arabic8	BFTYP_SBCS	-3	
HP-Greek8	BFTYP_SBCS	-3	
HP-Hebrew8	BFTYP_SBCS	-3	
HP-Turkish8	BFTYP_SBCS	-3	
HP-Kana8	BFTYP_SBCS	-3	
GB2312	BFTYP_GB		Guojia Biaozhun/Simplified Chinese.
GBK	BFTYP_GB	936	Chinese/GB (CP936).
GB18030	BFTYP_GB		Chinese National Standard/GB.
HZ	BFTYP_HZ		RFC1843, Arbitrarily Mixed Chinese and ASCII.
Big5	BFTYP_BIG5	950	Chinese/Big-5 (CP950).
Big5-5E	BFTYP_BIG5		Big-5.
Big5-2003	BFTYP_BIG5		Big-5.
Big5-HKSCS	BFTYP_BIG5		Big-5/Hong Kong Supplement.
Shift_JIS	BFTYP_MBCS		Shift JIS.
EUC-JP	BFTYP_MBCS		Japan/EUC.
CP932	BFTYP_MBCS	932	Windows-31J.
EUC-CN	BFTYP_MBCS		Chinese/EUC.

Name	Buffer Type	Code Page	Description
EUC-TW	BFTYP_MBCS		Tawian/EUC.
EUC-KR	BFTYP_MBCS	949	Korean/EUC (CP949).
ISO-2022-CN	BFTYP_ISO2022		
ISO-2022-KK	BFTYP_ISO2022		
ISO-2022-KP	BFTYP_ISO2022		

Parameters

encoding Optional encoding name, if omitted the encoding is derived from the buffer type (See: [set_buffer_type](#)).

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

nothing

Portability

A **GriefEdit** extension

See Also

[inq_encoding](#), [inq_buffer_type](#)

set_indent

```
int set_indent([int indent],
              [int bufnum] )
```

Set the buffers default indentation.

Description

The `set_indent()` primitive configures the indentation value for the specified buffer, representing the buffers default ruler. Indentation stops are set every `indent` stops after the last stop, with the first column within a line being column 1.

Indenting does not change the size represented by physical tabbing, it determines the buffers default indentation when a tab-character is `self_inserted()`, backfilling with either spaces and/or physical tabs dependent on whether or not hard=tabs are enabled (See: [use_tab_char](#)).

An indent value of 0, shall disable the buffers indentation setting defaulting to the current tab stop (See: [tabs](#)) unless a ruler is also in effect. If omitted the user shall be prompted for a new value as follows:

Enter indent amount:

Note that any user specified ruler (See: [set_ruler](#)) shall have priority over both this setting and the tabs configuration.

Parameters

indent Optional buffer indentation, if omitted the user shall be prompted.

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The `set_indent()` primitive returns the applied indentation value, otherwise if the user was prompted and they aborted -1 is returned.

Portability

A **GriefEdit** extension.

See Also

[inq_indent](#), [set_ruler](#), [tabs](#)

set_line_flags

```
int set_line_flags( [int bufnum],
                    [int start],
                    [int end],
                    [int and_mask],
                    [int or_value] )
```

Associate line flags.

Description

The `set_line_flags` primitive allows the flags of one or more line within a specific buffer to be modified.

The buffer flags was a set of 32 bit values separated into two namespaces, with the upper 8 bits being defined for user/macro usage and lower bits for system user/macro usage. As such only the lower 16 bits maybe affected by this primitive.

The defines `L_USER1` thru `L_USER7` maybe used as manifest constants to access the user/macro area.

Parameters

<code>bufnum</code>	Buffer number, if omitted the current buffer is referenced.
<code>start</code>	Start line number of region within the selected buffer, otherwise if omitted the current line is referenced.
<code>end</code>	End of the region within the selected buffer, otherwise if omitted the current line is referenced as such one line shall be affected.
<code>and_mask</code>	Value AND'ed with the flags of matched lines.
<code>or_value</code>	Value OR'ed with the flags of matched lines.

Compatibility

GriefEdit enforces two flag namespaces system and user each of 16 bits with only the lower user 16 bits being read-write, whereas *CRISP™* allows read-write to all 32 bits.

Returns

The `set_line_flags` primitive returns the number at lines which were modified.

see Also

[inq_line_flags](#), [find_line_flags](#)

set_margins

```
int set_margins( [int bufnum],
                  [int left = NULL],
                  [int right = NULL],
                  [int style = NULL],
                  [int colorcolumn = NULL])
```

Set buffer formatting margins.

Description

The `set_margins()` primitive configures one or more of the specified buffers `bufnum` margins.

Parameters

<code>bufnum</code>	Optional buffer number, if omitted the current buffer shall be referenced. A negative bufnum (e.g. -1) shall set the global margin parameters, which are applied when no buffer specific margin has been set.
<code>left</code>	Optional integer left margin. A non-positive value shall clear the buffer specific margin.
<code>right</code>	Optional integer right margin. A non-positive value shall clear the buffer specific margin.
<code>style</code>	Optional justification style.
<code>colorcolumn</code>	Optional colour column.

Returns

The `set_margins()` primitive returns 0 on success otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[inq_margins](#)

set_process_position

```
int set_process_position([int line],
                        [int column])
```

Set process insertion position.

Description

The `set_process_position()` primitive sets the line and/or column associated with the input from a subprocess.

Processes maintain their own independent input in the buffer so that it is easier to write macros which manipulate subprocesses.

Parameters

`line` Optional integer specifying the line number, if positive the cursor is set to the specified line.

`column` Optional integer specifying the column number, if positive the cursor is set to the specified column.

Returns

The `set_process_position()` primitive returns 0 on success, otherwise -1 if the current buffer is not attached to a process.

Portability

n/a

See Also

[inq_process_position](#), [connect](#)

set_ruler

```
int set_ruler([int bufnum],
              [list|string|int ...])
```

Configure the buffer ruler.

Description

The `set_ruler()` primitive configures the indentation ruler of the current buffer to the positions specified within *ruler*.

The primitive supports a number of alternative specification forms being either a set of integer parameters, a single string parameter containing space/comma separated numbers or a single list of integers. If omitted the ruler is cleared.

Regardless of the form each should be a sequence of columns in ascending order. The indentations for the remainder of the line are set using the difference between the last two stated positions, starting at the last specified.

Parameters

`bufnum` Optional buffer number, if omitted the current buffer shall be referenced.

`ruler` Optional ruler specification, being the sequence of columns in ascending order otherwise the ruler is cleared.

Returns

The `set_ruler()` primitive returns the number of applied ruler points, 0 is the ruler was cleared otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[tabs](#), [inq_ruler](#)

set_tab

```
int set_tab([int increment],
            [int bufnum]      )
```

Derive the buffer tab stops.

Description

The `set_tab()` primitive derives a `tabs` configuration from the specified tab increment *increment*. If omitted the user shall be prompted for the tab increment.

Enter tab amount:

Parameters

- increment Optional positive integer, stating the tab increment if omitted the user shall be prompted.
- bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

Returns the applied tab increment, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[tabs](#)

set_terminator

```
int set_terminator( [int bufnum],
                    int|string term )
```

Set a buffers line terminator.

Description

The `set_terminator()` primitive retrieves the line terminator of the specified buffer `bufnum`.

Parameters

- bufnum Optional buffer number, if omitted the current buffer shall be referenced.
- term Either the integer enumeration or string description of the line terminator to be assigned.

Enumerations

Constant	Description
LTERM_UNDEFINED	Unknown/default.
LTERM_NONE	<none> (i.e. binary)
LTERM_UNIX	CR/LF
LTERM_DOS	LF
LTERM_MAC	CR
LTERM_NEL	NEL
LTERM_UCSNL	Unicode next line
LTERM_USER	User defined

Returns

The `set_terminator()` primitive returns 1 if the line terminator was modified, 0 when no change occurred, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[inq_terminator](#)

sort_buffer

```
int sort_buffer( [int bufnum],
                 [string|int comparator = 0],
                 [int start],
                 [int end],
                 [int type = 3] )
```

Sort buffer content.

Description

The `sort_buffer()` primitive sorts the lines in the current buffer or the buffer specified by `bufnum`. If the buffer specified has a region marked, then only those lines within the region are sorted.

By default lines are sorted alphabetically yet the sort can be modified using the user specified macro or using one of the predefined system sort macros.

Parameters

bufnum	Optional buffer number, if omitted the current buffer shall be referenced.
comparator	Optional string comparison macro or integer direction. A string value states the comparison macro to be executed. Whereas an integer value states the direction, with zero selecting the built "sort_buffer::forward" comparator and a non-zero value selecting "sort_buffer::backwards". If omitted the comparator defaults to forward.
start	Optional integer line number stating the start of the region to be sorted, if omitted the buffer top is used unless a marked region is active.
end	Optional integer line number stating the end of the region to be sorted, if omitted the buffer end is used unless a marked region is active.
type	Optional integer stating the sort type, being
1	quicksort.
2	mergesort
3	heapsort (default).

Returns

The `sort_buffer()` primitive returns the number of lines sorted, otherwise a negative value on error.

Portability

Second argument allowing either a sort-order or user specified callback plus type selection are **GriefEdit** extensions.

See Also

[sort_list](#)

tabs

```
int tabs([string tabs | list tabs | int tab, ...])
```

Set buffer tab stops.

Description

The `tabs()` primitive configures the tabs of the current buffer to the positions specified within `tabs`.

The primitive supports a number of alternative specification forms being a set of integer parameters, a single string parameter containing space/comma separated numbers or a single list of integers. If omitted the user shall be prompted for each of the tab points, with an empty reply terminating the sequence as follows:

Enter tab stop (return terminates):

Regardless of the form each should be a sequence of columns in ascending order. Tabs for the remainder of the line are set using the difference between the last two `tabs` stated, starting at the last specified.

Example

The following sets the first tab at four spaces and all sequence `tabs` to three resulting in the `tabs` at (5, 8, 11, 14 ...)

```
tabs(5, 8);
```

As the tab primitive allows a number of specification forms, all the following are equivalent;

```
tabs("5 8");
tabs("5,8");
list ttabs = {5, 8};
tabs(ttabs);
```

Parameters

'`tabs`' Optional `tabs` specification, being the sequence of columns in ascending order otherwise the user is prompted.

Returns

The `tabs()` primitive returns the number of applied tab points otherwise if the user was prompted and they aborted -

1 is returned.

Portability

BRIEF limited the number of unique tab stops at 8, under **GriefEdit** this limit is 80.

See Also

[inq_tabs](#), [set_indent](#), [distance_to_tab](#), [distance_to_indent](#)

tagdb_close

```
int tagdb_close(int handle)
```

Tag database close.

Description

The *tagdb_close()* primitive closes a tag database handle, so that it no longer refers to any resources and may be reused.

Parameters

handle Tag database handle.

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[tagdb_open](#), [tagdb_search](#)

tagdb_open

```
int tagdb_open( string file,
                [int options],
                [int background] )
```

Tag database open.

Description

The *tagdb_open()* primitive given a pathname for a ctags database, returns a handle being a non-negative integer for use in subsequent database search operations using [tagdb_search](#). The tag database handle shall remain open until <*tagdb_close*> is executed against the handle.

ctags is a tool which permit easy navigation thru a large set of source files. ctags supports many languages including c, c++ and Java just to name a few.

Note:

GriefEdit relies on an external tag file generator. There are many versions of ctags; however, the recommended version is "Exuberant Ctags" available from
<http://ctags.sourceforge.net/>.

GriefEdit is generally bundled with a recent version within the bin installation folder as *extags*. Therefore, you would not need to download/install a tag binary to use this feature.

Parameters

- *TAG_ETAGS*
- *TAG_CTAGS*

file tag database path.

options Optional integer flags, being one or more of the following constants OR'ed together forming open options.

background Optional integer boolean value, if **true** the database loading shall be moved into the background.

Returns

The *tagdb_open()* primitive returns the new database descriptor, otherwise -1 if an error occurred.

Portability

A **GriefEdit** extension.

See Also

[tagdb_search](#), [tagdb_close](#)

tagdb_search

```
int tagdb_search( int handle,
                  string word,
                  [int flags])
```

Tag database search.

Description

The *tagdb_search()* primitive searches the tag database for symbols matching *pattern*.

Note:

Consult the *tags* macro source for an example.

Parameters

handle Tag database handle.
pattern String containing the search pattern.
flags Optional integer flags.

Returns

The *tagdb_search* returns a list containing the search results, otherwise a NULL list on error or no match was found.

Portability

A **GriefEdit** extension.

See Also

[tagdb_open](#), [tagdb_close](#)

write_buffer

```
int write_buffer( [string filename],
                  [int flags],
                  [string encoding] )
```

Write to buffer content.

Description

The *write_buffer()* primitive writes the content of the current buffer to its associated file.

Parameters

filename Options string containing the name of the output filename. If omitted then the file is written to the name associated with the current buffer during creation using <>create_buffer>.
flags Optional integer flags, one or more of the following flags OR'ed together control the functions of the write operation.
encoding Optional string containing the character encoding to be utilised within the output file.

Flags

Constant	Description
WRITE_APPEND	Append, otherwise overwrite.
WRITE_NOTRIGGER	Do not generate buffer triggers.
WRITE_NOREGION	Ignore any selected region.
WRITE_FORCE	Force write, even if no change.
WRITE_BACKUP	Generate a backup image regardless whether already performed for this edit session.

Returns

- Returns greater than zero on success.
- Returns zero if file was not saved, eg. because the file has already been saved.
- Returns less than zero if an error occurs.

Value	Description
-1	Disk space occurred.
-2	Output file could not be created.
-3	The output file was created with a different temporary name but could not be renamed to the target file due to permission errors.
-4	User aborted the attempt to save the file from one of the callback triggers.
-5	The output buffer does not have a valid filename.
-6	The originally loaded file has changed its permissions, size or status on disk. This option avoids potentially losing work when someone else has written to the file whilst we were editing it.
-7	The file is read-only, either due to file writes having been disabled by the command line switch (-R) or the current file permissions.

In many cases the underlying cause of the error condition can be derived from the system return code (See: [errno](#)), for example out of disk space.

Portability

Flags are incompatible with CrispEdit™

```
write_buffer([string filename], [int and_flags], [or_flags])
```

See Also

[edit_file](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Callbacks

Summary

Callbacks

Macros

<code>_bad_key</code>	Command prompt unknown key callback.
<code>_chg_properties</code>	Property change event.
<code>_default</code>	Default extension handler.
<code>_extension</code>	Buffer load handler.
<code>_fatal_error</code>	Fatal condition callback.
<code>_init</code>	Internal macro initialisation.
<code>_invalid_key</code>	Invalid key event.
<code>_prompt_begin</code>	Command prompt session begin callback.
<code>_prompt_end</code>	Command prompt session end callback.
<code>_startup_complete</code>	Startup event callback.
<code>main</code>	Macro entry point.

Macros

`_bad_key`

```
string _bad_key()
```

Command prompt unknown key callback.

Description

The `_bad_key()` callback is executed by GRIEF upon an unknown key being pressed during a command prompt session.

Parameters

none

Returns

The `_bad_key()` callback should return the replacement string otherwise NULL.

Portability

n/a

See Also

[get_parm](#), [_prompt_begin](#), [_prompt_end](#), [inq_command](#), [inq_cmd_line](#)

`_chg_properties`

```
void _chg_properties()
```

Property change event.

Description

The `_chg_properties` callback is executed by **GriefEdit** upon the borders configuration changing.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[Callbacks](#), [borders](#)

_default

```
void _default(string ext)
```

Default extension handler.

Description

The `_default()` callback is executed whenever GRIEF edits a file via the `edit_file` primitive and a extension specific macro of the format `_ext` was not available to be executed.

It is provided to allow macros to hook buffer loads, for example to setup defaults tabs on a file extension basis.

The extension case shall be preserved on case sensitive file-systems otherwise the extension is converted to lower case.

Note:

This interface is considered defunct; `register_macro` is the preferred method of capturing buffer edit events.

Parameters

`ext` File extension.

Returns

nothing

Portability

n/a

See Also

`edit_file`, `_extension`

_extension

```
void _extension(string ext)
```

Buffer load handler.

Description

The `_extension()` callback is executed whenever GRIEF edits a file via the `edit_file` primitive.

It is provided to allow macros to hook buffer loads, for example to setup defaults tabs before any extension specific settings are applied.

Once executed if defined the extension specific handler shall be executed, which should be named as `_ext`. If not available the default extension `_default` handler is executed.

The extension case shall be preserved on case sensitive file-systems otherwise the extension is converted to lower case.

Note:

This interface is considered defunct; `register_macro` is the preferred method of capturing buffer edit events.

Parameters

`ext` File extension.

Returns

nothing

Portability

The `ext` parameter is a **GriefEdit** extension.

See Also

`edit_file`, `_default`

fatal_error

```
void _fatal_error(int      signo,
                  string desc )
```

Fatal condition callback.

Description

The `_fatal_error` callback is executed by **GriefEdit** upon a fatal signal condition being thrown by the operating system; these generally represent editor bugs or possible macro usage violations.

The list of trapped condition is very system dependent, yet the following are generally included.

Constant	Description
SIGBUS	Bus error
SIGSEGV	Segmentation violation.
SIGILL	Illegal instruction.
SIGFPE	Floating point error.

Note:

The macro implementation should utilise as few **GriefEdit** facilities as possible to avoid re-triggering the same condition.

Refer to the `core` macro for an example.

Parameters

`signo` Integer signal number.
`desc` String containing the signal description.

Returns

The `_start_complete` should return nothing

Portability

n/a

See Also

[Callbacks](#)

init

```
void _init()
```

Internal macro initialisation.

Description

The `_init` macro is utilised by the Compiler to set-up file level variables (See: [global](#)).

Note:

This interface is an internal primitive, reserved for exclusive use by the GRIEF Macro Compiler and may change without notice. Management of variable scoping and argument binding shall be handled automatically by the compiler.

Returns

nothing

Portability

n/a

See Also

[main](#), [global](#), [load_macro](#)

_invalid_key

```
void _invalid_key()
```

Invalid key event.

Description

The `_invalid_key` callback is a legacy BRIEF interface.

This interface is provided for BRIEF compatibility, if the macro `_invalid_key` exists, it shall be called instead of any registered macros. When invoked the keystroke awaiting is the invalid key.

Parameters

none

Returns

The `_invalid_key` callback should return an integer value.

Portability

BRIEF compatibility.

See Also

[register_macro](#)

_prompt_begin

```
void _prompt_begin(string prompt)
```

Command prompt session begin callback.

Description

The `_prompt_begin()` callback is executed at the beginning of a command prompt session.

The interface allows the command line history and abbreviations mechanisms to be implemented.

Parameters

`prompt` String containing the prompt.

Returns

nothing

Portability

n/a

See Also

[get_parm](#), [_prompt_end](#), [_bad_key](#), [inq_command](#), [inq_cmd_line](#)

_prompt_end

```
void _prompt_end()
```

Command prompt session end callback.

Description

The `_prompt_end()` callback is executed at the termination of a command prompt session.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[get_parm](#), [_prompt_begin](#), [_bad_key](#), [inq_command](#), [inq_cmd_line](#)

_startup_complete

```
void _startup_complete(int mode)
```

Startup event callback.

Description

The `_startup_complete()` callback is executed by GRIEF upon startup.

It is executed after all command line switches have been process, after all command line files have been read in, and after the `startup()` macro has been called.

The specified *mode* states the command line status as follows.

- 0 No command line files were specified.
- 1 Command line file were specified.

This callback is utilised by the restore macro to avoid restoring the state of buffers and files when files have been specified on the command line.

Parameters

`mode` Command line operational mode.

Returns

The `_start_complete()` callback should return the nothing.

Portability

n/a

See Also

[Callbacks](#)

main

```
void main()
```

Macro entry point.

Description

The `main` macro is an optional callback which may be defined within each macro source module.

Upon a macro load, the macro `main` is executed if it is present within the macro object. This macro can be used to initialise and perform load-time specific variable initialisation and/or functions.

Note:

The `main` function is indirectly executed on the successful loading of a macro object by the `_init` entry point.

Returns

nothing

Portability

n/a

See Also

[_init, load_macro](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Debugging Primitives

Summary

Debugging Primitives

Macros

<code>abort</code>	Abnormal process.
<code>debug</code>	Control runtime debug and tracing.
<code>debug_support</code>	Internal debugger functionality.
<code>dprintf</code>	Formatted diagnostics output.
<code>error</code>	Issue an error message on the command line.
<code>inq_debug</code>	Retrieve the current debug setting.
<code>message</code>	Display a message on the command line.
<code>pause_on_error</code>	Set the error display mode.
<code>pause_on_message</code>	Set the message display mode.
<code>printf</code>	Print formatted string to stdout.
<code>profile</code>	Profiling support.
<code>watch</code>	Watch a symbol.

Macros

abort

```
void abort()
```

Abnormal process.

Description

The `abort()` primitive immediately exits **GriefEdit**.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[exit](#), [process](#)

debug

```
int debug([int flags|string flags|NULL],  
         [int echo = TRUE] )
```

Control runtime debug and tracing.

Description

The `debug()` primitive enables and disables run-time tracing of macros and system acts to the diagnostics log. The new debug flags shall be set to *flags*.

If *flags* is omitted, then the current trace mode is toggled from *off* to *DB_TRACE(1)* and from *on* to *off*.

Otherwise by specifying *flags* the associated events which shall be traced during subsequent macro operations. Flags can either be one or more **DB_XXX** constants OR'ed together or string containing a comma separated list of flag names, see the table below. If the given flag value is either 0 or an empty string, all trace operations are disabled.

The *echo* flags controls whether the user is informed of the debug state change. When omitted or a non-zero any changes are echoed on the command prompt, informing the user of the new debug status, otherwise the macro is silent.

Flags

The following *debug* flags are available, enabling tracing of both general and specific macro operations:

Constant	Name	Description
DB_TRACE	trace	General trace.
DB_REGEXP	regexp	Regular expression compilation and execution.
DB_UNDO	undo	Undo/red trace.
DB_FLUSH	flush	Flush output, shall cause the trace log to be flushed on each write; use should be avoided unless catching crash conditions due to associated high cost.
DB_TIME	time	Time stamp which trace log.
DB_TERMINAL	terminal	Terminal interface.
DB_VFS	vfs	Virtual file-system.
DB_NOTRAP	notrap	Disable SIGBUS/SIGSEGV trap handling.
DB_MEMORY	memory	Target specific <i>debug</i> memory service
DB_REFS	refs	Variable references.
DB_PROMPT	prompt	Command prompting.
DB_PURIFY	purify	Running under a memory analysis tool;
DB_HISTORY	history	Command history.

Command lines Options;

The following command option also effect the generation of diagnostics trace.

-d, --log	Enable trace (DB_TRACE)
-f, --flush	Enable log flushing (DB_FLUSH).
--nosig	Disable signal traps (DB_NOSIG).
-P, --dflags=x	Debug/profiling flags x, being a comma separated list of flag names, as defined above.
--rtc	Enable real-time memory checks, if available.
--native	Enable native memory allocator, if suitable.

Diagnostic Log

The default name of the diagnostics log is system dependent

.grief.log Unix and Unix systems, including Linux.
grief.log WIN32 and DOS based systems.

Environment Variables

GRIEF_LOG If the GRIEF_LOG variable exists within the environment then it overrides the default trace log name.

Returns

The *debug()* primitive returns the value of the *debug* flags prior to any changes, allowing the caller to restore later.

Example

Enable *debug*, invoke our macro for testing and then restore the previous flags.

```
int odebug = debug(1, FALSE);
myfunction();
debug(odebug, FALSE);
```

Portability

The *echo* option is a **GriefEdit** extensions.

See Also

[inq_debug](#), [dprintf](#), [pause_on_error](#), [error](#), [message](#)

debug_support

```
void debug_support(int what,
                  declare object,
                  declare arg )
```

Internal debugger functionality.

Description

The `debug_support()` primitive supports GRIEF debug functions.

The primitive is a switcher to the actual sub-functions based upon *what*, the arguments *object* and *arg* are what specific.

Returns

nothing

See Also

`debug`, `inq_debug`

dprintf

```
int dprintf(string format,
            ...      )
```

Formatted diagnostics output.

Description

This primitive is used to write a diagnostic messages to the trace log when debug is enabled.

The `dprintf()` primitive produces formatted output according to the format specification *format* into the diagnostics log. The trailing arguments ... are the integer, string or list expressions used to satisfy the % formatting options; refer to (See: [message](#)) for details on the supported format specifications.

The format primitive is similar to the C `fprintf()` primitive, exporting the formatted result to an external stream.

This function is one of a set of formatted output primitives each supporting a common feature set, see [message](#) for a complete discussion on the supported format specification syntax.

Parameters

- `format` String that contains the text to be written. It can optionally contain an embedded <Format Specification> that are replaced by the values specified in subsequent additional arguments and formatted as requested.
- `...` Optional format specific arguments, depending on the format string, the primitive may expect a sequence of additional arguments, each containing a value to be used to replace a format specifier in the format string.
There should be at least as many of these arguments as the number of values specified in the format specifiers. Additional arguments are ignored by the primitive.

Returns

The `dprintf()` primitive returns the number of charaters stored within the result string *buffer*.

Portability

This is a **GriefEdit** extension.

See Also

`debug`, `error`, `message`

error

```
int error(string format,
          ...      )
```

Issue an error message on the command line.

Description

The `error()` primitive is used to display a message on the prompt line at the bottom of the screen. *format* is a string and may contain printf-like % formatting characters. The following arguments *format* and so forth are integer or string expressions used to satisfy the % formatting options.

This primitive may be used to error messages on the bottom of the screen; if informational messages are required to be displayed then the `message` primitive macro should be used instead.

Error messages are printed in the error color (See: `color`) and truncated if they are too long.

In addition, upon enabling `pause_on_error` then the error message is displayed suffixed with a .. and **GriefEdit** waits for the user to type any key to continue; useful during debugging.

This function is one of a set of formatted output primitives each supporting a common feature set, see [message](#) for a complete discussion on the supported format specification syntax.

Parameters

- format** String that contains the text to be written. It can optionally contain an embedded <Format Specification> that are replaced by the values specified in subsequent additional arguments and formatted as requested.
- ... Optional format specific arguments, depending on the format string, the primitive may expect a sequence of additional arguments, each containing a value to be used to replace a format specifier in the format string.
There should be at least as many of these arguments as the number of values specified in the format specifiers. Additional arguments are ignored by the primitive.

Returns

The *message()* primitive returns the number of characters printed.

Example

```
message("File: %s", current_name);
```

Portability

Many of the format options are **GriefEdit** extensions, including *%b*, *%n*, *%t*, *%B* and * support.

The original implementation has a void declaration and returns nothing.

The standard interface only permits upto 6 arguments, whereas **GriefEdit** is unbound.

See Also

[message](#), [pause_on_error](#)

inq_debug

```
int inq_debug()
```

Retrieve the current debug setting.

Description

The *inq_debug()* primitive retrieves the current debug flags.

Returns

The *inq_debug()* primitive returns an integer representing the debug flags which are currently in effect.

Example

Enable debug of regular expressions, invoke our macro for testing and then restore the previous flags.

```
int odebug = inq_debug();
debug(odebug | DB_REGEXP, FALSE);
myfunction();
debug(odebug, FALSE);
```

See Also

[debug](#), [dprintf](#), [pause_on_error](#), [error](#), [message](#)

message

```
int message(string format,
           ...      )
```

Display a message on the command line.

Description

The *message()* primitive is used to display a message on the prompt line at the bottom of the screen. *format* is a string and may contain printf-like % formatting characters. The following arguments *format* and so forth are integer or string expressions used to satisfy the % formatting options.

This primitive may be used to display informational messages on the bottom of the screen; if error messages are to be displayed then the [error](#) primitive macro should be used instead.

Parameters

- `format` String that contains the text to be written. It can optionally contain an embedded <Format Specification> that are replaced by the values specified in subsequent additional arguments and formatted as requested.
- `...` Optional format specific arguments, depending on the format string, the primitive may expect a sequence of additional arguments, each containing a value to be used to replace a format specifier in the format string.
There should be at least as many of these arguments as the number of values specified in the format specifiers. Additional arguments are ignored by the primitive.

Format Specification

A format specification, which consists of optional and required fields, has the following form:

```
%[FLAGS] [WIDTH] [.PRECISION] [PREFIXES] type
```

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a type character (for example, %s). If a percent sign is followed by a character that has no meaning as a format field, the character is simply copied. For example, to print a percent-sign character, use %%.

The optional fields, which appear before the type character, control other aspects of the formatting, as follows:

- `type` Required character that determines whether the associated argument is interpreted as a character, a string, or a number.
- `flags` Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.
- `width` Optional number that specifies the minimum number of characters output.
- `precision` Optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.

The following format types are supported;

Type	Description
%d	Print an integer expression. Options such as %03d can be used to perform zero insertion and supply a field width.
%i	Print an integer expression, same as %d.
%u	Print an unsigned expression.
%x	Print integer expression in hex.
%t	Print using the symbol natural type (ie d = integers and s = strings).
%o	Print integer expression in octal.
%s	Print a string. Field width and alignments are allowed.
%c	Print a character.
%p	If its less than a second since the last call to <i>message</i> (or <i>error</i>), then don't display the <i>message</i> . This is used by macros which want to print progress messages.
%e	Double signed value having the form [-]d.dddd e[sign]ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or .
%E	Double identical to the e format except that E rather than e introduces the exponent.
%f	Double signed value having the form []dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
%g	Double signed value printed in f or e format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than 4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
%G	Double identical to the g format, except that E, rather than e, introduces the exponent (where appropriate).
%n	Number of characters successfully written so far to the stream or buffer; this value is stored in the integer variable given as the argument.
%b	Print integer expression in binary.

Type	Description
%B	Print integer expression as a decoded binary, using the bit values as described within the secondary string parameter of the form; <base><arg> Where <base> is the output base expressed as a control character, e.g. \10 gives octal; \20 gives hex. Each arg is a sequence of characters, the first of which gives the bit number to be inspected (origin 1), and the next characters (up to a control character, i.e. a character <= 32), give the name of the register. Thus:

Flags

The following flags are supported.

Flag	Description	Default
-	Left align the result within the given field width.	Right align.
+	Prefix the output value with a sign (+ or -) if the output value is of a signed type.	Sign appears only for negative signed values () .
0	If width is prefixed with 0, zeros are added until the minimum width is reached. If 0 and appear, the 0 is ignored. If 0 is specified with an integer format (i, u, x, X, o, d) the 0 is ignored.	No padding.
<space>	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear.	No blank appears.
#	When used with the o, x, or X format, the # flag prefixes any nonzero output value with 0, 0x, or 0X, respectively. When used with the e, E, or f format, the # flag forces the output value to contain a decimal point in all cases. When used with the g or G format, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. Ignored when used with c, d, i, u, or s.	No blank appears. Decimal point appears only if digits follow it. Decimal point appears only if digits follow it. Trailing zeros are truncated.

Prefixes

The following prefixes are supported.

Prefix	Description
l	User long format for printing, e.g. floats are printed with full double-precision.

Width

The second optional field of the format specification is the width specification. The width argument is a nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters in the output value is less than the specified width, blanks are added to the left or the right of the values depending on whether the flag (for left alignment) is specified until the minimum width is reached. If width is prefixed with 0, zeros are added until the minimum width is reached (not useful for left-aligned numbers).

width specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if width is not given, all characters of the value are printed (subject to the precision specification).

If the width specification is an asterisk (*), an int argument from the argument list supplies the value. The width argument must precede the value being formatted in the argument list. A nonexistent or small field width does not cause the truncation of a field; if the result of a conversion is wider than the field width, the field expands to contain the conversion result.

Precision

The third optional field of the format specification is the precision specification. It specifies a nonnegative decimal integer, preceded by a period (.), which specifies the number of characters to be printed, the number of decimal places, or the number of significant digits. Unlike the width specification, the precision specification can cause either truncation of the output value or rounding of a floating-point value. If precision is specified as 0 and the value to be converted is 0, the result is no characters output, as shown below:

```
"%.0d", 0 // characters output
```

If the precision specification is an asterisk (*), an int argument from the argument list supplies the value. The precision argument must precede the value being formatted in the argument list.

The type determines the interpretation of precision and the default when precision is omitted, as shown in the following table.

Type	Description	Default
c,C	Character is printed.	
d,i,u,o,x,X	Minimum number of digits to be printed. If the number of digits in the argument is less than precision, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds precision.	Default precision is 1.
e,E	Number of digits to be printed after the decimal point. The last printed digit is rounded.	Default precision is 6; if precision is 0 or the period (.) appears without a number following it, no decimal point is printed.
f	Number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.	Default precision is 6; if precision is 0, or if the period (.) appears without a number following it, no decimal point is printed.
g,G	Maximum number of significant digits printed.	Six significant digits are printed, with any trailing zeros truncated.
s	Maximum number of characters to be printed.	Characters in excess of precision are not printed. Characters are printed until EOS is encountered.

Examples

String output

```
message("See %s %.", "spot", "run");
message("%pMacro completed: %d%", (val*100)/total);
```

Binary output

```
message("val=%B", 3, "\\10\\2BITTWO\\1BITONE\\n")
val=3<BITTWO,BITONE>
```

Return

The `message()` primitive returns the number of characters printed.

Portability

Many of the format options are **GriefEdit** extensions, including `%b`, `%n`, `%t`, `%B` and `*` support.

The original implementation has a void declaration and returns nothing.

The standard interface only permits upto 6 arguments, whereas **GriefEdit** is unbound.

CRisPEdit™ allows the second argument to be a list expression, which emulates a `vsprintf()` style of operation. In this case, the first element of the list is assumed to be a string format descriptor, and subsequent elements of the list are the arguments to print; this feature is not currently supported.

See Also

[error](#), [printf](#), [sprintf](#)

pause_on_error

```
int pause_on_error([int pause = NULL],
                   [int echo = TRUE] )
```

Set the error display mode.

Description

The `pause_on_error()` primitive controls the automatic pausing upon a macro error; by default the *pause status* is off.

On the execution of the `error` primitive plus the internal equivalent, the *pause status* is checked and when enabled error messages shall be displayed with a .. suffix and **GriefEdit** shall await for the user to acknowledge the condition via a key-press before execution continues.

If `pause` is specified as a non-negative (≥ 0) then the current setting is set to the pause value of the integer expression, whereas a negative value (-1) behaves like a inquiry primitive reporting the current value without change. Otherwise if omitted the current status is toggled.

Unless `echo` is stated as **FALSE**, upon modification of the pause state, the new status shall be echoed on the command as follows

Pausing errors on.

or

Pausing errors off.

Parameters

`pause` Optional int flag, when a non-negative value (≥ 0) it states the new pause status, a negative value (-1) reports the current value without change, otherwise if omitted the current pause state is toggled.

`echo` Optional int flag, when stated as **FALSE** the function is silent, otherwise the change in the pause state is echoed on the command prompt.

Returns

The `pause_on_error` function returns the previous pause state.

Portability

`echo` is a **GriefEdit** extension.

Example

The following examples enables error acknowledgement during the execution of the macro `subshell()` and restores the previous status on completion.

```
int state = pause_on_error(TRUE);
subshell();
pause_on_error(status, FALSE);
```

See Also

[error](#), [message](#)

pause_on_message

```
int pause_on_message([int pause = NULL],
                     [int echo = TRUE] )
```

Set the message display mode.

Description

The `pause_on_message()` primitive controls the automatic pausing upon a macro message; by default the *pause status* is off.

On the execution of the `message` primitive plus the internal equivalent, the *pause status* is checked and when enabled error messages shall be displayed with a .. suffix and Grief shall await for the user to acknowledge the condition via a key-press before execution continues.

If `pause` is specified as a non-negative (≥ 0) then the current setting is set to the pause value of the integer expression, whereas a negative value (-1) behaves like a inquiry primitive reporting the current value without

change. Otherwise if omitted the current status is toggled.

Unless `echo` is stated as **FALSE**, upon modification of the pause state, the new status shall be echoed on the command as follows

Pausing all messages on.

or

Pausing all messages off.

Parameters

`pause` Optional int flag, when a non-negative value ($>=0$) it states the new pause status, a negative value (-1) reports the current value without change, otherwise if omitted the current pause state is toggled.

`echo` Optional int flag, when stated as **FALSE** the function is silent, otherwise the change in the pause state is echoed on the command prompt.

Returns

The `pause_on_message` function returns the previous pause state.

Portability

`echo` is a **GriefEdit** extension.

Example

The following examples enables error acknowledgement during the execution of the macro `subshell()` and restores the previous status on completion.

```
int state = pause_on_message(TRUE);
subshell();
pause_on_message(status, FALSE);
```

See Also

[error](#), [message](#)

printf

```
void printf(string format,
           ...     )
```

Print formatted string to stdout.

Description

The `printf()` primitive can be used for debugging macros, although its use is questionable. It exists for compatibility reasons with BRIEF. It causes the formatted string to be sent directly to stdout, bypassing all of Griefs internal screen manipulations.

The format primitive is similar to the C `printf()` primitive, exporting the formatted result to stdout. Use of this interface is debatable as output is written to standard out, which is also the primary interface to the terminal/display.

The `printf()` primitive produces formatted output according to the format specification `format` into the diagnostics log. The trailing arguments ... are the integer, string or list expressions used to satisfy the % formatting options; refer to (See: [message](#)) for details on the supported format specifications.

This function is one of a set of formatted output primitives each supporting a common feature set, see [message](#) for a complete discussion on the supported format specification syntax.

Parameters

`format` String containing the <Format Specification>.

... Optional list of arguments which should correspond to the values required to satisfy the format specification.

Returns

`nothing`

Portability

The standard function has a void declaration and returns nothing.

See Also

[error](#), [message](#), [dprintf](#)

profile

```
void profile([int flags])
```

Profiling support.

Description

The *profile()* primitive controls the profiler flags.

If *flags* is omitted, then the current profiler state is toggled, otherwise sets the profiler state to the specified *flags*.

Returns

nothing

See Also

[debug](#)

watch

```
void watch(string symbol)
```

Watch a symbol.

Description

The *watch()* primitive is reserved for future use.

Returns

n/a

Portability

n/a

See Also

[debug](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Dialog Primitives

Summary

Dialog Primitives

Macros

<code>dialog_create</code>	Build a dialog resource.
<code>dialog_delete</code>	Delete a dialog resource.
<code>dialog_exit</code>	Exit a dialog resource.
<code>dialog_run</code>	Execute a dialog resource.
<code>inq_dialog</code>	Retrieve the current dialog resource.
<code>widget_get</code>	Retrieve a widget attribute.
<code>widget_set</code>	Set a widget attribute.

Macros

`dialog_create`

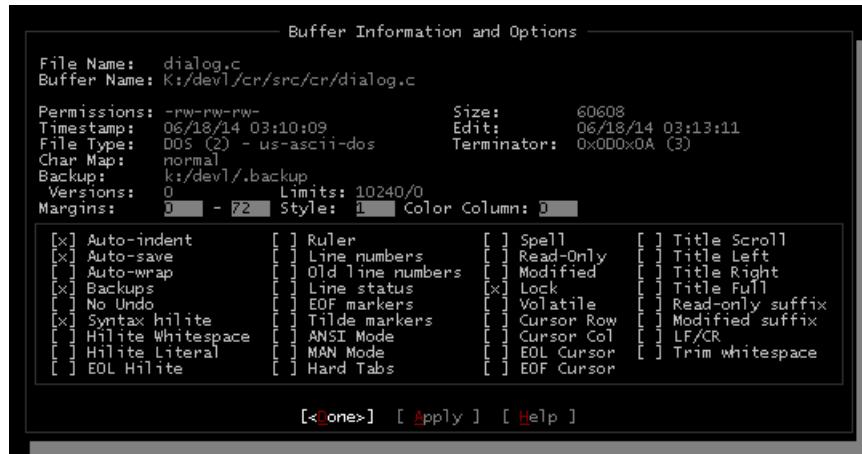
```
int dialog_create(list decl)
```

Build a dialog resource.

Description

The `dialog_create()` primitive creates a dialog resource from the specified definition `decl`.

The buffer information dialog is just one example.



The declaration list should contain the dialog elements plus their associated value, if any. Elements classifications are *Containers*, *Widgets* and *Attributes*.

Containers

Atom	Description
DLGC_GROUP	Variable sized visual and logical group container, with an optional frame and title.
DLGC_CONTAINER	General widget group, used to logical group a collection of widgets.
DLGC_TAB	Fixed size visual and logical container, with an optional title.

Widgets

Atom	Description
DLGC_CHECK_BOX	Check box
DLGC_COMBO_FIELD	Combo field
DLGC_EDIT_FIELD	Edit field

Atom	Description
DLGC_LABEL	Text label
DLGC_LIST_BOX	List box
DLGC_NUMERIC_FIELD	Numeric field
DLGC_PUSH_BUTTON	Push button
DLGC_RADIO_BUTTON	Radio button
DLGC_SEPARATOR_HORIZONTAL	Horizontal separator
DLGC_SEPARATOR_VERTICAL	Vertical separator
DLGC_SPACER	Spacer

Attributes

Atom	Type	Description
DLGA_TITLE	String	Attribute title.
DLGA_NAME	String	Attribute name.
DLGA_IDENT	Int	Identifier.
DLGA_CALLBACK	String	Macro callback.
DLGA_HELP	String	Help topic.
DLGA_TOOLTIP	String	Tooltip topic.
DLGA_X	Int	Horizontal position.
DLGA_Y	Int	Vertical position.
DLGA_COLS	Int	Width in columns.
DLGA_ROWS	Int	Height in rows.
DLGA_ATTACH_TOP	n/a	
DLGA_ATTACH_BOTTOM	n/a	
DLGA_ATTACH_LEFT	n/a	
DLGA_ATTACH_RIGHT	n/a	
DLGA_ALIGN_N	n/a	
DLGA_ALIGN_NE	n/a	
DLGA_ALIGN_E	n/a	
DLGA_ALIGN_SE	n/a	
DLGA_ALIGN_S	n/a	
DLGA_ALIGN_SW	n/a	
DLGA_ALIGN_W	n/a	
DLGA_ALIGN_NW	n/a	
DLGA_ALIGN_CENTER	n/a	
DLGA_ALLOW_EXPAND	n/a	
DLGA_ALLOW_FILLX	n/a	
DLGA_ALLOW_FILLY	n/a	
DLGA_PROPAGATE	Boolean	
DLGA_CANCEL_BUTTON	n/a	
DLGA_DEFAULT_BUTTON	n/a	
DLGA_DEFAULT_FOCUS	n/a	
DLGA_VALUE	Any	
DLGA_LABEL	String	
DLGA_TEXT_ONLY	n/a	
DLGA_GUI_ONLY	n/a	
DLGA_ACCELERATOR	String	
DLGA_HOTKEY	Integer	
DLGA_GREYED	n/a	
DLGA_ACTIVE	n/a	
DLGA_SENSITIVE	Integer	
DLGA_PADX	Integer	
DLGA_PADY	Integer	
DLGA_ORIENTATION	Integer	
DLGA_HIDDEN	n/a	
DLGA_VISIBLE	n/a	
DLGA_KEYDOWN	Boolean	
DLGA_TABSTOP	Boolean	
DLGA_AUTOMOVE	Boolean	
DLCA_USERDATA	Int	
DLGA_EDEDEDITABLE	Boolean	
DLGA_EDMAXLENGTH	Integer	

Atom	Type	Description
DLGA_EDVISIBILITY	Boolean	
DLGA_EDPOSITION	Integer	
DLGA_EDPLACEHOLDER	String	
DLGA_LBCOUNT	Integer	
DLGA_LBELEMENTS	Any	
DLGA_LBSORTMODE	Integer	
DLGA_LBHASSSTRINGS	Integer	
DLGA_LBICASESTRINGS	Integer	
DLGA_LBDUPLICATES	Integer	
DLGA_LBCLEAR	n/a	
DLGA_LBCURSOR	Integer	
DLGA_LBACTIVE	Integer	
DLGA_LBDISPLAYTEXT	Integer	
DLGA_LBDISPLAYTEXTLEN	Integer	
DLGA_LBTEXT	Integer	
DLGA_LBTEXTLEN	Integer	
DLGA_LBPAGEMODE	Boolean	
DLGA_LBINDEXMODE	Boolean	
DLGA_CBEDITABLE	Boolean	
DLGA_CBRELAXMODE	Integer	
DLGA_CBAUTOCOMPLETEMODE	Integer	
DLGA_CBPOPUPMODE	Integer	
DLGA_CBPOPUPSTATE	Integer	
DLGA_GAUGEMIN	Int Str	
DLGA_GAUGEMAX	Int Str	

Parameters

`decl Dialog definition.`

TODO

Allow the dialog definition to be XML.

Returns

On success returns a unique dialog resource identifier otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[dialog_run](#), [dialog_delete](#)

dialog_delete

```
int dialog_delete(int dialog)
```

Delete a dialog resource.

Description

The `dialog_delete()` primitive deletes the specified dialog resource.

Parameters

`dialog` Optional dialog instance handle, if omitted the current dialog is referenced.

Returns

The `dialog_delete()` primitive return non-zero on success, otherwise zero on error.

Portability

A **GriefEdit** extension.

See Also

[dialog_create](#)

dialog_exit

```
int dialog_exit(int retval = 0,
               [int dialog])
```

Exit a dialog resource.

Description

The *dialog_exit()* primitive exits the current dialog.

Parameters

retval Integer return value, returned to the original *dialog_run* caller.
dialog Optional dialog instance handle, if omitted the current dialog is referenced.

Returns

Returns 1 on success otherwise 0.

Portability

A **GriefEdit** extension.

See Also

[dialog_run](#)

```
int dialog_run(int dialog,
               [string args])
```

Execute a dialog resource.

Description

The *dialog_run()* primitive executes the dialog associated with the instance *dialog*, supplying the arguments *args*.

Parameters

dialog Dialog instance handle.
args Optional string containing

Returns

The *dialog_run()* primitive returns the exit value of the terminating *dialog_exit*.

Portability

A **GriefEdit** extension.

See Also

[dialog_create](#)

inq_dialog

```
int inq_dialog()
```

Retrieve the current dialog resource.

Description

The *inq_dialog()* primitive retrieves the resource handle of the current executing dialog source (if any), otherwise zero.

Parameters

none

Returns

The *inq_dialog()* primitive returns an integer representing the dialog resource handle (See: [dialog_create](#)) for more details.

Portability

A **GriefEdit** extension.

See Also

[dialog_create](#), [dialog_run](#)

widget_get

```
declare widget_get( [int dialog],
                    [int name|string name],
                    [int attr = DLGA_VALUE],
                    [int index = 0] )
```

Retrieve a widget attribute.

Description

The *widget_get()* primitive retrieves the value of a dialog resource.

Parameters

dialog Optional dialog instance handle, if omitted the current dialog is referenced.
name Resource identifier or name.
attr Attribute, if omitted DLGA_VALUE is assumed.
index Optional integer index, required for attributes with multiple elements.

Returns

The *widget_get()* primitive returns the assigned value, otherwise NULL on error.

Portability

A **GriefEdit** extension.

See Also

[widget_set](#)

```
declare widget_set(      [int dialog],
                        [int name|string name],
                        declare value,
                        [int attr = DLGA_VALUE],
                        [int index = 0] )
```

Set a widget attribute.

Description

The *widget_set()* primitive sets the value of a dialog resource.

Parameters

dialog Optional dialog instance handle, if omitted the current dialog is referenced.
name Resource identifier or name.
value Value to assign.
attr Attribute, if omitted DLGA_VALUE is assumed.
index Optional integer index, required for attributes with mutliple elements.

Returns

The *widget_set()* primitive returns the assigned value, otherwise NULL on error.

Portability

A **GriefEdit** extension.

See Also

[widget_get](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Environment Primitives

Summary

Environment Primitives

Macros

cd	Change directory.
create_char_map	Create a display character-map.
get_term_characters	Retrieve terminal special characters.
get_term_feature	Get value of a terminal feature.
get_term_features	Retrieve terminate features.
get_term_keyboard	Retrieve the terminal keyboard definitions.
getenv	Retrieve an environment variable.
geteuid	Retrieve the effective user identifier.
getpid	Retrieve the process identifier.
getuid	Retrieve the user identifier.
getwd	Get current working directory.
inq_backup	Get the backup creation mode.
inq_backup_option	Get backup options.
inq_brief_level	Retrieve the editor nesting level.
inq_char_timeout	Get the escape delay.
inq_environment	Retrieve an environment variable.
inq_feature	Retrieve an editor feature.
inq_hostname	Retrieve the local host name.
inq_idle_default	Retrieve idle interval.
inq_idle_time	Retrieve keyboard idle time.
inq_profile	Retrieve profile directory.
inq_username	Retrieve the user name.
putenv	Set an environment variable.
set_backup	Set backup creation mode.
set_backup_option	Set backup options.
set_char_timeout	Set the escape delay.
set_idle_default	Set idle interval.
set_term_characters	Set terminal special characters.
set_term_feature	Set a terminal attribute.
set_term_features	Define terminal attributes.
set_term_keyboard	Define terminal keyboard definitions.
uname	Retrieve system information.

Macros

cd

```
int cd([string dir])
```

Change directory.

Description

The `cd()` primitive changes the current directory on the specified drive to the specified path. Unlike `chdir` shell expansion shall occur when the path contains special characters, expanding home references `~`, wild cards and environment variables.

```
cd(newdir);
```

Under DOS/WIN32, if no drive is specified in path then the current drive is assumed. The path can be either relative to the current directory on the specified drive or it can be an absolute path name

If `dir` is omitted, the current directory shall be echoed on the command prompt without effecting any change.

```
cd();
```

Returns

The `cd()` primitive returns 1 if successful otherwise zero on error. When an error has occurred, the global `errno` contains a value indicating the type of error that has been detected.

See Also

`mkdir`, `rmdir`

create_char_map

```
int create_char_map( [int mapid|string name],
                     [int start = 0],
                     [list chars],
                     [list flags],
                     [string name] )
```

Create a display character-map.

Description

The `create_char_map()` primitive either creates or updates an existing character-map. Character-map's determine the mapping between the buffer encoding, special purpose characters and their display representation.

A map represents each of the 256 possible byte values which may be contained within any given 8-byte encoded buffer. In addition to the 256 base characters a number of special markers for End-Of-Line, Tabs and End-Of-File may also be defined.

The identifier `mapid` or the alias `name` either references an existing map or if omitted a new map shall be generated. On creation a character-map is derived from the system "normal" map.

Upon the existing or created map the specified `chars` list is then applied starting at the character `start`, updating each character in sequence until the end of the list is encountered. Optionally the pairs of integer values contained within the `flags` list are then applied in the same manor. If specified, the character-maps alias is changed to `name`.

Character-map Names

By default two system predefined character-map's are available known by the names "normal" and "binary", these are in addition to the number managed by the view's package; care should be taken not to overwrite these resources prior to understanding the impact.

Special Characters

The follow special character indexes have importance when displaying buffer content and can be controlled using `create_char_map` for example.

```
create_char_map("literal", CMAP_EOF, quote_list("[EOF]"));
```

Constant	Value	Description
CMAP_TABSTART	-	Beginning of a Tab
CMAP_TABVIRTUAL	-	Tab body
CMAP_TABEND	-	Tab end
CMAP_EOL	-	End-of-line marker
CMAP_EOF	-	End-of-file marker

By default, all specials are simple spaces as such invisible.

Note, for CRISP™ compatibility the `char` list may reference 257 characters with the 257th entry being equivalent to `CMAP_EOL`.

Character Flags

The `flags` argument should be given as a set of one or more integer pairs:

```
{ <character>, <flag> }
```

The character flags available include, with one

Constant	Value	Description
CMAP_DEFAULT	0	Default
CMAP_TAB	1	Horizontal tab
CMAP_BACKSPACE	2	Backspace

Constant	Value	Description
CMAP_ESCAPE	3	ESCAPE character

Normal Character Map

The "normal" or default character-map is built using the rules below.

- control-characters - ^[A-Z] in ASCII, otherwise use of UNICODE 0x2400-0x241F C0 character range.
- DEL - ^? otherwise UNICODE 0x2421.
- printable - all characters below 0x80 are themselves, otherwise characters above if 8bit and not 0xff or UNICODE.
- non-printable - hexadecimal representation of the form *0x##*.

Parameters

mapid, name Character-map reference, being either an integer map identifier or the associated map name as a string. If omitted a unique identifier shall be allocated.
start Optional integer, stating the first character index.
chars Character definition list, being a list of integer and/or string values for each character in the sequence starting at the offset start.
flags Character flag list, being one or more pairs of integer values. Within each pair, The first element is the integer character position, and the second is the integer flag value from the set "Character Flags".
name Optional string, being the unique name by which the character-map may be referenced.

Returns

The *create_char_map()* primitive returns the identifier of the resolved or created character-map otherwise -1 if the specified character map does not exist.

Example

The following enables an ASCII representation for control characters 0 to 31.

```
int map =
    create_char_map(0, NULL,
        quote_list(
            "<NUL>", "<SOH>", "<STX>", "<ETX>",
            "<EOT>", "<ENQ>", "<ACK>", "<BEL>",
            "<BS>", "<HT>", "<NL>", "<VT>",
            "<FF>", "<CR>", "<SO>", "<SI>",
            "<DLE>", "<DC1>", "<DC2>", "<DC3>",
            "<DC4>", "<NAK>", "<SYN>", "<ETB>",
            "<CAN>", "<EM>", "<SUB>", "<ESC>",
            "<FS>", "<GS>", "<RS>", "<US>"),
        quote_list('\t', CMAP_TAB));
set_buffer_cmap(map);
```

Portability

n/a

See Also

[inq_char_map](#)

get_term_characters

```
list get_term_characters( [top_left],
                           [top_right],
                           [bottom_left],
                           [bottom_right],
                           [vertical],
                           [horizontal],
                           [top_join],
                           [bottom_join],
                           [cross],
                           [left_join],
                           [right_join],
                           [scrol],
                           [thumb] )
```

Retrieve terminal special characters.

Description

The `set_term_characters()` primitive retrieves the set of characters which are utilised by the `tty` console driver to represent window borders.

`set_term_characters()` operators in one of two modes. Without arguments a list of strings, one for each character is retrieved. Alternatively each parameter is either an integer or string variable to be populated with the associated character value. Values can be omitted by supplying a NULL parameter against the associated character index.

Refer to [set_term_characters](#) for the order and meaning of each of these characters.

Parameters

... Integer character value or string escape sequence, one for each character within the set.

Returns

Without any arguments the `get_term_characters` primitive returns a list of strings, one for each terminal character, in the form `name=value`. Otherwise a NULL list is returned.

Portability

n/a

See Also

[set_term_characters](#), [set_term_feature](#)

get_term_feature

```
int get_term_feature(string | int ident,
                     declare value )
```

Get value of a terminal feature.

Description

The `get_term_feature()` primitive retrieves the value associated with the specified attribute `ident`.

Parameters

`ident` Either the integer identifier or the a string containing the name of the terminal attribute to be retrieved.
`value` Variable to populated with the referenced attribute.

Returns

The `set_term_feature()` primitive returns 0 on success, otherwise 1 on error.

Portability

A **GriefEdit** extension.

See Also

[set_term_feature](#)

get_term_features

```
list get_term_features(...)
```

Retrieve terminate features.

Description

The `get_term_features()` primitive retrieves a list of string pairs one for each terminal attribute.

Parameters

none

Returns

Without any arguments the `get_term_features()` primitive returns a list of strings, one for each terminal attribute, in the form `name=value`. Otherwise a NULL list is returned.

Portability

Many of the return values are only meaningful on systems that use a `tty` based console interface.

The CRISPEdit™ version is similar to [get_term_feature](#).

See Also

[get_term_keyboard](#), [get_term_feature](#)

get_term_keyboard

```
list get_term_keyboard()
```

Retrieve the terminal keyboard definitions.

Description

The *get_term_keyboard()* primitive retrieves the current terminal keyboard definition.

Under Unix™ console attributes are derived from the system *termcap* or *terminfo* databases or similar interfaces under non-unix systems.

See *get_term_keyboard* for further details.

Parameters

none

Returns

The *get_term_keyboard()* primitive returns a list of key binding pairs, each pair consisting of an integer key-code plus a string containing the associated escape sequence.

Portability

On systems not utilising a *tty* based console interface, the list shall be empty.

See Also

[set_term_features](#)

getenv

```
string getenv(string name)
```

Retrieve an environment variable.

Description

The *getenv()* primitive searches the environment of the calling process for the environment variable name if it exists and returns the value of that environment variable. If the specified environment variable cannot be found, an empty string shall be returned.

Parameters

name String containing the name of the environment variable.

Returns

The *getenv()* primitive returns the value of the corresponding environment variable, otherwise an empty string.

Portability

n/a

See Also

[putenv](#), [inq_environment](#)

geteuid

```
int geteuid()
```

Retrieve the effective user identifier.

Description

The *geteuid()* primitive shall return the effective user ID of the calling process.

Parameters

none

Returns

The *geteuid()* primitive shall always be successful returning the current effective user identifier.

Portability

n/a

See Also

[getpid](#), [getuid](#)

getpid

```
int getpid()
```

Retrieve the process identifier.

Description

The *getpid()* primitive shall return the process ID of the calling process.

Parameters

none

Returns

The *getpid()* primitive shall always be successful returning the current process identifier.

Portability

n/a

See Also

[getuid](#)

getuid

```
int getuid()
```

Retrieve the user identifier.

Description

The *getuid()* primitive shall return the real user ID of the calling process.

Parameters

none

Returns

The *getuid()* primitive shall always be successful returning the current user identifier.

Portability

n/a

See Also

[getpid](#), [geteuid](#)

getwd

```
int getwd(int ignored,
          string dir      )
```

Get current working directory.

Description

The *getwd()* primitive primitive retrieves the name of the current working directory.

The *ignored* parameter exists for compatibility with BRIEF.

Returns

The *getwd()* primitive returns 1 if successful otherwise zero on error. When an error has occurred, the global [errno](#) contains a value indicating the type of error that has been detected.

See Also

[cd](#), [mkdir](#), [rmdir](#)

inq_backup

```
int inq_backup([int bufnum])
```

Get the backup creation mode.

Description

The *inq_backup()* primitive retrieves the value of the associated objects backup mode. If *bufnum* is specified, then

the value of the stated buffer is returned, otherwise upon being omitted the global (default) backup mode shall be returned.

Returns

The *inq_backup()* primitive returns the current value of the backup flag.

Portability

A **GriefEdit** extension.

See Also

[set_backup](#), [set_backup_option](#), [inq_backup_option](#)

inq_backup_option

```
declare inq_backup_option(int what,
                         [int bufnum])
```

Get backup options.

Description

The *inq_backup_option()* primitive retrieves the value of the backup option *what* for the associated object. If *bufnum* is specified, then the value of the stated buffer is returned, otherwise upon being omitted the global (default) backup option shall be modified.

Returns

The *inq_backup_option()* primitive returns the current value associated with the attribute *what*, as follows

- **BK_MODE** - Backup creation mode, as an integer.
- **BK_AUTOSAVE** - Buffer autosave flag, as an integer.
- **BK_DIR** - Backup directory, as a string.
- **BK_DIRMODE** - Directory creation mode, as an integer.
- **BK VERSIONS** - Number of versions to be maintained, as an integer.
- **BK_PREFIX** - Backup filename prefix, as a string.
- **BK_SUFFIX** - Backup filename suffix, as a string.
- **BK_ONESUFFIX** - Whether only a single suffix to used replacing any existing, as an integer.
- **BK_ASK** - Filesize watermark at which point backups shall be prompted before created a backup image, as an integer.
- **BK_DONT** - Filesize watermark at which point backups shall not be created regardless of the current backup mode, as an integer.

Portability

A **GriefEdit** extension.

See Also

[set_backup](#), [inq_backup](#)

inq_brief_level

```
int inq_brief_level()
```

Retrieve the editor nesting level.

Description

The *inq_brief_level()* primitive retrieves the number of copies of GRIEF running within the current session.

The original implementation returned the total number of instances in memory, whereas this emulation only reports the number of instances visible with the associated terminal.

This function is provided for compatibility using the *getenv* interface; see the [getenv](#) primitive and the *brief* macro module for details.

Parameters

none

Returns

The current number of active editor sessions.

Portability

Provided for BRIEF compatibility, retrieving the current **GRLEVEL** environment variable level.

See Also[inq_environment](#)**inq_char_timeout**

```
int inq_char_timeout()
```

Get the escape delay.

Description

The *inq_char_timeout()* primitive retrieves the current value of the terminal escape delay.

Parameters

none

Returns

The *inq_char_timeout()* primitive returns the current character timeout.

Portability

A **GriefEdit** extension

See Also[inq_kbd_char](#), [read_char](#)**inq_environment**

```
string inq_environment(string name)
```

Retrieve an environment variable.

Description

The *inq_environment()* primitive shall search the environment of the calling process for the environment variable *name* if it exists and return the value of the environment variable. If the specified environment variable cannot be found, an empty string shall be returned.

This function is provided for compatibility using the *getenv* interface; see the [getenv](#) primitive and the *brief* macro module for details.

Parameters

name String containing the name of the environment variable.

Returns

The *inq_environment()* primitive returns the value of the corresponding environment variable, otherwise an empty string.

Portability

n/a

See Also[getenv](#)**inq_feature**

```
int|list inq_feature([int|string feature],
                      [string value]           )
```

Retrieve an editor feature.

Description

The *inq_feature()* primitive sets the status of the specific feature *feature*, returning its current configuration value.

Warning:

The *inq_feature()* primitive is an experimental interface and may change without notice.

Parameters

feature Name of the feature.

value Configuration value.

Return

The *inq_feature()* primitive returns non-zero on success, otherwise zero on error.

Macro Portability; A **GriefEdit** extension.

See Also

[set_feature](#), [inq_display_mode](#), <set>

inq_hostname

```
string inq_hostname()
```

Retrieve the local host name.

Description

The *inq_hostname()* primitive retrieves the name of the local host.

Parameters

none

Returns

The *inq_hostname()* primitive returns a string containing the local host name.

Portability

A **GriefEdit** extension.

See Also

[inq_username](#), [getenv](#)

inq_idle_default

```
int inq_idle_default()
```

Retrieve idle interval.

Description

The *inq_idle_default()* primitives retrieves the current keyboard idle interval, see [set_idle_default](#).

Parameters

none

Returns

The *inq_idle_default()* primitives returns the current value of the interval timer.

Portability

n/a

See Also

[set_idle_default](#)

inq_idle_time

```
int inq_idle_time()
```

Retrieve keyboard idle time.

Description

The *inq_idle_time()* primitives retrieves the number of seconds since the user last pressed a key, representing the time keyboard input has been idle.

Parameters

none

Returns

The *inq_idle_time()* primitive returns the idle timer in seconds.

Portability

n/a

See Also

[inq_idle_default](#), [set_idle_default](#)

inq_profile

```
string inq_profile()
```

Retrieve profile directory.

Description

The *inq_profile()* primitive retrieves the directory within which user specific profile information can be stored, examples include the restore state and personal dictionary.

This directory shall be system specific.

- On Unix style systems, the profile is normally a sub-directory within the home directory of the current user is specified by the \$HOME environment variable; this is also represented by the ~ directory.

```
~/.grief
```

- On Windows systems, the profile is normally a sub-directory within the directory that serves as a common repository for application-specific data. A typical path is

```
C:\Documents and Settings\<user>\Application Data\.grief.
```

Parameters

none

Returns

The *inq_profile()* primitive returns a string containing system dependent profile directory.

Portability

A **GriefEdit** extension.

See Also

[inq_environment](#)

inq_username

```
string inq_username()
```

Retrieve the user name.

Description

The *inq_username()* primitive retrieves the current user name.

Parameters

none

Returns

The *inq_username()* primitive returns a string containing the current user name.

Portability

A **GriefEdit** extension.

See Also

[inq_hostname](#), [getenv](#)

putenv

```
void putenv(string name,  
           [string value])
```

Set an environment variable.

Description

The *putenv()* primitive shall use the string arguments to set the value of an environment variable.

The arguments must construct a string of the form

```
"name=value"
```

This construction may be stated by one or two means. Firstly by the explicit string arguments *name* and *value*, or alternatively a single argument *name* already correctly formed containing both components separated by an embedded = within the string.

Parameters

- name* String containing the name of the environment variable and if *value* is omitted should also contain the associated value separated by an equals =.
- value* Optional value to be assigned.

Returns

Upon successful completion, *putenv* shall return one on success, otherwise zero and set *errno* to indicate the error.

Portability

A **GriefEdit** extension.

See Also

[getenv](#)

set_backup

```
int set_backup(int mode,
              [int bufnum])
```

Set backup creation mode.

Description

The *set_backup()* primitive either toggles, set or retrieves the current backup flag of the associated object. If *bufnum* is specified, then the stated buffer is effected, otherwise upon being omitted the global (default) backup mode shall be effected.

The backup flag is tested each time a buffer is written to its underlying storage (See: [write_buffer](#)). The flag maybe one of two states, either *off* or *on*; when *on*, then a backup files shall be created.

If *mode* is not specified, then the value of the associated backup flag is toggled from *on* to *off* or *off* to *no*. If given as zero backups shall be disabled, with a positive non-zero value enabling backups.

The *set_backup()* primitive may also be used to inquire the value of the associated backup flag. If the specified *mode* is -1 then the current flag value of the associated object shall be returned without any other effects.

When invoked from the command, one of the following messages shall be echoed

```
"Backups will be created".
or "Backups will not be created."
```

```
"Backups will be created for the buffer".
or "Backups will not be created for the buffer."
```

Returns

The *set_backup()* primitive returns the previous value of the backup flag.

Portability

n/a

See Also

[inq_backup](#), [set_backup_option](#), [inq_backup_option](#)

set_backup_option

```
int set_backup_option(int what,
                     [int bufnum],
                     parameter )
```

Set backup options.

Description

The `set_backup_option()` primitive sets the value of the backup option *what* for the associated object. *backup mode*. If *bufnum* is specified, then the value of the stated buffer is modified, otherwise upon being omitted the global (default) backup mode shall be modified.

parameter shall be *what* specific, with the following options available;

- **BK_MODE** - Backup creation mode, see [set_backup](#) and [inq_backup](#). A zero integer *parameter* shall disable backups, with a non-zero value enabling backups.
- **BK_AUTOSAVE** - Buffer autosave flag. A zero integer *parameter* shall disable auto-backups, with a non-zero value enabling auto-backups.
- **BK_DIR** - Backup directory. *parameter* should a string containing the backup directory path.
- **BK_DIRMODE** - Directory creation mode. *parameter* should be a integer value specifying the file creation umask.
- **BK VERSIONS** - Number of versions to be maintained. *parameter* should be an integer value specifying the number of backup versions to be kept in the range [1 .. 99]; values outside shall set the versions to the lower/upper bounds.
- **BK_PREFIX** - Backup filename prefix. *parameter* should be a string containing the prefix, an empty string shall clear the current suffix.
- **BK_SUFFIX** - Backup filename suffix/extension. *parameter* should be a string containing the suffix, an empty string shall clear the current prefix.
- **BK_ONESUFFIX** - Whether only a single suffix to used replacing any existing, otherwise append the suffix shall be appended. *parameter* should be integer, with a non-zero enabling or zero disabling one-suffix filtering on backup filenames.
- **BK_ASK** - Filesize watermark at which point backups shall be prompted before created a backup image. *parameter* should be a positive integer value being the watermark filesize in bytes, with a value of zero disabling any prompts.
- **BK_DONT** - Filesize watermark at which point backups shall not be created regardless of the current backup mode. *parameter* should be a positive integer value being the watermark filesize in bytes, with a value of zero disabling any affected.

Returns

The `set_backup_option()` on success returns zero, otherwise -1 on error.

Portability

A [GriefEdit](#) extension.

See Also

[inq_backup_option](#), [set_backup](#), [inq_backup](#)

set_char_timeout

```
void set_char_timeout([int timeout])
```

Set the escape delay.

Description

The `set_char_timeout()` primitive sets the length of time to wait before timing out and treating the ESC keystroke as the ESC character rather than combining it with other characters in the buffer to create a key sequence.

The most common instance where you may wish to change this value is to work with slow hosts, e.g., running on a network. If the host cannot read characters rapidly enough, it will have the same effect as if the terminal did not send characters rapidly enough. The library will still see a timeout.

The escape-delay value defaults to 750 milliseconds, but this shall be overridden by the `ESCDELAY` environment variable or the `--escdelay` command line option.

The `ESCDELAY` environment is defined specifies the default timeout interval, which is generally measured in millisecond. If `ESCDELAY` is 0, the system immediately composes the ESCAPE response without waiting for more information from the buffer. The user may choose any value between 0 and 9999, inclusive.

Note:

The `ESCDELAY` value is generally measured in milliseconds under most unix environments, but under AIX this value is measured in fifths of a milliseconds (200 microseconds) to be compatible with `libcursor` implementations; the default value is 500, or 0.1 second

Parameters

`timeout` Optional character timeout is milliseconds, otherwise the current **ESCDELAY** shall be reapplied.

Returns

The `set_char_timeout()` primitive returns the previous character timeout.

Portability

n/a

See Also

[inq_char_timeout](#)

set_idle_default

```
int set_idle_default(int internal = 0)
```

Set idle interval.

Description

The `set_idle_default()` primitives set the keyboard idle interval as a measure of seconds between the last keystroke and when the **REG_IDLE** event is generated, see [register_macro](#) for details.

Parameters

`interval` Integer idle interval in seconds, if omitted the system default shall be utilised.

Returns

The `set_idle_default()` primitives returns the previous value of the interval timer.

Portability

n/a

See Also

[inq_idle_default](#)

set_term_characters

```
int set_term_characters(int value0,
                      .... )
```

Set terminal special characters.

Description

The `set_term_characters()` primitive is one of a set of functions which value add to the console interface specification. Under Unix™ console attributes are derived from the system *termcap* or *terminfo* databases or similar interfaces under non-console and non-unix systems.

The `set_term_characters()` primitive configures the set of characters which are utilised by the *tty* console driver to represent window borders.

Each of the character parameters listed below may be either an integer or string expression representing the character value. An integer value (including character constants) are treated as a single character whilst running within graphics-mode, whereas a string shall be interpreted as an escape sequence. Values can be omitted by supplying a NULL parameter against the associated character index.

This function should generally be invoked prior to the display being enabled (See: [display_windows](#)), otherwise a [redraw](#) is required.

Index	Name	Description
0	top_left	Top left of a window.
1	top_right	Right right of a window.
2	bottom_left	Bottom left of a window.
3	bottom_right	Bottom right of a window.
4	vertical	Vertical window sides.
5	horizontal	Horizontal window sides.
5	top_join	Top intersection of window corners.
6	bottom_join	Bottom intersection of window corners.
7	cross	Intersection of window corners.
8	left_join	Left Window intersection.
9	right_join	Right window intersection.

Index	Name	Description
10	scroll	Scroll bar arena.
11	thumb	Scroll bar thumb.

Parameters

... Integer character value or string escape sequence, one for each character within the set.

Example

The following configures a terminal to use simple non-graphical characters to represent window borders.

```
set_term_characters(
    '+', // 0. Top left of window.
    '+', // 1. Top right of window.
    '+', // 2. Bottom left of window.
    '+', // 3. Bottom right of window.
    '|', // 4. Vertical bar for window sides.
    '- ', // 5. Top and bottom horizontal bar for window.
    '+', // 6. Top join.
    '+', // 7. Bottom join.
    '+', // 8. Window 4-way intersection.
    '+', // 9. Left hand join.
    '+', // 10. Right hand join.
    '**', // 11. Thumb wheel.
);
```

Note:

For further examples refer to the *tty* macros, which setup the terminal interface for many well known environments.

Returns

nothing

Portability

n/a

See Also

[set_term_features](#), [set_term_keyboard](#)

set_term_feature

```
int set_term_feature( int|string ident,
                      [string|int value])
```

Set a terminal attribute.

Description

The *set_term_feature()* primitive is one of a set of functions which value add to the console interface specification. Under Unix™ console attributes are derived from the system *termcap* or *terminfo* databases or similar interfaces under non-console and non-unix systems.

The *set_term_feature()* primitive allows certain attributes about the current display to be modified; for example features which are not adequately handled by *termcap* or *terminfo*. The attribute identified by *ident* shall be set to *value*.

Parameters

ident Either the integer identifier or a string containing the name of the terminal attribute to be modified.

value Value to be applied against the attribute, if omitted or NULL the current value is cleared. The type of the value is specific to the attribute being modified, either in string or integer form.

Types

Each of the attributes are defined using one of four data types.

Escape	Terminal specific escape sequence being a series of characters that shall be written to the attached terminal.
	Controls supporting parameters should use printf style %d as the place holders for parameters; allowing terminfo/termcap independent macro development.
	Most of the sequence attributes are no longer required since the current generation of curses/ncurses terminfo or terminfo databases are complete. Supplied values shall only be utilised in the event these are missing from the current terminal description.
Boolean	Boolean flag, which can be stated as either a numeric zero/non-zero value or a string value yes/no.
Integer	Numeric value, either stated as an integer value or string containing a decimal value.
String	String value.

Attributes

The following terminal attributes are exposed

Constant	Name	Type	Description
TF_PRINT_SPACE	repeat_space	Escape	Control sequence to print a space <i>n</i> times.
TF_PRINT_BITEIGHT	print_character	Escape	Control sequence to print eight bit characters.
TF_INSERT_CURSOR	insert_cursor	Escape	Control sequence which modifies the cursor looks, utilised when GriefEdit has insert-mode enabled.
TF_OVERWRITE_CURSOR	overwrite_cursor	Escape	Control sequence which modifies the cursor looks, utilised when GriefEdit has overwrite-mode enabled.
TF_VINSERT_CURSOR	virt_insert_cursor	Escape	Control sequence which modifies the cursor looks, utilised when GriefEdit has insert-mode enabled and is positioned over a virtual-space.
TF_VOVERWRITE_CURSOR	virt_overwrite_cursor	Escape	Control sequence which modifies the cursor looks, utilised when GriefEdit has overwrite-mode enabled and is positioned over a virtual-space.
TF_PRINT_ESCAPE	print_escape	Escape	Control sequence which prints an ESC character graphically.
TF_REPEAT_LAST	repeat_last	Escape	Control sequence for repeating the previous character printed <i>n</i> times.
TF_0M_RESETS_COLOR	0m_resets_color	Boolean	When non-zero, then whenever an ESC[0m is printed, it is assumed that the background and foreground colours are reset (defunct).
TF_COLOR	color	Boolean	When non-zero, the terminal is assumed to support color.
TF_CURSOR_RIGHT	parm_cursor_right	Escape	Control sequence for move the cursor <i>n</i> characters within on the same line.
TF_CLEAR_IS_BLACK	clear_is_black	Boolean	When true states that area erase commands set the background of the erased line to black, otherwise the current background is assumed.
TF_DISABLE_INSDEL	noinsdel	Boolean	When true disables the use of window scrolling when attempting to optimise output; on several displays its faster to rewrite the window.
TF_GRAPHIC_MODE	graphics_mode	Escape	Control sequence to send when sending graphics (line drawing) characters.
TF_TEXT_MODE	text_mode	Escape	Control sequence to send when exiting graphics mode and going back to normal character set.
TF_RESET	reset	Escape	Control sequence to be executed during terminal initialisation operations.
TF_INIT	init	Escape	Control sequence to be executed during terminal reset operations.
TF_COLORSETFGBG	colorset_fgbg	Escape	Control sequence when executed sets both the foreground and backgrounds colours.
TF_COLORSET_FG	colorset_fg	Escape	Control sequence when executed sets the foreground colour.

Constant	Name	Type	Description
TF_COLORSET_BG	colorset_bg	Escape	Control sequence when executed sets the background colour.
TF_COLORDEPTH	color_depth	Integer	Colour depth.
TF_DEFAULT_FG	default_fg_color	Integer	Terminals default foreground color.
TF_DEFAULT_BG	default_bg_color	Integer	Terminals default background color.
TF_SCHEMEDARK	scheme_dark	Boolean	When true the dark colour scheme variant shall be select over the light; if available.
TF_COLORMAP	color_map	String	Xterm colour map specification; a comma separated list of colours defining the colour palette.
TF_COLORPALETTE	color_palette	String	User defined palette.
TF_EIGHT_BIT	eight_bit	Boolean	When true the console supports the display of eight bit characters.
TF_MOUSE	mouse	String	Mouse device.
TF_MOUSECLKMS	mouse_clickms	Integer	Mouse click time in milliseconds.
TF_WINDOW_SETSIZE	winsetsize	Escape	Control sequence to set the window size to x by y.
TF_WINDOW_SETPOS	winsetpost	Escape	Control sequence to set the window position at x by y.
TF_CODEPAGE	codepage	Integer	Character code page.
TF_DISABLE_SCROLL	scroll_disable	Boolean	When true disables use of scroll region commands.
TF_SCROLL_MAX	scroll_max	Integer	Upper size of window to be scrolled permitting use of scroll region commands.
TF_NOALTSCREEN	noaltscreen	Boolean	When true denotes no alternative screen selection is available.
TF_LAZYUPDATE	lazy_update	Integer	When set to a positive value, indicates lazy screen updates are enabled. Screen updates are delayed until the keyboard becomes idle upto the stated number of lines.
TF_ATTRIBUTES	attributes	Integer	Terminal attribute bitmap; see the table below.
TF_NAME	name	String	Name of the terminal.
TF_TTY_FAST	tty_fast	Integer	When true disables use of pad characters.
TF_TTY_GRAPHICSBOX	tty_graphicsbox	Integer	When true enables use of graphic box characters.
TF_SCREEN_ROWS	screen_rows	Integer	Number of screen rows.
TF_SCREEN_COLS	screen_cols	Integer	Number of screen columns.
TF_LINENO_COLUMNS	lineno_columns	Integer	Terminal specific number of columns allocated for the display of buffer number lines.
TF_WINDOW_MINROWS	window_minrows	Integer	Minimum vertical size of a window.
TF_WINDOW_MINCOLS	window_mincols	Integer	Minimum horizontal size of a window.
TF_XTERM_COMPAT	xterm_compat	Integer	When true terminal to treated like an xterm.
TF_XTERM_CURSOR	xterm_cursor	Integer	When true terminal supports xterm cursor control commands.
TF_XTERM_TITLE	xterm_title	Integer	Control sequence when executed set the console title.
TF_XTERM_PALETTE	xterm_palette	Integer	Xterm palette selector.
TF_VT_DATYPE	vt_datype	Integer	Terminal type; xterm device attribute, known values are. 0 xterm. 77 mintty. 83 screen. 82 rxvt. 85 rxvt unicode.
TF_VT_DAVERSION	vt_daversion	Integer	Terminal version.
TF_ENCODING	encoding	String	Terminal character encoding.
TF_UNICODE_VERSION	unicode_version	String	UNICODE interface version, for example "6.0.1".

TF_ATTRIBUTE

The reported **TF_ATTRIBUTE** attribute represents special terminal features mined during terminal initialisation. The flag argument can contain none or more of the following symbols bitwise OR'ed together.

Constant	Description
TF_AGRAPHICCHARACTERS	Graphic characters (ACS defined)
TF_AFUNCTIONKEYS	F1-F10 function keys.
TF_ACYGIN	Cygwin native console.
TF_AUTF8ENCODING	UTF8 character encoding, Unicode implied.
TF_AUNICODEENCODING	Unicode character encoding.
TF_AMETAKEY	Meta keys.

Note:

Many of the exposed attributes are specific to the underlying terminal, a well known is the *Xterm Control Sequences* available on <http://invisible-island.net>.

Returns

The `set_term_feature()` primitive returns 0 on success, otherwise 1 on error.

Portability

A **GriefEdit** extension.

See Also

`set_term_features`, `set_term_characters`, `set_term_keyboard`

set_term_features

```
int set_term_features(list features)
```

Define terminal attributes.

Description

The `set_term_features()` primitive is one of a set of functions which value add to the console interface specification. Under Unix™ console attributes are derived from the system *termcap* or *terminfo* databases or similar interfaces under non-console and non-unix systems.

Similar to the `set_term_feature`, `set_term_features` allows attributes about the current display to be modified; for example features which are not adequately handled by *termcap* or *terminfo*.

`set_term_features` utilises a list interface with elements within the list implied by the TF_XXXX enumeration. This permits bulk attribute initialisation at the expense of visibility. See `set_term_feature` for further details on the exposed attributes.

Parameters

`list` Terminal attributes one or each attribute in the order implied by the TF_XXXX enumeration.
 Each parameter may be either an integer or string expression representing the character value. An integer value (including character constants) are treated as a single character whilst within graphics-mode, whereas a string shall be interpreted as an escape sequence. Values can be omitted by supplying a NULL parameter against the associated character index.

Returns

nothing

Example

```

set_term_features(
    NULL,           // TF_PRINT_SPACE
    NULL,           // TF_PRINT_BITEIGHT
    NULL,           // TF_INSERT_CURSOR
    NULL,           // TF_OVERWRITE_CURSOR
    NULL,           // TF_VINSERT_CURSOR
    NULL,           // TF_VOVERWRITE_CURSOR
    NULL,           // TF_PRINT_ESCAPE
    NULL,           // TF_REPEAT_LAST
    FALSE,          // TF_0M_RESETS_COLOR
    FALSE,          // TF_COLOR
    "\x1B[%dc",    // TF_CURSOR_RIGHT
    TRUE,           // TF_CLEAR_IS_BLACK
    FALSE,          // TF_DISABLE_INSDEL
    "\x1B(0",       // TF_GRAPHIC_MODE
    "\x1B(B"        // TF_TEXT_MODE
);

```

Portability

Many of the return values are only meaningful on systems that use a *tty* based console interface.

See Also

[set_term_feature](#), [set_term_characters](#), [set_term_keyboard](#)

set_term_keyboard

```
int set_term_keyboard(list kbd)
```

Define terminal keyboard definitions.

Description

The *set_term_keyboard()* primitive is one of a set of functions which value add to the console interface specification. Under Unix™ console attributes are derived from the system *termcap* or *terminfo* databases or similar interfaces under non-unix systems.

The *set_term_keyboard* function manages the keyboard scan-code (terminal escape sequence) to key-code mapping table. For each key definition *set_term_keyboard* expects a pair of arguments, the first being Griefs internal key-code (See: [key_to_int](#)) with the second stating the associated escape sequence.

The mapped escape sequence shall be treated as unique; in what only the last instance shall be retained. Whereas key-codes can be associated with multiple escape sequences. For example, a generic Xterm mapping could support alternative function key reports.

The second argument may takes one of two forms. The first being a single string containing the escape sequence, the second being a key-list permitting a range of keys to be bound.

Example simple key associates including non-unique key codes.

```

set_term_keyboard(
    :
    KEY_PAGEUP,      "\x1b[5~",   // vt220 mode
    KEY_PAGEDOWN,    "\x1b[6~",
    KEY_INS,          "\x1b[2~",
    KEY_DEL,          "\x7f",
    KEY_PAGEUP,      "\x1b[5z",   // sunFunctionKeys mode
    KEY_PAGEDOWN,    "\x1b[6z",
    KEY_INS,          "\x1b[2z",
    :
);

```

Character Ranges

The key-lists are simply a list of strings describing each consecutive key within a set or range of keys. A constant set of keys can be simply quoted using the [quote_list](#) function, alternatively the list can be constructed with the other list primitives.

The following character definitions are generally defined using a consecutive set of keys.

- F1_F12
- SHIFT_F1_F12

- CTRL_A_Z
- CTRL_F1_F12
- CTRLSHIFT_F1_F12
- ALT_F1_F12
- ALT_A_Z
- ALT_0_9
- KEYPAD_0_9
- CTRL_KEYPAD_0_9
- SHIFT_KEYPAD_0_9
- ALT_KEYPAD_0_9

Example of a consecutive definition against a constant set of sequences.

```
set_term_keyboard(
    F1_F12, quote_list(
        "\x1bOP",      "\x1bOQ",      "\x1bOR",      "\x1bOS",
        "\x1b[15~",    "\x1b[17~",    "\x1b[18~",    "\x1b[19~",
        "\x1b[20~",    "\x1b[21~",    "\x1b[23~",    "\x1b[24~"
    );
)
```

Ambiguous

Due to the complex nature of terminal escape sequences, many key definitions result in ambiguous mappings; for example two or more mappings starting with the same sequence of characters. The mapping of ambiguous sequences to their associated key-code utilise a scan delay mechanism. When an ambiguous sequence is detected the driver waits several milliseconds before selecting the longest matching sequence, see [set_char_timeout](#) for details.

Note:

Many of the reported escapes are specific to the underlying terminal, a well known is the *Xterm Control Sequences* available on <http://invisible-island.net>.

Parameters

none

Returns

nothing

Example

The following is an example from the Linux terminal definition.

```
set_term_keyboard(
    :
    F1_F12,          quote_list(
        "\x1bOP",      "\x1bOQ",      "\x1bOR",      "\x1bOS",
        "\x1b[15~",    "\x1b[17~",    "\x1b[18~",    "\x1b[19~",
        "\x1b[20~",    "\x1b[21~",    "\x1b[23~",    "\x1b[24~"
    ),
    SHIFT_F1_F12,    quote_list(
        NULL,         NULL,         "\x1b[25~",    "\x1b[26~",
        "\x1b[28~",    "\x1b[29~",    "\x1b[31~",    "\x1b[32~",
        "\x1b[33~",    "\x1b[34~",    "\x1b[23;2~",  "\x1b[24;2~"
    ),
    CTRL_F1_F12,    quote_list(
        "\x1bO5P",    "\x1bO5Q",    "\x1bO5R",    "\x1bO5S",
        "\x1b[15;5~", "\x1b[17;5~", "\x1b[18;5~", "\x1b[19;5~",
        "\x1b[20;5~", "\x1b[21;5~", "\x1b[23;5~", "\x1b[24;5~"
    ),
    :
);
```

Note:

For further examples refer to the *tty* macros, which setup the terminal interface for many well known environments.

Portability

n/a

See Also[set_term_characters](#), [set_term_features](#)**uname**

```
int uname( [string &sysname],  
          [string &nodename],  
          [string &version],  
          [string &release],  
          [string &machine] )
```

Retrieve system information.

Description

The *uname()* primitive retrieves the strings corresponding to the result of a *uname* system call. On non-POSIX systems, these strings maybe blank unless there is an equivalent value available.

Parameters

sysname	System name.
nodename	Network node name.
version	Kernel version
release	Kernel release.
machine	Machine type.

Returns

The *uname()* primitive returns 0 otherwise, -1 on error.

Portability

A **GriefEdit** extension.

See Also[version](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

File Primitives

Summary

File Primitives

Macros

access	Test file accessibility.
basename	Return the last component of a pathname.
chdir	Change directory.
chmod	Change mode.
chown	Change owner.
compare_files	Binary file compare.
copy_ea_info	Copy file extended information.
del	Delete a file.
dirname	Report the parent directory name of a file pathname.
edit_file	Edit a file.
edit_file2	Extended file edit.
exist	Check file existence.
expandpath	Expand the path.
fclose	Close a stream.
feof	Test end-of-file indicator on a stream.
ferror	Test error indicator on a stream.
fflush	Flush a stream.
file_canon	Canonicalize a path.
file_match	File match utility.
file_pattern	Directory file pattern.
filename_match	Perform file pattern matching.
filename_realpath	Return a resolved pathname.
find_file	Read next directory entry.
find_file2	Extended read next directory entry.
find_macro	Determine path of a macro object.
ioctl	File miscellaneous control.
flock	File lock operations.
mktemp	Make a unique filename.
fopen	Open a stream.
fread	Read from a stream.
fseek	Reposition a file-position indicator in a stream.
fstat	Stream status information.
ftell	Report the file-position indicator of a stream.
ftest	Test file type.
ftruncate	Truncate the specified file.
fwrite	Write to a stream.
glob	Generate pathnames matching a pattern.
inq_buffer	Retrieve a buffer identifier.
inq_file_magic	Retrieve the file type detection rules.
inq_home	Retrieve the user home directory.
inq_tmpdir	Get the temporary-file directory.
inq_vfs_mounts	Retrieve list of mounts.
link	Link a file.
lstat	Get symbolic link status.
mkdir	Create a directory.
mktemp	Make a temporary filename.
output_file	Change the output file-name.
read_ea	Read file extended information.
read_file	Read into the current buffer.
readlink	Read the contents of a symbolic link.
realpath	Resolve a pathname.
remove	Remove a file.
rename	Rename a file.
rmdir	Remove directory.
searchpath	Searches for a given file in a specified path.
set_binary_size	Set binary chunk size.
set_ea	Set file extended information.
set_file_magic	Define the file type detection rules.
stat	Obtain file information.
symlink	Create a symbolic link.
umask	Set and get the file mode creation mask.
unlink	Unlink a file.
vfs_mount	Mount a virtual file-system.
vfs_unmount	Unmount a virtual file-system.

File Modes

Traditional Unix file mode consist of a number of components including type, permissions including special bits.

Macros

access

```
int access(string path,
          int      mode )
```

Test file accessibility.

Description

The *access()* primitive is used to check the accessibility of a file. The *file* parameter is the name of a directory or file which is to be tested and *mode* are a set of access bits which are used to check the effective access.

When the value of *mode* is zero, only the existence of the file is verified. The meaning of mode is operating system dependent yet most operating systems support the following definitions:

- F_OK* Check if file exists (0).
- X_OK* Check to see if file is executable (1).
- W_OK* Check if file is writable (2).
- R_OK* Check if file is readable (3).

Returns

On successful return this primitive returns ≥ 0 ; -1 is returned on an error, and the global variable **errno** is set to the reason for the failure.

Portability

A **GriefEdit** extension.

See Also

[exist](#), [stat](#), [lstat](#), [fstat](#), [chmod](#)

basename

```
int basename(string pathname,
            [string suffix])
```

Return the last component of a pathname.

Description

The *basename()* primitive returns a string containing the final component of a *pathname* contained in the string path argument, deleting trailing path separators.

To accomplish this, basename first checks to see if name consists of nothing, but slash (/) or backslash (\) characters. If so, basename replaces name with a single slash and the process is complete. If not, basename removes any trailing slashes. If slashes still remain, basename strips off all leading characters up to and including the final slash.

If you specify *suffix* and the remaining portion of name contains a suffix which matches *suffix*, basename removes that suffix.

Returns

The *basename()* primitive returns a string containing the final component of the specified *pathname* after processing following the above rules.

See Also

[dirname](#), [strfilecmp](#)

chdir

```
int chdir(string path)
```

Change directory.

Description

The `chdir()` primitive changes the current directory on the specified drive to the specified path.

Unlike `cd` no shell expansion shall occur.

Returns

The `chdir()` primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global `errno` contains a value indicating the type of error that has been detected.

Portability

A **GriefEdit** extension.

See Also

`cd`, `mkdir`, `rmdir`, `getwd`

chmod

```
int chmod(string path,
          int     mode )
```

Change mode.

Description

The `chmod()` primitive changes the permissions for a file specified by `path` to be the settings in the `mode` given by permission.

The access permissions for the file or directory are specified as a combination of bits which are operating system specific.

Returns

The `chmod()` primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global `errno` contains a value indicating the type of error that has been detected.

See Also

`chown`, `stat`, `Istat`, `umask`

chown

```
int chown(string path,
          int     owner,
          int     group )
```

Change owner.

Description

The `chown()` primitive is reserved for future use.

Returns

The `chown()` primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global `errno` contains a value indicating the type of error that has been detected.

See Also

`chmod`, `stat`, `Istat`, `umask`

compare_files

```
int compare_files(      [int flags],
                      string file1,
                      string file2      )
```

Binary file compare.

Description

The `compare_files` performs simple comparison byte-for-byte of the on disk images of the two file `file1` and `file2`.

The `flags` are intended for future options, supporting simple line processing.

It can be used to quickly determine if two files are identical (byte for byte). Note that if either file is an active buffer, no unsaved changes shall be included.

In contrast, the `<diff_buffers>` primitive is used to compare files and mark up the differences.

Returns

Returns 1 if the files are identical, 0 if they differ. -1 if the first file cannot be read, -2 if the second file cannot be read.

Portability

A **GriefEdit** extension.

See Also

`<diff_buffers>`

copy_ea_info

```
int copy_ea_info(string sourcename,
                 string destname    )
```

Copy file extended information.

Description

The `copy_ea()` primitive is reserved for future BRIEF compatibility.

Parameters

n/a

Returns

n/a

Portability

Provided for BRIEF compatibility.

See Also

`read_ea`, `set_ea`

del

```
int del(string name)
```

Delete a file.

Description

The `del()` primitive deletes the specified file *name*.

This function is provided for compatibility using the `remove` interface; see the `remove` primitive and the `brief` macro module for details.

Parameters

`name` String containing the file to be removed.

Returns

`remove`, `delete_buffer`

dirname

```
int dirname(string path)
```

Report the parent directory name of a file pathname.

Description

The `dirname()` primitive shall take a string *path* that contains a pathname, and return a string containing that is a pathname of the parent directory of that file. Trailing directory separators (/ and \) characters in the path are not counted as part of the path.

If *path* does not contain a directory separator (/ and \) or is an empty string, then `dirname` shall return a string ".".

Returns

The `dirname()` primitive returns string containing the parent directory name of a file pathname, otherwise a string containing " ".

Portability

A **GriefEdit** extension.

See Also

basename

edit_file

```
int edit_file(...)
```

Edit a file.

Description

The `edit_file()` primitive creates a new buffer, loads the contents, attaches the buffer to the current window and executes any registered macros, see [register_macro](#).

The argument specification is a set of one or more strings with optional leading integer modes. Modes control the flags under which the subsequent files are processed.

If the specified file is already within another buffer, that buffer becomes the current buffer and is attached to the current window.

If more than one argument is specified; either directly or as the result of file expansion; then a separate edit action is performed on each file, with the current buffer being the last stated file.

File Expansion

The `edit_file()` primitive performs file name *globbing* in a fashion similar to the *csh* shell. It returns a list of the files whose names match any of the pattern arguments.

The pattern arguments may contain any of the following special characters:

<code>~[user/]</code>	Home directory of either the current or the specified user.
<code>?</code>	Matches any single character.
<code>*</code>	Matches any sequence of zero or more characters.
<code>[ch]</code>	Matches any single character in chars. If ch's contains a sequence of the form <i>a-b</i> then any character between <i>a</i> and <i>b</i> (inclusive) will match.
<code>\x</code>	Matches the character <i>x</i> .

File detection

During file loading GRIEF performs a number of tests against the sections of the file content in an attempt to determine the file encoding. These operations are generally invisible to end user and behave on most file types without interaction.

The current default scanners include, see [set_file_magic](#) for additional details on each.

<code>mark</code>	Encoding: < marker >
<code>utf32bom</code>	UTF-32 BOM marker.
<code>utf16bom</code>	UTF-16 BOM marker.
<code>udet</code>	Mozilla Universal Character Detector.
<code>magic</code>	File magic.
<code>binary</code>	Possible binary image.
<code>ascii</code>	ASCII only (7-bit).
<code>latin1</code>	Latin-1 (ISO-8859-x) data.
<code>big5</code>	Chinese Big5.
<code>gb18030</code>	Chinese GB-18030.
<code>shiftjis</code>	Shift-JIS.
<code>xascii</code>	Extended ASCII.

Parameters

... Argument specification is a set of one or more strings with optional leading integer modes.
Modes control the flags under which the subsequent files are processed, see examples. Unless specified files are loaded using **EDIT_NORMAL**.
If no arguments are specified, the user shall be prompted for a file specification.

Modes

Constant	Description
<code>EDIT_NORMAL</code>	Default mode; auto-guess the file type.
<code>EDIT_BINARY</code>	Force file to be read in binary mode, see set_binary_size .
<code>EDIT_ASCII</code>	Force file to be read in ASCII mode.
<code>EDIT_STRIP_CR</code>	Force CR removal on input and write them on output.
<code>EDIT_STRIP_CTRLZ</code>	Force Ctrl-Z removal.
<code>EDIT_SYSTEM</code>	System buffer.

Constant	Description
EDIT_RC	Enable extended return codes.
EDIT_QUICK	Quick detection, removes more costly character-encoding methods.
EDIT AGAIN	<edit_again> implementation.
EDIT_LOCAL	Edit locally, do not rely on the disk being stable.
EDIT_READONLY	Set as read-only.
EDIT_LOCK	Lock on read (strict-locking).
EDIT_NOLOCK	Disable auto-locking.
EDIT_PREVIEW	Limit the load to the window size.

Returns

The `edit_file()` primitive returns a positive value on success, 0 if the user was prompted and cancelled, otherwise -1 on error.

If more than one argument is specified; either directly or as the result of file expansion; the return relates to the last loaded file.

Under **EDIT_RC** the return code are extended to differentiation between success conditions as follows.

Value	Description
-1	Error.
0	Cancelled.
1	Successfully loaded buffer.
2	New image, file created.
3	Pre-existing buffer, not reloaded.

Example

Expand and load all .cpp files located within the current working directory.

```
edit_file(".cpp");
```

Loads a system buffer with the content from `config.ini` sourced from the users home directory.

```
edit_file(EDIT_SYSTEM, "~/config.ini");
```

Portability

The feature set exposed differs from implementations. It is therefore advised that the symbolic constants are using within a #ifdef construct.

See Also

`edit_file2`, `read_file`, `attach_buffer`, `call_registered_macro`, `create_buffer`, `set_buffer`, `set_file_magic`

edit_file2

```
int edit_file2( string      encoding,
                string|list file      )
```

Extended file edit.

Description

The `edit_file2()` primitive extends the functionality provided by `edit_file` with the leading parameter of `encoding`.

Parameters

`encoding` String containing the character encoding of the source file.
... See `edit_file`

Returns

The `edit_file2()` primitive returns a positive value on success, 0 if the user was prompted and cancelled, otherwise -1 on error.

See `edit_file` for additional return information.

Portability

A **GriefEdit** extension.

See Also

[edit_file](#), [read_file](#), [attach_buffer](#), [call_registered_macro](#), [create_buffer](#), [set_buffer](#), [set_file_magic](#)

exist

```
int exist(string path,
          [int canon = TRUE])
```

Check file existence.

Description

The `exist()` primitive tests whether of the specified file *path* exists.

If *canon* is not specified, then the filename is canonised first. This involves expanding any tilde prefixes and converting DOS style filenames to Unix style ones.

Returns

The `exist()` primitive returns non-zero if the file exists, otherwise 0 on error. When an error has occurred, the global `errno` contains a value indicating the type of error that has been detected.

See Also

[access](#), [stat](#), [fstat](#)

expandpath

```
string expandpath(string path,
                  [int env])
```

Expand the path.

Description

The `expandpath()` primitive expands any shell style home directory `~[user]` references/short-hands contained within the specified *path*, returning the result.

The supported shell constructs are.

`~/` is expanded to your current home directory.
`~user/` is expanded to the specified *users* home directory (unix only).

If *env* is specified and is non-zero, then the any embedded environment variable references are also expanded following the following syntax. When a macro is not defined it expands to `\\" (an empty string), and {} are synonyms for ()`.

`$ (name)` Expands to value of *name*.
`$(name)` Expands to value of *name*.
`$name/` Expands to value of *name*.

Returns

The `expandpath()` primitive returns a string containing the expanded path following the above rules, otherwise an empty string.

Portability

A **GriefEdit** extension.

See Also

[strfilecmp](#), [file_canon](#), [file_glob](#), [getenv](#), [searchpath](#)

fclose

```
int fclose(int handle)
```

Close a stream.

Description

The `fclose()` primitive shall cause the stream pointed to by *stream* to be flushed and the associated file to be closed. Any unwritten buffered data for the stream shall be written to the file; any unread buffered data shall be discarded.

Returns

On successful completion, *fclose()* shall return 0; otherwise, it shall return -1 and set *errno* to indicate the error.

Portability

A **GriefEdit** extension.

See Also

fopen, *fread*, *fwrite*

feof

```
int feof(int handle)
```

Test end-of-file indicator on a stream.

Description

The *feof()* primitive shall test the end-of-file indicator for the stream pointed to by *stream*.

Returns

The *feof()* primitive returns non-zero if and only if the end-of-file indicator is set for stream.

Portability

A **GriefEdit** extension.

Notes

The *feof()* primitive is reserved for future use, and currently returns -1 in all cases.

See Also

fopen, *ferror*

ferror

```
int ferror(int handle,
           [int clearerr])
```

Test error indicator on a stream.

Description

The *ferror()* primitive shall test the error indicator for the stream referenced by *handle*.

Returns

The *ferror()* primitive shall return non-zero if and only if the error indicator is set for stream.

Portability

A **GriefEdit** extension.

Notes

The *ferror()* primitive is reserved for future use, and currently returns -1 in all cases.

See Also

fopen, *feof*

fflush

```
int fflush(int handle,
          [int sync])
```

Flush a stream.

Description

The *fflush()* primitive is reserved for future use.

Returns

The *fflush* function returns non-zero if a write error occurs and zero otherwise. When an error has occurred, *errno* contains a value indicating the type of error that has been detected

Portability

An experimental **GriefEdit** extension; functionality may change.

Notes

The *fflush()* primitive is reserved for future use, and currently returns -1 in all cases.

See Also

[fopen](#)

file_canon

```
string file_canon(string filepath)
```

Canonicalize a path.

Description

The *file_canon()* primitive performs canonicalizes the specified path *filepath* and returns a new path.

The new path differs from path in

- Slashes are normalised with \ replaced with '/'.
- Relative paths are prefixed with the current working directory.
- Multiple `/'s are collapsed to a single `/`.
- Leading `./`s and trailing `./`s are removed.
- Trailing `/'s are removed.
- Non-leading `../`s and trailing `../`s are handled by removing portions of the path.

Returns

The *file_canon()* primitive returns the canonicalized file path.

Portability

A [GriefEdit](#) extension.

See Also

[glob](#), [file_pattern](#), [find_file](#), [expandpath](#)

file_match

```
int file_match(pattern,
               file,
               [flags] )
```

File match utility.

Description

The *file_match()* primitive performs wild-card name matching, which has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry.

The *find_file()* primitive is an example of this. It can also be used by the ts macro to process its pattern operands, or by applications that need to match strings in a similar manner.

The *file_match()* primitive is intended to imply filename match, rather than pathname match. The default action of this primitive is to match filenames, rather than path names, since it gives no special significance to the slash character.

pattern can be a list of search expressions or a string containing a single expression. *file* is the file-name that shall be tested.

If supplied, *flags* allows control over how directory delimiters slash(/) and dots (.) are matched within file-name. Otherwise it defaults to setting the MATCH_PERIODA and MATCH_NOCASE based on the current operating environment similar to ones used during *edit_file()* pattern matching.

Pattern

The following patterns matching a single character match a single character: ordinary characters, special pattern characters and pattern bracket expressions. The pattern bracket expression will also match a single collating element.

An ordinary character is a pattern that matches itself. It can be any character in the supported character set except for NUL, those special shell characters that require quoting, and the following three special pattern characters.

When unquoted and outside a bracket expression, the following three characters will have special meaning in the specification of patterns:

- ? A question-mark is a pattern that will match any character.

- * An asterisk is a pattern that will match multiple characters, as described in Patterns Matching Multiple Characters, see below.
- [] The open bracket will introduce a pattern bracket expression, see below.

Patterns Matching Multiple Characters

The following rules are used to construct patterns matching multiple characters from patterns matching a single character:

- The asterisk (*) is a pattern that will match any string, including the null string.
- The concatenation of patterns matching a single character is a valid pattern that will match the concatenation of the single characters or collating elements matched by each of the concatenated patterns.
- The concatenation of one or more patterns matching a single character with one or more asterisks is a valid pattern. In such patterns, each asterisk will match a string of zero or more characters, matching the greatest possible number of characters that still allows the remainder of the pattern to match the string.

Character Set Range Match

[introduces a pattern bracket expression, that will matches a single collating element contained in the non-empty set of collating elements. The following rules apply:

- A bracket expression is either a matching list expression or a non-matching list expression. It consists of one or more expressions.
- A matching list expression specifies a list that matches any one of the expressions represented in the list. The first character in the list must not be the circumflex (^). For example, [abc] is a pattern that matches any of the characters *a*, *b* or *c*.
- A non-matching list expression begins with a circumflex (^), and specifies a list that matches any character or collating element except for the expressions represented in the list after the leading circumflex. The circumflex will have this special meaning only when it occurs first in the list, immediately following the left-bracket.
- A range expression represents the set of collating elements that fall between two elements in the current collation sequence, inclusively. It is expressed as the starting point and the ending point separated by a hyphen (-). For example, [a-z] is a pattern that matches any of the characters *a* to *z* inclusive.
- A bracket expression followed by + means one or more times.

Character Classes

Within bracket expressions, the name of a character class enclosed in [: and :] stands for the list of all characters belonging to that class.

Standard (POSIX style) character classes are;

alnum	An alphanumeric (letter or digit).
alpha	A letter.
blank	A space or tab character.
cntrl	A control character.
csym	An alphanumeric (letter or digit) or or underscore character.
digit	A decimal digit.
graph	A character with a visible representation.
lower	A lower-case letter.
print	An alphanumeric (same as alnum).
punct	A punctuation character.
space	A character producing white space in displayed text, space or a tab.
upper	An upper-case letter.
xdigit	A hexadecimal digit.

Flags

If supplied the flags argument shall modify the interpretation of pattern and string. It is the bitwise-inclusive OR of zero or more of the flags defined in *grief.h*.

MATCH_PATHNAME - If the MATCH_PATHNAME flag is set in flags, then a slash character ('/') in string shall be explicitly matched by a slash in pattern; it shall not be matched by either the asterisk (*) or question-mark (?) special characters, nor by a bracket expression ([]). If the MATCH_PATHNAME flag is not set, the slash character shall be treated as an ordinary character.

MATCH_NOCASE - If the MATCH_NOCASE flag is set in flags, then the pattern is treated non-case sensitively, i.e. A matches a. Otherwise the search is performed with case being sensitive.

MATCH_NOESCAPE - If MATCH_NOESCAPE is not set in flags, a backslash character (\\) in pattern followed by any other character shall match that second character in string. In particular, "\\" shall match a backslash in string. If MATCH_NOESCAPE is set, a backslash character shall be treated as an ordinary character.

MATCH_PERIOD - If MATCH_PERIOD is set, a leading period in string will match a period in pattern; where the location of "leading" is indicated by the value of MATCH_PATHNAME as follows:

If not set, no special restrictions are placed on matching a period.

- is set, a period is “leading” if it is the first character in string or if it immediately follows a slash.
 - is not set, a period is “leading” only if it is the first character of string.
- MATCH_PERIODA, MATCH_PERIODQ and MATCH_PERIODB - If set these allow selective control whether the asterisk (*) or question mark (?) special characters, nor by a bracket ([]) expression have affect.

Examples

```
a [bc]
```

matches the strings ab and ac.

```
a*d
```

matches the strings ad, abd and abcd, but not the string abc.

```
a*d*
```

matches the strings ad, abcd, abcdef, aaaad and adddd.

```
*a*d
```

matches the strings ad, abcd, efabcd, aaaad and adddd.

Note

`file_match()` does not perform tilde expansion (See: [expandpath](#)).

Returns

The `file_match()` primitive returns 1 if file matches the specified pattern; returns 0 if pattern isn't matched. If pattern is a list, then the index into the list that matched the expression or -1.

Portability

A [GriefEdit](#) extension.

See Also

[file_pattern](#), [filename_match](#)

file_pattern

```
int file_pattern(string filespec)
```

Directory file pattern.

Description

The `file_pattern()` primitive sets the search pattern for files, using [find_file](#), and reset the internal status to the first matching file.

Parameters

`filespec` String containing the file pattern specification which should evaluate to a file name or a wild-card filename, see the [file_match](#) for details regarding the supported wildcard.

Returns

The `file_pattern()` primitive returns 0 on success, otherwise -1 on error. When an error has occurred, the global `errno` contains a value indicating the type of error that has been detected.

Notes

This interface enforces the rule (MATCH_PERIODA) that a leading ““ wont match any files which begin with a “.“ as such use “?” to match all files contained within a directory. See [file_match](#) for details on the `filespec` syntax.

See Also

[file_match](#), [glob](#), [expandpath](#)

filename_match

```
int filename_match(string file,
                  declare pattern)
```

Perform file pattern matching.

Description

The *filename_match()* primitive is similar to the *file_match()* primitive but is provided for CRISP™ compatibility. It is used to compare a filename to see if it matches a filename regular expression.

A filename expression is a regular expression similar to that accepted by the command line shells on Unix systems (e.g. you can use * for wildcard, ? for wild-character, and [...] to select a range of characters).

file is the filename that shall be tested. *pattern* can be a list of search expressions or a string containing a single expression (See: [file_match](#)) for details on the expression syntax.

Returns

The *filename_match()* primitive return value is dependent on the pattern type. If pattern is a string, then 1 if the filename matches; 0 otherwise. If pattern is a list, then the index into the list that matched the expression or -1.

Portability

Functionality has not been formally verified against CRISP Edit and whether it supports the [...] expression construct.

See Also

[file_match](#), [strfilecmp](#), [file_glob](#)

filename_realpath

```
string filename_realpath(string pathname)
```

Return a resolved pathname.

Description

The *filename_realpath()* primitive shall derive, from the *pathname* an absolute pathname that names the same file, whose resolution does not involve ., .., or symbolic links.

Returns

The *file_realpath()* primitive returns the resolved file if successful otherwise the original *pathname*.

See Also

[realpath](#)

find_file

```
int find_file( [string &filename],
               [int &size],
               [int &mtime],
               [int &ctime],
               [int &mode] )
```

Read next directory entry.

Description

The *find_file()* primitive retrieves the next file which matches the current file pattern. The active source directory and pattern are controlled using [file_pattern](#).

Example

The following example retrieves all .txt files within the current working directory.

```
string name;
int size;

file_pattern("*.txt");
while (file_find(name)) {
    parsename(name, size);
}
```

This primitive is used in conjunction with [file_pattern](#).

Parameters

- For regular files, the file size in bytes.

- For symbolic links, the length in bytes of the path-name contained in the symbolic link.
- For a shared memory object, the length in bytes.
- For a typed memory object, the length in bytes.
- For other file types, the use of this field is unspecified.

`filename` Optional string, is populated with the base filename; without any path.
`size` Optional integer, if specified is populated with the size of the related file in bytes.
`mtime` Optional integer, populated with the files modification time (See: [time](#)).
`ctime` Optional integer, populated with the time of the last status change.
`mode` Mode of file (See: [stat](#)).

Returns

Returns zero if there are no more files; returns 1 if next directory entry successfully received.

Portability

The `mtime`, `ctime` and `mode` parameters are **GriefEdit** extensions.

See Also

[file_pattern](#), [find_file2](#), [file_glob](#), [expandpath](#), [stat](#), [mode_string](#)

find_file2

```
int find_file2( string filename,
                [int &size],
                [int &mtime],
                [int &ctime],
                [int &atime],
                [int &mode],
                [int &uid],
                [string &uid2name],
                [int &gid],
                [string &gid2name],
                [int &nlink],
                [int &inode] )
```

Extended read next directory entry.

Description

The `find_file2()` primitive is an extended version of [find_file](#) returning additional file information on which matching file.

Parameters

- For regular files, the file size in bytes.
- For symbolic links, the length in bytes of the path-name contained in the symbolic link.
- For a shared memory object, the length in bytes.
- For a typed memory object, the length in bytes.
- For other file types, the use of this field is unspecified.

`filename` Optional string, is populated with the base filename; without any path.
`size` Optional integer, if specified is populated with the size of the related file in bytes.
`mtime` Optional integer, populated with the files modification time (See: [time](#)).
`ctime` Optional integer, populated with the time of the last status change.
`atime` Optional integer, populated with the last access time.
`mode` Optional integer, mode of file (See: [stat](#)).
`uid` Optional integer, user identifier of the file.
`uid2name` User name associated with the file uid.
`gid` Optional integer, group identifier of the file.
`gid2name` Group name associated with the file gid.
`nlink` Optional integer, number of hard links to the file.
`inode` Optional integer, populated with the file-system internal node number.

Returns

Returns zero if there are no more files; returns 1 if next directory entry successfully received.

Portability

A **GriefEdit** extension.

See Also

[file_pattern](#), [find_file](#), [file_glob](#), [expandpath](#), [stat](#), [mode_string](#)

find_macro

```
string find_macro(string filename)
```

Determine path of a macro object.

Description

The *find_module* resolves the full-path to the macro object *filename*. Using the same mechanism as [load_macro](#), the macro object search path environment variable **GRPATH** is utilised to search for the specified macro object.

Parameters

filename String containing the macro object to resolve. If the specified file has an extension, then an exact match is located. Otherwise files matching one of the following extensions are *.cm*, *.cr*, and *.m* are located.

Returns

The *find_module* returns a string containing the full-path to the resolved macro object, otherwise an empty string if the object could not be found.

Portability

n/a

See Also

[autoload](#), [load_macro](#)

fioctl

```
int fioctl(int handle,
           ..      )
```

File miscellaneous control.

Description

The *fioctl()* primitive is reserved for future use.

Returns

Returns -1.

Portability

An experimental **GriefEdit** extension; functionality may change.

See Also

[fopen](#), [flock](#)

flock

```
int flock(int handle,
          ..      )
```

File lock operations.

Description

The *flock()* primitive is reserved for future use.

Returns

Returns -1.

Portability

A **GriefEdit** extension.

Notes

The *flock()* primitive is reserved for future use, and currently returns -1 in all cases.

See Also

[fopen](#), [fioctl](#)

fmktemp

```
int fmktemp(string template);
```

Make a unique filename.

Description

The *fmktemp()* primitive shall replace the contents of the string *template* by a unique filename, and return a stream for the file open for reading and writing.

The primitive is equivalent to the *mkstemp()*.

The function thus prevents any possible race condition between testing whether the file exists and opening it for use. The string in *template* should look like a filename with six trailing X's; *fmktemp* replaces each X with a character from the portable filename character set.

The characters are chosen such that the resulting name does not duplicate the name of an existing file at the time of a call to *fmktemp*.

Returns

Upon successful completion, *fmktemp()* shall return an open stream and return the resulting *filename*. Otherwise, -1 shall be returned if no suitable file could be created.

Portability

A **GriefEdit** extension.

See Also

[fopen](#), [mktemp](#)

fopen

```
int fopen( string path,
           int|string flags,
           [int mode = 0644],
           [int bufsiz] )
```

Open a stream.

Description

The *fopen()* primitive shall open the file whose pathname is the string *path*, and associates a stream with it.

The *flags* argument contains a string. If the string is one of the following, the file shall be opened in the indicated mode.

- r Open file for reading.
- w Truncate to zero length or create file for writing.
- a Append; open or create file for writing at end-of-file.
- r+ Open file for update (reading and writing).
- w+ Truncate to zero length or create file for update.
- a+ Append; open or create file for update, writing at end-of-file.
- wx create text file for writing with exclusive access.
- wbx create binary file for writing with exclusive access.
- w+x create text file for update with exclusive access.
- w+bx create binary file for update with exclusive access.

Parameters

- path String containing either the full or relative path name of a file to be opened.
- flags Creation flags, see above.
- mode Optional creation mode.
- bufsiz Optionally stream buffer size, in bytes.

Returns

Upon successful completion, *fopen()* shall return a handle to the stream. Otherwise, -1 shall be returned, and *errno* shall be set to indicate the error.

Portability

A **GriefEdit** extension.

See Also

[fclose](#), [fread](#), [fwrite](#)

fread

```
int fread( int handle,
           string buffer,
           [int bufsiz = BUFSIZ],
           [int null = ' ']
         )
```

Read from a stream.

Description

The *fread()* primitive shall read into the array pointed to by *ptr* up to *nitems* elements whose size is specified by *size* in bytes, from the stream referenced by *handle*.

Parameters

- handle* Stream handle.
- buffer* String buffer to be written.
- bufsiz* Optional length of buffer to be read.

Returns

Upon successful completion, *fread()* shall return the number of elements successfully read which is less than *bufsiz* only if a read error or end-of-file is encountered.

If *nitems* is 0, *fread()* shall return 0 and the contents of the array and the state of the stream remain unchanged. Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

Portability

A **GriefEdit** extension.

See Also

[fopen](#), [fwrite](#)

fseek

```
int fseek(int handle,
          int offset,
          int whence )
```

Reposition a file-position indicator in a stream.

Description

The *fseek()* primitive shall set the file-position indicator for the stream pointed to by *stream*. If a read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.

The argument *offset* is the position to seek to relative to one of three positions specified by the argument *whence*.

- SEEK_SET* The new file position is computed relative to the start of the file. The value of *offset* must not be negative.
- SEEK_CUR* The new file position is computed relative to the current file position. The value of *offset* may be positive, negative or zero.
- SEEK_END* The new file position is computed relative to the end of the file.

Parameters

- handle* Stream handle.
- offset* The new position, measured in bytes from the beginning of the file, shall be obtained by adding *offset* to the position specified by *whence*.
- whence* The specified point is the beginning of the file for **SEEK_SET**, the current value of the file-position indicator for **SEEK_CUR**, or end-of-file for **SEEK_END**.

Returns

On success the *fseek()* functions returns, otherwise -1 and set *errno* to indicate the error.

Portability

A **GriefEdit** extension.

See Also

[ftell](#), [fwrite](#), [fread](#)

fstat

```
int fstat( int handle,
           [int size],
           [int mtime],
           [int ctime],
           [int atime],
           [int mode],
           [int uid],
           [string uid2name],
           [int gid],
           [string gid2name],
           [int nlink] )
```

Stream status information.

Description

The *fstat()* primitive obtain information about the file referenced by the handle *handle*.

This information is returned in the parameters following the *handle* parameter (if supported by the underlying filesystem)

<i>size</i>	Total file size, in bytes.
<i>mode</i>	File's mode (See: chmod).
<i>mtime</i>	The files "last modified" time (See: time).
<i>atime</i>	Time the file was "last accessed".
<i>ctime</i>	Time of the files "last status change".
<i>uid</i>	user-id.
<i>gid</i>	group-id.
<i>nlink</i>	Number of hard links.

Returns

The *fstat* function returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, *errno* contains a value indicating the type of error that has been detected.

Portability

A [GriefEdit](#) extension.

See Also

[fopen](#), [stat](#), [lstat](#)

f.tell

```
int ftell(int handle)
```

Report the file-position indicator of a stream.

Description

The *ftell()* primitive shall obtain the current value of the file-position indicator for the stream pointed to by *stream*.

Returns

On successful completion, *ftell()* shall return the current value of the file-position indicator for the stream measured in bytes from the beginning of the file.

Otherwise, *ftell()* shall return -1, and set *errno* to indicate the error.

Portability

A [GriefEdit](#) extension.

See Also

[fseek](#), [fwrite](#), [fread](#)

f.test

```
int ftest(int|string condition,
          string      path      )
```

Test file type.

Description

The *ftest()* primitive tests the type of the specified file *path* against the type *condition*. The file test operations are modelled on standard unix test operators as follows.

- a **True** if file exists.
- b **True** if file exists and is a block special file.
- B **True** if file exists and is a possible binary stream.
- c **True** if file exists and is a character special file.
- d **True** if file exists and is a directory.
- e **True** if file exists.
- f **True** if file exists and is a regular file.
- g **True** if file exists and its set group ID flag is set.
- G **True** if file exists and its group matches the effective group ID of this process.
- h **True** if file exists and is a symbolic link.
- k **True** if file exists and has its sticky bit set.
- L **True** if file exists and is a symbolic link.
- N **True** if file exists and has been modified since last access.
- O **True** if file exists and is owned by the effective user ID of this process.
- p **True** if file is a named pipe (FIFO).
- r **True** if file exists and is readable.
- s **True** if file exists and has a size greater than zero.
- S **True** if file exists and is a socket.
- u **True** if file exists and its set-user-ID flag is set.
- w **True** if file exists and is writable. **True** will indicate only that the write flag is on. The file will not be writable on a read-only file system even if this test indicates true.
- x **True** if file exists and is executable. **True** will indicate only that the execute flag is on. If file is a directory, true indicates that file can be searched.

Returns

The *ftest()* primitive returns the result of the test operation either 1 when **true** otherwise 0.

Portability

A **GriefEdit** extension.

See Also

[access](#), [exist](#), [stat](#), [chmod](#), [chown](#)

ftruncate

```
int ftruncate(int handle,
              [int size])
```

Truncate the specified file.

Description

The *ftruncate()* primitive cause the regular file referenced by *handle* to be truncated to a size of precisely length bytes.

If the file previously was larger than this size, the extra data is lost. If the file previously was shorter, it is extended, and the extended part reads as null bytes (\0).

The file offset is not changed.

With *ftruncate()*, the file must be open for writing.

Returns

The *ftruncate* function returns zero on success. When an error has occurred, *errno* contains a value indicating the type of error that has been detected

Portability

A **GriefEdit** extension.

Notes

The *ftruncate()* primitive is reserved for future use, and currently returns -1 in all cases.

See Also

[fseek](#), [ftell](#), [fwrite](#)

fwrite

```
int fwrite(int handle,
          string buffer,
          [int length])
```

Write to a stream.

Description

The *fwrite()* primitive shall write, from the string *buffer*, up to *length*, to the stream pointed to by *handle*. If omitted, *length* by default to the string current length.

The file-position indicator for the stream (if defined) shall be advanced by the number of bytes successfully written. If an error occurs, the resulting value of the file-position indicator for the stream is unspecified.

Parameters

handle Stream handle.
buffer String buffer to be written.
length Optional length of buffer to be written.

Returns

The *fwrite()* primitive shall return the number of elements successfully written, which may be less than *length* if a write error is encountered.

If *length* is 0, *fwrite()* shall return 0 and the state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

Portability

A **GriefEdit** extension.

See Also

[fopen](#), [fclose](#)

glob

```
string glob(string pattern)
```

Generate pathnames matching a pattern.

Description

The *glob()* primitive expands the specified *pattern* into single string similar to that which occurs on the shell command line.

- * Match any string of characters.
- [] Character class.
- ? Match any single character.
- ~ User name home directory.
- \x Quote the next metacharacter x.

Returns

The *glob()* primitive returns a string containing the result of the pattern expansion on success, otherwise an empty string.

Portability

A **GriefEdit** extension.

See Also

[file_glob](#), [file_pattern](#), [find_file](#), [expandpath](#)

inq_buffer

```
int inq_buffer([string filename])
```

Retrieve a buffer identifier.

Description

The *inq_buffer()* primitive retrieves either the identifier associated with the buffer containing the specified file *filename*, otherwise if omitted the identifier of the current buffer.

Parameters

bufname Optional string containing the file name to be matched against existing buffers.

Returns

The *inq_buffer()* primitive returns the unique identifier associated with the referenced file or the current buffer if no *file_name* was specified. If the specified *file_name* was not matched, zero is returned.

If omitted then the current buffer is returned.

Portability

Unlike BRIEF partial matches do not occur.

See Also

[attach_buffer](#), [create_buffer](#), [delete_buffer](#), [next_buffer](#), [set_buffer](#)

inq_file_magic

```
string inq_file_magic([int &isdefault])
```

Retrieve the file type detection rules.

Description

The *inq_file_magic* primitive retrieves the current file character encoding detection rules. The returned shall contain one or comma separated detector names with optional arguments, see [set_file_magic](#).

Example

```
mark,utf8,udet,xascii,ascii
```

Parameters

n/a

Returns

The *inq_file_magic()* primitive returns a string containing the current encoder specification.

Portability

A **GriefEdit** extension.

See Also

[edit_file](#), [set_file_magic](#)

inq_home

```
string inq_home()
```

Retrieve the user home directory.

Description

The *inq_home()* primitive retrieves the current users home directory.

Parameters

none

Returns

The *inq_home()* primitive returns a string containing the users home directory.

Portability

A **GriefEdit** extension.

See Also

[inq_username](#), [getenv](#)

inq_tmpdir

```
string inq_tmpdir()
```

Get the temporary-file directory.

Description

The *inq_tmpdir()* primitive retrieves the temporary file directory. The directory is determined by the current

environment, which is host specific, for example on UNIX systems the default value of this property is typically "/tmp" or "/var/tmp"; on Microsoft Windows systems it is typically "c:\temp".

Windows

On XP and greater the system folder *local application data* directory is queried.

If this fails, the function checks for the existence of environment variables in the following order and uses the first path found:

- The path specified by the **TMP** environment variable.
- The path specified by the **TEMP** environment variable.
- The path specified by the **USERPROFILE** environment variable.
- The Windows directory.

UNIX like Systems

TMPDIR is the canonical Unix environment variable that should be used to specify a temporary directory for scratch space.

Other forms accepted are **TEMP**, **TEMPDIR** and **TMP**, but these alternatives are used more commonly by non-POSIX operating systems or non-conformant programs.

Parameters

none

Returns

The *inq_tmpdir()* primitive returns a string containing the temporary file directory.

Portability

A **GriefEdit** extension.

See Also

[getenv](#)

inq_vfs_mounts

```
list inq_vfs_mounts()
```

Retrieve list of mounts.

Description

The *inq_vfs_mounts()* primitive retrieves a list of three elements describing each of the current mounted virtual file-systems.

Parameters

none

Returns

Returns a list of mount points, each mount description contains the following elements.

<i>mountpoint</i>	String containing the mount point, being the logical path representing the root of the mounted resource.
<i>prefix</i>	Prefix string, unique name detailing the underlying file-syste type, examples include <i>ftp</i> and <i>gzip</i> .
<i>flags</i>	Integer flags.

Portability

A **GriefEdit** extension.

See Also

[vfs_mount](#), [vfs_unmount](#)

link

```
int link(string path1,
        string path2 )
```

Link a file.

Description

The *link()* primitive is reserved for future use.

The *link()* primitive shall create a new link (directory entry) for the existing file, *path1*.

The *path1* argument points to a pathname naming an existing file. The *path2* argument points to a pathname naming the new directory entry to be created. The *link()* primitive shall atomically create a new link for the existing file and the link count of the file shall be incremented by one.

If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the implementation supports using *link()* on directories.

If *path1* names a symbolic link, it is implementation-defined whether *link()* follows the symbolic link, or creates a new link to the symbolic link itself.

If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.

Returns

Upon successful completion, *link* shall return 0; otherwise, it shall return -1 and set the global **errno** to indicate the error.

Portability

A **GriefEdit** extension.

See Also

[symlink](#), [unlink](#), [stat](#), [remove](#)

Istat

```
int lstat( string path,
           [int size],
           [int mtime],
           [int ctime],
           [int atime],
           [int mode],
           [int uid],
           [string uid2name],
           [int gid],
           [string gid2name],
           [int nlink],
           [int inode] )
```

Get symbolic link status.

Description

The *lstat()* primitive shall be equivalent to [stat](#), except when path refers to a symbolic link. In that case *lstat* shall return information about the link, while *stat* shall return information about the file the link references.

For symbolic links, the *mode* member shall contain meaningful information when used with the file type macros, and the *size* member shall contain the length of the pathname contained in the symbolic link. File mode bits and the contents of the remaining members of the stat structure are unspecified. The value returned in the *size* member is the length of the contents of the symbolic link, and does not count any trailing null.

Parameters

- For regular files, the file size in bytes.
- For symbolic links, the length in bytes of the path-name contained in the symbolic link.
- For a shared memory object, the length in bytes.
- For a typed memory object, the length in bytes.
- For other file types, the use of this field is unspecified.

path	String containing the file path.
size	Optional integer, if specified is populated with the size of the related file in bytes.
mtime	Optional integer, populated with the files modification time (See: time).
ctime	Optional integer, populated with the time of the last status change.
atime	Optional integer, populated with the last access time.
mode	Optional integer, mode of file ((see File Modes)).
uid	Optional integer, user identifier of the file.
uid2name	User name associated with the file uid.
gid	Optional integer, group identifier of the file.
gid2name	Group name associated with the file gid.
nlink	Optional integer, number of hard links to the file.
inode	Optional integer, populated with the file-system internal node number.

Returns

The *lstat()* primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global **errno** contains a value indicating the type of error that has been detected.

Portability

A **GriefEdit** extension.

See Also

[stat](#)

mkdir

```
int mkdir(string pathname,
         int mode      = 0755)
```

Create a directory.

Description

The *mkdir()* primitive creates a new subdirectory with name *path*. The path can be either relative to the current working directory or it can be an absolute path name.

mode is the protection codes used to create the directory. If omitted, the value **0755** (*-rwxr-xr-x*) will be used.

Returns

The *mkdir()* primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global **errno** contains a value indicating the type of error that has been detected.

See Also

[rmdir](#), [cd](#), [getwd](#)

mktemp

```
string mktemp(string path)
```

Make a temporary filename.

Description

The *mktemp()* primitive shall replace the contents of the string *path* to by template by a unique filename and return template.

The application shall initialize template to be a filename with six trailing 'X's. *mktemp()* shall replace each X with a single byte character from the portable filename character set.

Returns

The *mktemp()* primitive shall return the string template. If a unique name cannot be created, template shall point to an empty string.

See Also

[fmktemp](#), [create_buffer](#)

output_file

```
int output_file([string filename])
```

Change the output file-name.

Description

The *output_file()* primitive changes the file-name associated with the current buffer to *filename*; the new name should be unique, it cannot be the file-name of an exist buffer or file.

By default the associated file-name associated with a buffer is the file-name specified on an [edit_file](#) or [create_buffer](#).

If *filename* is omitted the user shall be prompted as follows;

Enter new output file name:

Note:

Once changed backups shall be created under the new file-name, not the original name.

Parameters

`filename` Optional string containing the new output file-name, if omitted the user is prompted.

Returns

The `output_file()` primitive returns greater than zero on success, otherwise zero or less on error.

The following error conditions shall be reported to the user;

- `filename` must be a unique buffer.

Duplicate buffer name 'xxx'.

- Unless a system buffer stated `filename` must not already exist on the file-system.

Output file 'xxx' already exists.

Portability

n/a

See Also

[edit_file](#), [create_buffer](#)

read_ea

```
int read_ea(string filename,
            ...)
```

Read file extended information.

Description

The `read_ea()` primitive is reserved for future BRIEF compatibility.

Parameters

n/a

Returns

n/a

Portability

Provided for BRIEF compatibility.

See Also

[copy_ea_info](#), [set_ea](#)

read_file

```
int read_file( [string filename],
               [int glob = TRUE],
               [string encoding = NULL])
```

Read into the current buffer.

Description

The `read_file()` primitive reads the content of the specified file `filename` into the current buffer.

If `filename` is omitted the user shall be prompted as follows;

File to read:

Parameters

`filename` Optional string containing the file to read, if omitted the user shall be prompted.

`glob` Optional boolean flag, if either **TRUE** or omitted the filename shall be expanded, see [expandpath](#).

`encoding` Optional string containing the character encoding to be applied to the source file.

Returns

The *read_file()* primitive returns a positive value on success, 0 if the user was prompted and cancelled, otherwise -1 on error.

Portability

n/a

See Also

[create_buffer](#), [edit_file](#)

readlink

```
string|int readlink(string path,
                     [string &link])
```

Read the contents of a symbolic link.

Description

The *readlink()* primitive shall place the contents of the symbolic link referred to by *path* in the string *link*.

Returns

If *link* is omitted the return value is a *string*. Upon successful completion *readlink()* shall return the resolved link, otherwise it shall return an empty string and set the global *errno* to indicate the error.

Then *link* is given the return value is an *int*. Upon successful completion, *readlink* shall return the count of character placed in the string. Otherwise, it shall return a value of -1, and set the global *errno* to indicate the error.

Portability

A **GriefEdit** extension.

See Also

[lstat](#), [realpath](#), [symlink](#)

realpath

```
int realpath(string pathname,
            string resolved_path)
```

Resolve a pathname.

Description

The *realpath()* primitive shall derive, from the *pathname* an absolute pathname that names the same file, whose resolution does not involve ., .., or symbolic links.

Returns

The *realpath()* primitive returns 0 if successful otherwise -1 on error. When an error has occurred, the global *errno* contains a value indicating the type of error that has been detected.

Portability

A **GriefEdit** extension.

See Also

[getwd](#), [symlink](#)

remove

```
int remove(string filename)
```

Remove a file.

Description

The *remove()* primitive deletes the file whose name is contained within the string *filename*.

Returns

The *remove()* primitive returns zero if successful, and a non-zero value (-1)f otherwise. When an error has occurred, the global *errno* contains a value indicating the type of error that has been detected.

See Also

[exist](#), [access](#)

rename

```
int rename(string old,
          string new )
```

Rename a file.

Description

The *rename()* primitive causes the file whose name is indicated by the string *old* to be renamed to the name given by the string *new*.

Returns

The *rename()* primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global **errno** contains a value indicating the type of error that has been detected.

See Also

[remove](#), [create_buffer](#), [link](#), [unlink](#)

rmdir

```
int rmdir(string path)
```

Remove directory.

Description

The *rmdir()* primitive removes (deletes) the specified directory. The directory must not contain any files or directories. The path can be either relative to the current working directory or it can be an absolute path name.

Returns

The *rmdir()* primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global **errno** contains a value indicating the type of error that has been detected.

See Also

[mkdir](#), [chdir](#), [cd](#), [getwd](#)

searchpath

```
int searchpath(      [string path],
                    string filename,
                    [string extension],
                    string &result,
                    [int mode],
                    [int expand = FALSE])
```

Searches for a given file in a specified path.

Description

The *searchpath()* primitives searches the directory path *path* for a instance of the file name *file*. *path* is a list of delimiter separated directory names, with the same syntax as the shell variable *GRPATH*.

Parameters

<i>path</i>	Optional string containing the path to be searched for the file. If omitted the system <i>PATH</i> specification shall be referenced.
<i>filename</i>	String containing the name of the file for which to search.
<i>extension</i>	Optional string containing the file extension to be added to the file name when searching for the file. The first character of the file name extension must be a period (.). The extension is added only if the specified file name does not end with an extension. If a file name extension is not required or if the file name contains an extension, this parameter can be NULL.
<i>result</i>	String variable reference to be populated with the first instance of the file resolved along the given search path.

Returns

The *expandsearch()* primitive returns the length of the string that is copied to the buffer *result*; otherwise on failure returns a value of zero.

Portability

A **GriefEdit** extension.

See Also

[expandpath](#)

set_binary_size

```
int set_binary_size([int size])
```

Set binary chunk size.

Description

The *set_binary_size()* primitive sets the chunk or block size utilised when loading binary file images. Each chunk shall be represented within the buffer as a single line.

Note!:! If the specified value is equal or less-than zero (≤ 0), then files shall never be interpreted as binary when read; instead the default system specific type as be applied.

Parameters

size Optional integer number, if omitted the current size is not changed.

Returns

The *set_binary_size()* primitive returns the previous chunk size.

Portability

n/a

See Also

[inq_terminator](#)

set_ea

```
int set_ea(string filename,
           ...)
```

Set file extended information.

Description

The *set_ea()* primitive is reserved for future BRIEF compatibility.

Parameters

n/a

Returns

n/a

Portability

Provided for BRIEF compatibility.

See Also

[read_ea](#), [copy_ea_info](#)

set_file_magic

```
int set_file_magic([string encoding],
                  [int cost]          )
```

Define the file type detection rules.

Description

The *set_file_magic* primitive configures the global file character encoding detection logic.

Parameters

spec Optional file character encoding specification. If a non-empty string the detection rules shall be set to the given specification, whereas an empty string shall clear the current user specification, enabling the system default. If the specification is omitted the current rules remain unchanged.

cost Optional integer, stating the character cost the detection logic is permitted to incur.

Detection Types

The order below is somewhat important as the MBCS checks can result in false positives, as such are generally last in line.

Name	Default	Description
mark	yes	Explicit "Encoding: <marker>" within the leading file content.

Name	Default	Description
utf32bom	yes	UTF-32 BOM marker.
utf16bom	yes	UTF-16 BOM marker.
utf8	yes	UTF-8
bom	yes	Non UTF BOM markers.
udet	yes	Mozilla Universal Character Detector, see libcharudet .
magic	yes	File magic, see libmagic .
binary	yes	Possible binary image.
ascii	yes	ASCII only (7-bit).
latin1	yes	Latin-1 (ISO-8859-x) data.
big5	yes	Chinese Big5.
gb18030	yes	Chinese GB-18030.
shiftjis	yes	Shift-JIS.
xascii	yes	Extended ASCII.
charset	no	Explicit character-set.
guess	n/a	see libguess .

Without going into the full details of each search algorithms, several are summarised below.

mark

Markup languages have ways of specifying the encoding in a signature near the top of the file.

The following appears inside the <head> area of an HTML page.

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```

In XML, the XML declaration specifies the encoding.

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

Recognised formats are.

```
grief/vim  encoding:<xxx>
emacs      coding:<xxx>
html       charset=["]<xxx>["]
```

utf32bom, utf16bom

BOM stands for Byte Order Mark which literally is meant to distinguish between *little-endian* LE and *big-endian* BE byte order. For UTF-32 and UTF-16 the code point U+FEFF ("zero width no-break space") are used.

UTF-32 Big Endian	0X00,0X00,0XFE,0XFF
UTF-32 Little Endian	0XF0,0XFE,0X00,0X00

UTF-16 Big Endian	0XFE,0XFF
UTF-16 Little Endian	0XFF,0XFE

uft8

UTF-8 auto-detection (when there is no UTF-8 BOM), performs UTF-8 decoding on the file looking for an invalid UTF-8 sequence; correct UTF-8 sequences look like this:

0xxxxxxxxx	ASCII < 0x80 (128)
110xxxxx 10xxxxxx	2-byte >= 0x80
1110xxxx 10xxxxxx 10xxxxxx	3-byte >= 0x400
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4-byte >= 0x10000

bom

BOM stands for Byte Order Mark which literally is meant to distinguish between *little-endian* LE and *big-endian* BE byte order.

For Unicode files, the BOM ("Byte Order Mark" also called the signature or preamble) is a set of leading bytes at the

beginning used to indicate the type of Unicode encoding.

The key to the *BOM* is that it is generally not included with the content of the file when the file's text is loaded into memory, but it may be used to affect how the file is loaded into memory.

Recognised sequences include;

UTF-32 Big Endian	0X00,0X00,0XFE,0XFF
UTF-32 Little Endian	0XF0,0XFE,0X00,0X00

UTF-16 Big Endian	0XFE,0XFF
UTF-16 Little Endian	0XF0,0XFE

UTF-7	0X2B,0X2F,0X76,0X38
UTF-7	0X2B,0X2F,0X76,0X39
UTF-7	0X2B,0X2F,0X76,0X2B
UTF-7	0X2B,0X2F,0X76,0X2F

UTF-EBCDIC	0XDD,0X73,0X66,0X73
GB18030	0X84,0X31,0X95,0X33
BOCU-1	0XFB,0XEE,0X28,0xFF
BOCU-1	0XFB,0XEE,0X28
SCSU	0XOE,0XFE,0XFF
UTF-1	0XF7,0X64,0X4C
UTF-8	0XF,0XBB,0XBF

udet

Mozilla Universal Character Detector employs a composite approach that utilizes Code Scheme, Character Distribution and 2-Char Sequence Distribution methods to identify language/encodings has been proven to be very effective and efficient in our environment; see [libcharudet](#).

<http://www-archive.mozilla.org/~projects/intl-/UniversalCharsetDetection.html>

big5

Performs **Big5** decoding on the file looking for an invalid sequences.

Big5 does not conform to the ISO-2022 standard, but rather bears a certain similarity to Shift-JIS.

It is a double-byte character set with the following structure.

First Byte:	0xA1 - 0xf9 (non-user-defined characters)
	or 0x81 - 0xfe (extended range)
Second Byte:	0x40 - 0x7e or 0xa1 - 0xfe

gb18030

Performs **GB18030** decoding on the file looking for an invalid sequences.

GB18030-2000 has the following significant properties;

- It incorporates Unicode's CJK Unified Ideographs Extension A completely.
- It provides code space for all used and unused code points of Unicode's plane 0 (BMP) and its 15 additional planes. While being a code- and character-compatible "superset" of GBK, GB18030-2000, at the same time, intends to provide space for all remaining code points of Unicode. Thus, it effectively creates a one-to-one relationship between parts of GB18030-2000 and Unicode's complete encoding space.
- In order to accomplish the Unihan incorporation and code space allocation for Unicode 3.0, GB18030-2000 defines and applies a four-byte encoding mechanism.

GB18030-2000 encodes characters in sequences of one, two, or four bytes. The following are valid byte sequences (byte values are hexadecimal):

Single-byte:	0x00-0x7f
Two-byte:	0x81-0xfe + 0x40-0x7e, 0x80-0xfe
Four-byte:	0x81-0xfe + 0x30-0x39 + 0x81-0xfe + 0x30-0x39

The single-byte portion applies the coding structure and principles of the standard GB 11383 (identical to ISO 4873:1986) by using the code points of 0x00 through 0x7f.

The two-byte portion uses two eight-bit binary sequences to express a character. The code points of the first (leading) byte range from 0x81 through 0xfe. The code points of the second (trailing) byte ranges from 0x40 through 0x7e and 0x80 through 0xfe.

The four-byte portion uses the code points 0x30 through 0x39, which are vacant in GB 11383, as an additional means to extend the two-byte encodings, thus effectively increasing the number of four-byte codes to now include code points ranging from 0x81308130 through 0xfe39fe39.

GB18030-2000 has 1.6 million valid byte sequences, but there are only 1.1 million code points in Unicode, so there are about 500,000 byte sequences in GB18030-2000 that are currently unassigned.

shiftjis

Performs **Shift-JIS** decoding on the file looking for an invalid sequences.

- Single Byte

ASCII:	0x21 – 0x7F (also allow control)
Katakana:	0xA1 – 0xDF

- Multiple Byte

JIS X 0208 character
First byte: 0x81 – 0x9F or 0xE0 – 0xEF
Second byte (old 1st): 0x40 – 0x9E
Second byte (even 1st): 0xA0 – 0xFD

guess

libguess employs discrete-finite automata to deduce the character set of the input buffer. The advantage of this is that all character sets can be checked in parallel, and quickly. Right now, *libguess* passes a byte to each DFA on the same pass, meaning that the winning character set can be deduced as efficiently as possible; see [libguess](#).

Returns

The `set_file_magic()` primitive returns a positive value on success, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[edit_file](#), [inq_file_magic](#)

stat

```
int stat( [string path],
          [int size],
          [int mtime],
          [int ctime],
          [int atime],
          [int mode],
          [int uid],
          [string uid2name],
          [int gid],
          [string gid2name],
          [int nlink],
          [int inode]      )
```

Obtain file information.

Description

The `stat()` primitive obtain information about the file or directory referenced in *path*.

This information is returned in the parameters following the *path* parameter (if supported by the underlying filesystem).

- *size* - Total file size, in bytes.
- *mode* - File mode ((see [File Modes](#))).
- *mtime* - The files "last modified" time (See: [time](#)).
- *atime* - Time the file was "last accessed".
- *ctime* - Time of the files "last status change".

- uid - User identifier.
- uid2name - User name associated with the file *uid*.
- gid - Group identifier.
- gid2name - Group name associated with the file *gid*.
- nlink - Number of hard links.
- inode - File-system internal node number.

Parameters

- For regular files, the file size in bytes.

`path` String containing the file path. If omitted the statistics of the current buffer shall be retrieved.

`size` Optional integer, if specified is populated with the size of the related file in bytes.

- For symbolic links, the length in bytes of the path-name contained in the symbolic link.
 - For a shared memory object, the length in bytes.
 - For a typed memory object, the length in bytes.
 - For other file types, the use of this field is unspecified.

`mtime` Optional integer, populated with the files modification time (See: [time](#)).

`ctime` Optional integer, populated with the time of the last status change.

`atime` Optional integer, populated with the last access time.

`mode` Optional integer, mode of file ((see [File Modes](#))).

`uid` Optional integer, user identifier of the file.

`uid2name` User name associated with the file `uid`.

`gid` Optional integer, group identifier of the file.

`gid2name` Group name associated with the file `gid`.

`nlink` Optional integer, number of hard links to the file.

`inode` Optional integer, populated with the file-system internal node number.

Returns

The `stat()` primitive returns zero if successful, and a non-zero value (-1) otherwise. When an error has occurred, the global `errno` contains a value indicating the type of error that has been detected.

Portability

n/a

See Also

[lstat](#), [access](#), [exist](#), [fstat](#)

symlink

```
int symlink(string path1,
           string path2 )
```

Create a symbolic link.

Description

The `symlink()` primitive shall create a symbolic link called *path2* that contains the string *path1*.

In other words, *path2* is the name of the symbolic link created, *path1* is the string contained in the symbolic link.

If the `symlink()` primitive fails for any reason other than any file named by *path2* shall be unaffected.

Returns

Upon successful completion, `symlink()` shall return 0; otherwise, it shall return -1 and set the global `errno` to indicate the error.

Portability

A [GriefEdit](#) extension.

See Also

[lstat](#), [readlink](#)

umask

```
int umask(int cmask = NULL)
```

Set and get the file mode creation mask.

Description

The *umask()* primitive shall set the processes file mode creation mask to *cmask* and return the previous value of the mask.

Only the file permission bits of *cmask* are used; the meaning of the other bits is implementation-defined.

The process' file mode creation mask is used to turn off permission bits in the mode argument supplied during calls to the following functions:

- [fopen](#)
- [create_buffer](#)
- [mkdir](#)

Returns

The file permission bits in the value returned by *umask()* shall be the previous value of the file mode creation mask.

See Also

[chmod](#), [stat](#), [lstat](#)

unlink

```
int unlink(string path)
```

Unlink a file.

Description

The *unlink()* primitive is reserved for future use.

The *unlink()* primitive shall remove a link to a file. If path names a symbolic link, *unlink()* shall remove the symbolic link named by path and shall not affect any file or directory named by the contents of the symbolic link. Otherwise, *unlink()* shall remove the link named by the pathname pointed to by path and shall decrement the link count of the file referenced by the link.

When the file's link count becomes 0 and no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have the file open when the last link is removed, the link shall be removed before *unlink()* returns, but the removal of the file contents shall be postponed until all references to the file are closed.

The path argument shall not name a directory unless the process has appropriate privileges and the implementation supports using *unlink()* on directories.

Returns

Upon successful completion, *link* shall return 0; otherwise, it shall return -1 and set the global [errno](#) to indicate the error.

Portability

A [GriefEdit](#) extension.

See Also

[link](#), [symlink](#), [stat](#), [remove](#)

vfs_mount

```
int vfs_mount( string mountpoint,
               int      flags           = 0,
               string  vfsname,
               [string arguments]     )
```

Mount a virtual file-system.

Description

The *vfs_mount()* primitive mounts a virtual file-system.

Parameters

mountpoint String containing the mount point, being the logical path representing the root of the mounted resource.

flags Integer mount flags.

vfsname String containing the virtual file-system driver to be applied.

arguments Optional string arguments to be passed upon the underlying vfs implementation; the format and values required are specific to the virtual file-system driver referenced within *vfsname*.

Returns

The `vfs_mount()` primitive returns 0 on success, otherwise -1 on error and `errno` contains a value indicating the type of error that has been detected.

Portability

A **GriefEdit** extension.

See Also

`vfs_unmount`, `inq_vfs_mounts`

vfs_unmount

```
int vfs_unmount(string mountpoint,
                int      flags      = 0)
```

Unmount a virtual file-system.

Description

The `vfs_unmount()` primitive unmounts the specified virtual file-system referenced by `mountpoint`.

Parameters

`mountpoint` String containing the mount point, being the logical path representing the root of the mounted resource.
`flags` Optional integer unmount flags.

Returns

The `vfs_unmount()` primitive returns 0 on success, otherwise -1 on error and `errno` contains a value indicating the type of error that has been detected.

Portability

A **GriefEdit** extension.

See Also

`vfs_mount`, `inq_vfs_mounts`

File Modes

Traditional Unix file mode consist of a number of components including type, permissions including special bits.

The `mode_string` primitive creates a human readable string version of these bits in a system independent form.

Type

The file type is represented by the following constants.

<code>S_IFBLK</code>	Block special.
<code>S_IFCHR</code>	Character special.
<code>S_IFIFO</code>	FIFO special.
<code>S_IFREG</code>	Regular.
<code>S_IFDIR</code>	Directory.
<code>S_IFLNK</code>	Symbolic link.
<code>S_IFSOCK</code>	Socket.

Classes

Permissions on Unix-like systems are managed in three distinct classes. These classes are known as user, group, and others.

Files and directories are owned by a *user*. The owner determines the file's owner class. Distinct permissions apply to the owner.

Files and directories are assigned a *group*, which define the file's group class. Distinct permissions apply to members of the file's group. The owner may be a member of the file's group.

Users who are not the owner, nor a member of the group, comprise a file's *others* class. Distinct permissions apply to *others*.

The effective permissions are determined based on the user's class. For example, the user who is the owner of the file will have the permissions given to the owner class regardless of the permissions assigned to the group class or others class.

Permissions

For each group there are three specific permissions on Unix-like systems that apply to each class:

- Read permission - grants the ability to read a file. When set for a directory, this permission grants the ability to read the names of files in the directory (but not to find out any further information about them such as contents, file type, size, ownership, permissions, etc.)
- Write permission - grants the ability to modify a file. When set for a directory, this permission grants the ability to modify[clarify] entries[clarify] in the directory. This includes creating files, deleting files, and renaming files.
- Execute permission - grants the ability to execute a file. This permission must be set for executable binaries (for example, a compiled C++ program) or shell scripts (for example, a Perl program) in order to allow the operating system to run them. When set for a directory, this permission grants the ability to access file contents and metainfo if its name is known, but not list files inside the directory (unless read is set).

The file permissions are represented by the following constants.

Read, write, execute/search by owner.

`S_IRUSR` Read permission, owner.

`S_IWUSR` Write permission, owner.

`S_IXUSR` Execute/search permission, owner.

Read, write, execute/search by group.

`S_IRGRP` Read permission, group.

`S_IWGRP` Write permission, group.

`S_IXGRP` Execute/search permission, group.

Read, write, execute/search by others.

`S_IROTH` Read permission, others.

`S_IWOTH` Write permission, others.

`S_IXOTH` Execute/search permission, others.

Special Bits

Unix-like systems typically employ three additional modes. These are actually attributes but are referred to as permissions or modes. These special modes are for a file or directory overall, not by a class.

- The set user ID, setuid, or SUID mode. When a file with setuid is executed, the resulting process will assume the effective user ID given to the owner class. This enables users to be treated temporarily as root (or another user).
- The set group ID, setgid, or SGID permission. When a file with setgid is executed, the resulting process will assume the group ID given to the group class. When setgid is applied to a directory, new files and directories created under that directory will inherit the group from that directory. (Default behaviour is to use the primary group of the effective user when setting the group of new files and directories.)
- The sticky mode, when on a directory, the sticky permission prevents users from renaming, moving or deleting contained files owned by users other than themselves, even if they have write permission to the directory. Only the directory owner and superuser are exempt from this.

The special bits are represented by the following constants.

`S_ISUID` Set-user-ID on execution.

`S_ISGID` Set-group-ID on execution.

`S_ISVTX` On directories, restricted deletion flag. [Option End]

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Keyboard Primitives

Summary

Keyboard Primitives

Macros

<code>assign_to_key</code>	Assign command to key or key sequence.
<code>copy_keyboard</code>	Copy a keyboard.
<code>get_mouse_pos</code>	Retrieve last mouse action.
<code>input_mode</code>	Effect of certain system keys.
<code>inq_assignment</code>	Get key assignment for function.
<code>inq_kbd_char</code>	Peek at the keyboard.
<code>inq_kbd_flags</code>	Get keyboard key flags.
<code>inq_kbd_name</code>	Retrieve the assigned keyboard name.
<code>inq_keyboard</code>	Retrieve the keyboard identifier.
<code>inq_keystroke_macro</code>	Retrieve the current keystroke macro.
<code>inq_keystroke_status</code>	Determine keystroke macro status.
<code>inq_local_keyboard</code>	Retrieve local keyboard identifier.
<code>inq_mouse_action</code>	Retrieve the keyboard mouse handler.
<code>inq_mouse_type</code>	Retrieve the button type.
<code>inq_remember_buffer</code>	Determine the keystroke buffer name.
<code>int_to_key</code>	Convert an keycode to mnemonic key string.
<code>key_list</code>	Retrieve keyboard bindings.
<code>key_to_int</code>	Convert key name to a code.
<code>keyboard_flush</code>	Flush the keyboard buffer.
<code>keyboard_pop</code>	Pop a keyboard from the keyboard stack.
<code>keyboard_push</code>	Push a keyboard onto the keyboard stack.
<code>keyboard_typeables</code>	Assign self_insert to all typeable keys.
<code>load_keystroke_macro</code>	Load a recorded macro from a file.
<code>pause</code>	Pause keystroke definition.
<code>playback</code>	Replay a keystroke macro.
<code>process</code>	Invoke a Grief engine.
<code>push_back</code>	Push back a character into the keyboard.
<code>read_char</code>	Read next key from the keyboard.
<code>remember</code>	Start remembering keystrokes.
<code>save_keystroke_macro</code>	Save the current keystroke macro.
<code>set_kbd_name</code>	Set the keyboard name.
<code>set_mouse_action</code>	Set keyboard mouse handler.
<code>set_mouse_type</code>	Sets the mouse type.
<code>undo</code>	Undo previous edit operations.
<code>use_local_keyboard</code>	Associate a keyboard with a buffer.

Macros

`assign_to_key`

```
int assign_to_key([string key],
                 [string macro])
```

Assign command to key or key sequence.

Description

The `assign_to_key` primitive assigns a key or key sequence to a function.

The key is defined by the string parameter `key`, which if omitted shall be prompted. `key` may be specified as either an internal key code, or may be specified by a portable abbreviation, see below.

A key is "unassigned" by assigning the function `nothing` to the key.

If `key` evaluates to more than one keystroke, this shall define a multikey sequence, that is more than one key must be pressed to execute the macro. In this case, an internal key code is assigned for the key. This internal key code is returned as the value of this macro. Multikey sequences do not time out between key presses unlike the other internal keys when there is an ambiguity.

Parameters

`key` String identifying the particular keystroke that is being assigned.

macro String containing the command which shall be invoked, which may contain literal integer and/or string arguments to be passed upon the macro upon execution.

Key Sequences

Generally keys are defined using their associated mnemonic possibility prefixed with one or modifiers enclosed within a set of "<>" brackets, examples.

- <F1>
- <Ctrl-F1>
- <Alt-Ctrl-Up>.

The following are the set of supported modifiers

- Shift - Key Shift, for example a to Z.
- Ctrl - Control Key.
- Alt - Alt Key.
- Meta - Alias for Alt.
- Keypad - Keypad key variate.

In the case of simple ASCII keys their character value can be used, for example "a".

In addition to a mnemonic "<xxx>" or ASCII "x" syntax additional alternative forms and special characters are supported.

#xxx Substitutes the # lead sequence of digits with the represent value. For example #123 result in the key code 123.

^x The ^ characters treats the following character as a control code, For example ^a, is control-a.

\ The \ character escapes the next character, removing any special meaning.

When it is required to state any of the special characters "<, >, %, # or ^" these can be referenced using their escaped form, example "\\<".

The following table details the supported key sequence encoding and suitable modifiers; all names are matched case insensitive.

For examples review current supplied macro code.

Key	Description	Keypad	Shift	Ctrl	Alt	Meta
ASCII	ASCII key		x	x	x	x
F1..F12	Function keys		x	x	x	x
PgDn	Page Down					
PgUp	Page Up					
Left	Cursor Left	x	x	x	x	
Right	Cursor Right	x	x	x	x	
Up	Cursor Up	x	x	x	x	
Down	Cursor Down	x	x	x	x	
Tab						
Back-Tab	Shifted Tab					
Backspace						
Back						
Del	Delete					
Enter	Enter/Return Key	x				
Esc	Escape key					
Space	Space ()					
Home	Cursor Home	x				
End	Cursor End	x				
Ins	Insert	x				
Plus	Plus (+)	x				
Minus	Minus (-)	x				
Star	Multiply (*)	x				
Divide	Div (/)	x				
Equals	Equal (=)	x				
Cancel	Cancel Key					
Command	Command Key					
Copy	Copy Key					
Cut	Cut Key					
Exit	Exit Key					
Help	Help Key					
Menu	Menu Key					
Next	Next Key					
Open	Open key					

Key	Description	Keypad	Shift	Ctrl	Alt	Meta
Paste	Paste key					
Prev	Prev Key					
Prtsc	Print-Screen Key					
Redo	Redo Key					
Replace	Replace					
Save	Save					
Scroll	Scroll					
Search	Search					
Undo	Undo					
Keypad-#	Keypad 0..9	x	x	x	x	
Grey-#	Aliases for keypad					
Button#	Button number #					
Button#-Up						
Button#-Double						
Button#-Motion						
Button#-Down						
Private#	Private keys					
Mouse	Special Mouse Event					
Wheel-Up	Mousewheel up movement					
wheel-Down	Mousewheel down					

Returns

The `assign_to_key` returns the key value assigned to sequence, otherwise -1 if the key sequence is invalid or the operation aborted.

Portability

n/a

See Also

[key_to_int](#)

copy_keyboard

```
int copy_keyboard( int kbdid,
                  [string cmd ...])
```

Copy a keyboard.

Description

The `copy_keyboard()` primitive copies the key bindings associated with the keyboard identifier `kbdid`. The keyboard is either copied in its entirety or optionally the subset of key bindings associated with the specified list of commands `cmd`.

The primary use of this primitive is to obtain an explicit subset of a full keyboard for a specific use; for example a keyboard to be used by an popup for a special editing window. Another way of thinking, it a macro can inherit a keyboard, modify and then utilise locally without knowledge of the callers keyboard bindings.

Parameters

`kbdid` Keyboard identifier.

... Optional list of command names whose key assignments should be duplicated in the current keyboard. If no commands are given, then this just creates a duplicate keyboard.

Returns

The `copy_keyboard()` primitive returns the Keyboard identifier of the new keyboard. On error 0 is returned if the commands given are not found.

Portability

n/a

See Also

[keyboard_push](#), [keyboard_pop](#), [assign_to_key](#)

get_mouse_pos

```
void get_mouse_pos( [int &x],
                    [int &y],
                    [int &winnum],
                    [int &line],
                    [int &col],
                    [int &where],
                    [int &region],
                    [int &event] )
```

Retrieve last mouse action.

Description

The `get_mouse_pos()` primitive retrieves the details of the last mouse action.

Note:

The details returned are only valid immediately after the macro assigned to a button press event. Any subsequent calls to `read_char` or `process` will overwrite the internal values.

Parameters

<code>x, y</code>	Screen coordinates.
<code>winnum</code>	Optional an integer variable which shall be populated with the window identifier within which the (x, y) position falls, otherwise -1 if no window was mapped.
<code>line, col</code>	Optional integer variables which shall be populated with the translated window coordinates, otherwise -1 if no window was mapped.
<code>where</code>	Where the cursor is located within the window.
<code>region</code>	Cursor position relative to any marked regions.
<code>event</code>	Associated mouse event.

Where

Value	Definition
<code>MOBJ_NOWHERE</code>	Not in any window.
<code>MOBJ_LEFT_EDGE</code>	Left bar of window.
<code>MOBJ_RIGHT_EDGE</code>	Right bar of window.
<code>MOBJ_TOP_EDGE</code>	Top line of window.
<code>MOBJ_BOTTOM_EDGE</code>	Bottom line of window.
<code>MOBJ_INSIDE</code>	Mouse inside window.
<code>MOBJ_TITLE</code>	On title.
<code>MOBJ_VSCROLL</code>	Vertical scroll area.
<code>MOBJ_VTHUMB</code>	Vertical scroll area.
<code>MOBJ_HSCROLL</code>	Horizontal scroll area.
<code>MOBJ_HTHUMB</code>	Horizontal scroll area.
<code>MOBJ_ZOOM</code>	Zoom button,
<code>MOBJ_CLOSE</code>	Close.
<code>MOBJ_SYSMENU</code>	System Menu.

Region

Value	Description
0	No region selected.
1	Cursor is before the region.
2	Cursor is inside the region.
3	Cursor is after the region.
4	Cursor is to the left of a column region.
5	Cursor is to the right of a column region.

Returns

`nothing`

Portability

n/a

See Also

`process_mouse`, `translate_pos`

input_mode

```
int input_mode(int char,
               int flag )
```

Effect of certain system keys.

Description

The *input_mode()* primitive controls the effect of certain characters when typed, setting the actions of character *char* to the desired *state*.

Normally the keyboard driver acts internal on specific characters which perform the control actions of XON/XOFF and the Job control stop; specifically Ctrl-S (0x13), Ctrl-Q (0x11) and Ctrl-Z (0x1a).

Parameters

char Integer value of the ascii character to be effected.

flags Boolean flag, if *true* enable the specified character to flow thru to GRIEF otherwise **false** to reset the default behaviour.

Returns

The *input_mode()* primitive returns 1 if character previously enabled; zero otherwise.

Portability

n/a

See Also

[inq_keyboard](#)

inq_assignment

```
string inq_assignment(int|string val,
                      [int tokey = FALSE])
```

Get key assignment for function.

Description

The *inq_assignment()* primitive either retrieves the command that is assigned to the particular key *val* or the key sequence(s) that will invoke a specific command.

Parameters

val String denoting the key sequence to be decoded or an integer representing the internal key code. The string representation should be of the form described by [assign_to_key](#).

tokey Optional boolean value, if **true** then *val* is taken as a macro name and the keys assigned to invoke this macro are returned. The key assignment returned is returned using the portable key definitions defined for [assign_to_key](#).

Returns

The *inq_assignment()* primitive returns a string containing the desired conversion.

For key sequence to command: the name of the command; OR "nothing" if no command is assigned; OR "ambiguous" if there is more than key sequence starting with the *val*.

For command to key sequence enabled when *tokey* is non-zero

a list of the valid key sequences for the command. The list elements are formatted with the following separators:

- or Separates different ways of specifying a single key.
- and Separates keys in a multi-key sequence.
- also Separates multiple (different) key sequences.

Portability

n/a

See Also

[assign_to_key](#)

inq_kbd_char

```
int inq_kbd_char()
```

Peek at the keyboard.

Description

The *inq_kbd_char()* primitive determines whether a character is available to be retrieved from the keyboard using *read_char*.

Note:

This primitive only tests the internal look ahead buffer not the external terminal and/or keyboard stream.

Parameters

none

Returns

The *inq_kbd_char()* primitive returns non-zero if one or more characters are available, otherwise 0 if the keyboard buffer is empty.

Portability

n/a

See Also

[read_char](#)

inq_kbd_flags

```
int inq_kbd_flags()
```

Get keyboard key flags.

Description

The *inq_kbd_flags()* primitive returns the state of the special keyboard keys.

Bit	Definition
0x01	Right Shift
0x02	Left Shift
0x04	Ctrl
0x08	Alt
0x10	Scroll
0x20	Num Lock
0x40	Caps Lock

Parameters

none

Returns

Always returns 0.

Portability

Provided only for BRIEF compatibility.

See Also

[inq_kbd_char](#)

inq_kbd_name

```
string inq_kbd_name([int kbdid])
```

Retrieve the assigned keyboard name.

Description

The *inq_kbd_name()* primitive retrieves the name assigned to the keyboard *kbdid* using <set_kdb_name>.

Parameters

kbdid Optional integer keyboard identifier, if omitted the current keyboard shall be referenced.

Returns

The `inq_kbd_name()` primitive retrieves the keyboard name otherwise an empty string is none has been assigned.

Portability

n/a

See Also

[set_kbd_name](#)

inq_keyboard

```
int inq_keyboard()
```

Retrieve the keyboard identifier.

Description

The `inq_keyboard()` primitive retrieves the identifier associated with the current keyboard.

Parameters

none

Returns

The `inq_keyboard()` primitive returns the current keyboard identifier.

Portability

n/a

See Also

[keyboard_pop](#), [keyboard_push](#)

inq_keystroke_macro

```
string inq_keystroke_macro([int macroid],
                           [int &bufnum] )
```

Retrieve the current keystroke macro.

Description

The `inq_keystroke_macro()` primitive retrieves a string containing the definition of the current keyboard macro, which may then be saved in a file and reloaded.

The returned definition may be edited and/or saved for future use; see [load_keystroke_macro](#).

Note:

The primitive and the returned definition is a useful reference when developing customised macros.

Consult the `remember` source as an example how this buffer can be saved.

Parameters

`macroid` Optional integer macro identifier specifies the keyboard against which to derive the buffer name, if omitted the current macro identifier is utilised.

`bufnum` Omitted integer variable if supplied shall be populated with the associated buffer number.

Returns

nothing

Portability

Under BRIEF this primitive behaved like the functionality available via [inq_keystroke_status](#).

See Also

[remember](#), [load_keystroke_macro](#)

inq_keystroke_status

```
int inq_keystroke_status([int &macroid])
```

Determine keystroke macro status.

Description

The *inq_keystroke_macro()* primitive determines whether keystroke macro record or playback is active.

Parameters

macroid Optional integer variable if supplied shall be populated with the current macro identifier.

Returns

The *inq_keystroke_macro()* primitive returns greater than zero if a keystroke macro is being recorded or played back.

Portability

Under BRIEF this primitive is named *inq_keystroke_macro*.

See Also

[remember](#), [playback](#)

inq_local_keyboard

```
int inq_local_keyboard()
```

Retrieve local keyboard identifier.

Description

The *inq_local_keyboard()* primitive retrieves the identifier associated with current local keyboard.

Parameters

none

Returns

The *inq_local_keyboard()* primitive returns the associated keyboard identifier, otherwise 0 if none is available.

Portability

n/a

See Also

[use_local_keyboard](#)

inq_mouse_action

```
string inq_mouse_action()
```

Retrieve the keyboard mouse handler.

Description

The *inq_mouse_button()* primitive is reserved for future BRIEF compatibility.

The *inq_mouse_action()* primitive retrieves the name of the current mouse action handler. This function can be used to save the current mouse handle prior to pushing a new keyboard.

Parameters

none

Returns

Returns the current mouse action.

Portability

n/a

See Also

[set_mouse_action](#)

inq_mouse_type

```
int inq_mouse_type()
```

Retrieve the button type.

Description

The *inq_mouse_type()* primitive retrieves the current mouse type.

Parameters

none

Returns

Returns the current mouse type.

Value	Description
0	No mouse.
1	One-button mouse.
2	Two-button mouse.
3	Three-button mouse.

Portability

n/a

See Also**inq_remember_buffer**

```
string inq_remember_buffer([int macroid])
```

Determine the keystroke buffer name.

DescriptionThe *inq_remember_buffer()* primitive derives the buffer name associated with the keyboard macro *macro*.Unlike BRIEF multiple keystroke macros can be maintained. Each remember execution shall create a buffer named *KBD-MACRO-#* where # is the associated keyboard macro identifier. This buffer maybe then edited and/or saved to later use.**Parameters**

macroid Optional integer macro identifier specifies the keyboard against which to derive the buffer name, if omitted the current macro identifier is utilised.

ReturnsThe *inq_remember_buffer()* primitive returns the buffer name associated with the remember buffer.**Portability**

n/a

See Also[remember](#)**int_to_key**

```
string int_to_key(int key)
```

Convert an keycode to mnemonic key string.

DescriptionThe *int_to_key()* primitive generates the corresponding mnemonic representation for the specified keycode *key*; which can be used by [assign_to_key](#)**Parameters**

key Integer keycode.

ReturnsThe *int_to_key()* primitive returns the key mnemonic string associated with the specified keycode, for example "<Ctrl-D>"; see [assign_to_key](#) for full description of key mnemonics.**Portability**

n/a

See Also[assign_to_key](#)

key_list

```
list key_list( int kbdid,
               [int self_inserts = 0],
               [int bufnum] )
```

Retrieve keyboard bindings.

Description

The `key_list()` primitive retrieves the key bindings associated with the keyboard `kbdid`. The definitions are returned as a list of string pairs, the first element being the key name with the second being assigned macro.

Up to two keyboards can be specified. If `bufnum` is specified, then the local keyboard assigned to that buffer is used. If not specified then the local keyboard map of the current buffer is used. The returned list is a union of the set keyboards.

This primitive is designed to display a list of all valid keys currently mapped in both the current local and global keyboard maps. For an example see the `key_map` macro.

Parameters

<code>kbdid</code>	Optional integer keyboard identifier. If omitted then firstly the local keyboard if available is referenced otherwise the current keyboard is referenced.
<code>self_inserts</code>	Optional boolean flag when non-zero keys assigned to <code>self_insert</code> shall be including, otherwise they are omitted from the generated list.
<code>bufnum</code>	Optional buffer identifier to source the secondary local keyboard. If omitted then the local keyboard map of the current buffer is used.

Returns

The `key_list()` primitive returns a list of key binding, as a set of string pairs, the first element is the key name and second being assigned macro to that key. Otherwise on error a null list.

Portability

n/a

See Also

`keyboard_pop`, `keyboard_push`

key_to_int

```
int key_to_int(string key,
               int raw )
```

Convert key name to a code.

Description

The `key_to_int()` primitive converts a mnemonic key string to an integer.

The following scheme is utilised for encoding internal key-codes, allowing for simple conversion of ASCII character code to the internal codes and vice-versa.

Firstly key-codes are divided into several ranges.

Key Code	Range	Description
RANGE_CHARACTER	0x0 ... 1fffff	Character ASCII/Unicode range.
RANGE_FUNCTION	0x02000...	Function keys.
RANGE_KEYPAD	0x03000...	Keypad keys.
RANGE_MISC	0x04000...	Miscellaneous.
RANGE_MULTIKEY	0x05000...	Multi-key stroke.
RANGE_PRIVATE	0x06000...	Private key definitions for users.
RANGE_BUTTON	0x07000...	Mouse buttons and movement.

These ranges can be OR'ed with one or more of the following bits to indicate a modifier key, for example a `Shift-F1` key or `Ctrl-Shift-F2`.

Modifier	Code	Description
MOD_SHIFT	0x00200000	Shift'ed.
MOD_CTRL	0x00400000	Control.
MOD_META	0x00800000	Meta or Alt.

To further simplify key handling, the follow special key manifest constants are predefined.

Key Code	Description
CTRL_1 .. CTRL_10	Control 1 thru 10.
ALT_1 .. ALT_10	Alt 1 thru 10.
CTRL_A .. CTRL_Z	Control A thru Z.
ALT_Z .. ALT_Z	Alt A thru Z.
KEY_BACKSPACE	Backspace.
KEY_BREAK	Break.
KEY_CANCEL	Cancel key.
KEY_CLOSE	Close key.
KEY_COMMAND	
KEY_COPY	Copy to clipboard.
KEY_COPY_CMD	
KEY_CUT	Cut to clipboard.
KEY_CUT_CMD	
KEY_DEL	Delete, rubout.
KEY_DOWN	Move down, down arrow.
KEY_END	End key.
KEY_ENTER	Enter key.
KEY_ESC	Escape.
KEY_EXIT	
KEY_HELP	Help, usage.
KEY_HOME	Home key.
KEY_INS	Insert.
KEY_LEFT	Move left, left arrow.
KEY_MENU	Menu key.
KEY_NEWLINE	New line.
KEY_NEXT	Next.
KEY_OPEN	Open key.
KEY_PAGEDOWN	Page down.
KEY_PAGEUP	Page up.
KEY_PASTE	Paste clipboard.
KEY_PREV	Prior, previous.
KEY_REDO	Redo, again.
KEY_REPLACE	
KEY_RIGHT	Move right, right arrow.
KEY_SAVE	
KEY_SEARCH	Search.
KEY_TAB	Tab.
KEY_UNDO	Undo key.
KEY_UNDO_CMD	Undo key.
KEY_UP	Move up, up arrow.
KEY_WDOWN	
KEY_WDOWN2	
KEY_WLEFT	
KEY_WLEFT2	
KEY_WRIGHT	
KEY_WRIGHT2	
KEY_WUP	
KEY_WUP2	
KEY_VOID	NUL key-code.

Warning:

The key encoding are exposed only for reference and may change without notice; for example to fully support Unicode. As such its advised keys should only be referenced using their string mnemonic representation when dealing directly with key-codes, for example.

```

switch (keycode) {
    case key_to_int("<F1>"):
        f1();
        break;
    case key_to_int("<Ctrl-A>"):
        ctrlA();
        break;
    case key_to_int("<Ctrl-B>"):
        ctrlB();
        break;
    :
}

```

Parameters

key String contains a single mnemonic key description (like "i" or "<Ctrl-z>").

raw Optional boolean flag. If non-zero, then **key** is taken to be a raw key stroke/escape sequence, as entered by a function key on the keyboard. In this case, the key-code assigned to that key is returned.

Returns

The **key_to_int()** primitive returns an integer representing the internal key code assigned to the specified key sequence, or -1 if the sequence does not correspond to a valid key.

Portability

n/a

See Also

[assign_to_key](#), [int_to_key](#)

keyboard_flush

```
void keyboard_flush()
```

Flush the keyboard buffer.

Description

The **keyboard_flush()** primitive flushes the keyboard input buffer, consuming any pending input, permanently removing the keystrokes.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[inq_kbd_char](#), [read_char](#)

keyboard_pop

```
void keyboard_pop([int save = FALSE])
```

Pop a keyboard from the keyboard stack.

Description

The **keyboard_pop()** primitive removes the top keyboard, from the last-in/first-out (LIFO) keyboard stack.

Each invocation of **keyboard_push** must have a corresponding invocation of **keyboard_pop**.

The keyboard resource and associated identifier shall only remain valid if the **save** argument is non-zero, otherwise the keyboard definition is deleted and its memory reclaimed.

If the current buffer is the same as it was when the keyboard was pushed, the current local keyboard is also restored to its former value.

Parameters

save Optional boolean value, if specified as **true** the keyboard resource is retained otherwise if **false** or omitted it is discarded.

Returns

nothing

Portability

n/a

See Also

[keyboard_push](#), [inq_keyboard](#)

keyboard_push

```
void keyboard_push([int kbdid])
```

Push a keyboard onto the keyboard stack.

Description

The *keyboard_push()* primitive pushes a keyboard, either an existing or new, onto the last-in/first-out (LIFO) keyboard stack. Being a stack each invocation of *keyboard_push* must have a corresponding invocation of *keyboard_push*; otherwise the keyboard objects shall never be released, resulting in a resource/memory leak.

Either the specified keyboard *kbdid* or if omitted a new empty keyboard object shall be pushed onto the stack and become the current keyboard (See: [inq_keyboard](#)). When a new keyboard is created, it is empty and the current local keyboard is temporarily unassigned.

A keyboard resource is a table of key bindings, that is keys mapped against the macro which should be executed when encountered. During initialisation the default keyboard resource binds the usual editing keys, *keyboard_push()* permits the creation of macro specific keyboards, supporting temporarily bind macros to keys for their operation.

Defaults

The default key bindings along with *keyboard_typeables* includes -

Key	Macro
F1	change_window
F2	move_edge
F3	create_edge
F4	delete_edge
F5	search_fwd
F6	translate
F7	remember
F8	playback
F9	load_macro
F10	execute_macro
Shift-F7	pause
Ins	paste
End	end_of_line
Down	down
Left	left
Right	right
Home	beginning_of_line
Up	up
Del	delete_block
Cut	cut
Copy	copy
Undo	undo
Page-Down	page_down
Page-Up	page_up
Wheel-Down	page_down
Wheel-Up	page_up
Shift-Keypad-2	"change_window 2", see change_window
Shift-Keypad-4	"change_window 3"
Shift-Keypad-6	"change_window 1"
Shift-Keypad-8	"change_window 0"
Ctrl-Keypad-1	end_of_window
Ctrl-Keypad-3	end_of_buffer
Ctrl-Keypad-7	top_of_window

Key	Macro
Ctrl-Keypad-9	top_of_buffer
Ctrl-X	exit
Alt-0	"drop_bookmark 0", see drop_bookmark
Alt-1	"drop_bookmark 1"
Alt-2	"drop_bookmark 2"
Alt-3	"drop_bookmark 3"
Alt-4	"drop_bookmark 4"
Alt-5	"drop_bookmark 5"
Alt-6	"drop_bookmark 6"
Alt-7	"drop_bookmark 7"
Alt-8	"drop_bookmark 8"
Alt-9	"drop_bookmark 9"
Alt-A	"mark 4", see mark
Alt-B	<buffer_list>
Alt-C	"mark 2", see mark
Alt-D	delete_line
Alt-E	edit_file
Alt-F	<feature>
Alt-G	goto_line
Alt-I	insert_mode
Alt-J	goto_bookmark
Alt-K	delete_to_eol
Alt-L	mark
Alt-M	mark
Alt-O	output_file
Alt-P	print
Alt-Q	<quote>
Alt-R	read_file
Alt-S	search_fwd
Alt-T	translate
Alt-U	undo
Alt-V	version
Alt-W	write_buffer
Alt-X	exit
Alt-Z	shell

Parameters

`kbdid` Integer keyboard identifier, if omitted or zero a new keyboard is created and pushed. Keyboards within the stack are assumed to be unique; beware pushing a keyboard which already exists which shall have undefined effects.

Returns

The `keyboard_push()` primitive returns the identifier of the referenced keyboard, otherwise -1 on error.

Example

The following example creates a new keyboard with only typeable key binds active.

```
keyboard_pop();
keyboard_push();
keyboard_typeables();
```

The following example shows how to create a temporary keyboard mapping for use within a macro.

```
keyboard_push();
assign_to_key("<Alt-H>", "help");
// additional key bindings
::
process();
keyboard_pop();
```

Portability

n/a

See Also

[inq_keyboard](#), [keyboard_pop](#), [keyboard_typeables](#), [process](#)

keyboard_typeables

```
int keyboard_typeables()
```

Assign `self_insert` to all typeable keys.

Description

The `keyboard_typeables()` primitive populates the keyboard with all of the standard keys, example include *ASCII* keys (for example A-Z and 0-9), *Backspace*, *Tab*, and *Enter* are bound to `self_insert`.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[inq_keyboard](#)

load_keystroke_macro

```
int load_keystroke_macro(string def)
```

Load a recorded macro from a file.

Description

The `load_keystroke_macro()` primitive loads the specified keystroke macro *macro*, being a string in a similar format returned by [inq_keystroke_macro](#). This primitive is designed to allow user macros to load macros from external files.

Parameters

`macro` String containing the macro definition.

Returns

The `load_keystroke_macro()` primitive returns the newly associated macro identifier, otherwise -1 on error.

Portability

n/a

See Also

[inq_keystroke_macro](#)

pause

```
void pause()
```

Pause keystroke definition.

Description

The `pause()` primitive pauses a keyboard macro definition.

Usually all keyboard input typed during a `remember` sequence is saved in the keyboard macro buffer. Pressing a key assigned to `pause` causes the `remember` sequence to suspend saving the characters.

Parameters

none

Returns

nothing

Portability

n/a

See Also

remember

playback

```
int playback([int macroid])
```

Replay a keystroke macro.

Description

The *playback()* primitive replays the previously saved keyboard macro *macro*.

Parameters

macroid Optional integer macro identifier specifies the keyboard against which to derive the buffer name, if omitted the current macro identifier is utilised.

Returns

The *playback()* primitive returns greater than or equal to zero if playback was successful, otherwise less than zero on error.

Portability

n/a

See Also

remember

process

```
void process()
```

Invoke a Grief engine.

Description

The *process()* primitive invokes an instance of the Grief command-loop recursively by accepting keystrokes and calling the functions assigned thru the associated key bindings (See: [assign_to_key](#)).

process is usually invoked after building the required environment including buffers and/or windows with an associated keyboard. Process nesting may occur to any depth, within the bounds of Giefs maximum nesting level. The

current command loop executes until the session is terminated by an **exit** command.

Parameters

none

Returns

nothing

Portability

n/a

See Also

exit, suspend

push_back

```
void push_back( int key,
                [int front],
                [int x],
                [int y]      )
```

Push back a character into the keyboard.

Description

The *push_back()* primitive pushes the specified key code *key* into the keyboard input buffer.

The *front* is zero or omitted the key shall pushed to the back of the keyboard buffer, otherwise to the front.

Parameters

key Key code.

`front` Optional boolean value, which if specified and is **TRUE** (non-zero) then the character is pushed at the front of any previous characters. Otherwise if either omitted or **FALSE** (zero) then the key is pushed back after all previously pushed back characters.

`x, y` Mouse position, required when pushing back mouse events.

Returns

nothing

Portability

Mouse support is a **GriefEdit** extension.

See Also

[inq_kbd_char](#), [key_to_int](#), [read_char](#)

read_char

```
void read_char([int timeout = 0],  
              [int mode = 0]      )
```

Read next key from the keyboard.

Description

The `read_char()` primitive attempts to retrieve the next character from the keyboard.

The `timeout` argument specifies the handling of non key conditions, in other words when no keys are available. If a positive value the `timeout` argument specifies the interval in milliseconds `read_char()` should block waiting for a key to become ready. If the time limit expires before any key events `read_char` returns -1. If omitted or zero `read_char` shall block unconditionally until a key is available. A negative (-1) timeout shall not block returning immediately effectively polling the keyboard.

The `mode` parameter controls the type of keyboard event to be returned. Supported modes are as follows.

Mode	Description
0	Normal.
>0	Raw mode.
<0	Extended, returning additional keys codes including mouse.

Parameters

`timeout` Optional timeout in milliseconds, if omitted or zero `read_char` blocks until a key-code is available, otherwise -1 shall simply poll for the next key.

`mode` Optional request mode, see above.

Returns

The `read_char()` primitive returns the key-code, otherwise -1 on a timeout.

Portability

Use of a '-1' timeout value is required to emulate BRIEF.

See Also

[int_to_key](#), [inq_kbd_char](#), [inq_kbd_flags](#)

remember

```
int remember([string|int overwrite],  
            [int macroid]      )
```

Start remembering keystrokes.

Description

The `remember()` primitive control macro recording, starts remembering keystrokes, which can be later replayed with the **playback** function.

Recording is stopped by a second call to the `remember` macro.

The `remember()` primitive is not usually called as part of a macro but is usually bound to a keyboard key, for example (<F7>).

Unlike BRIEF multiple keystroke macros can be maintained. Each remember execution shall create a buffer named *KBD-MACRO-#* where # is the associated keyboard macro identifier. This buffer maybe then edited and/or saved for later use. The [inq_remember_buffer](#) primitive shall derive the buffer name.

Note:

Only the macros directly executed by the user are saved within the macro. In other words top level macros, for example the resulting dialog, shall not be reported.

The **pause** primitive temporarily stops recording the current keystroke sequence; by default assigned to (<Shift+F7>). This feature is useful if part of the keystroke macro being remembered is different each time the macro is played back. Before the variable part of the macro, press **Shift+F7**. Perform the variable part and press **Shift+F7** to resume recording.

When the macro is played back, it will pause at the point **Shift+F7** was pressed to allow the user to perform the variable portion of the sequence. Pressing **Shift+F7** then resumes playback.

Parameters

- `overwrite` Optional string containing whether or not the current macro should be overwritten. If either a case insensitive string contained "y[es]" or a non-zero the macro shall be overwritten. If omitted the user shall be prompted.
- `macroid` Optional integer macro identifier specifies the keyboard against which to store the keystrokes, if omitted the next free keyboard identifier is utilised.

Returns

The *remember()* primitive returns the positive identifier assigned to the keystroke macro on completion, 0 if the *remember* was cancelled, otherwise -1 on error or the recording began.

Portability

n/a

See Also

[pause](#)

save_keystroke_macro

```
int save_keystroke_macro(string filename)
```

Save the current keystroke macro.

Description

The *save_keystroke_macro()* primitive saves the current remembered keystrokes macro to the specified file *filename*.

Parameters

- `filename` String containing the name of the destination file to which the keystroke macro is to be saved. If no file extension is stated, the extension *.km* shall be used.

Returns

The *save_keystroke_macro()* primitive returns greater than zero on success, otherwise -1 on error.

See Also

[playback](#), [remember](#)

set_kbd_name

```
void set_kbd_name(string name,
                  [int kbdid])
```

Set the keyboard name.

Description

The *set_kbd_name()* primitive assigned the label *name* as the name of the keyboard *kbdid*, allowing primitives to assign a descriptive or meaningful definition to a keyboard. These names or descriptions may then be utilised by macros.

Parameters

- `name` String containing the name to be assigned. If empty the current name shall be cleared.
- `kbdid` Optional integer keyboard identifier, if omitted the current keyboard shall be referenced.

Returns

`nothing`

Portability

n/a

See Also[inq_kbd_name](#)**set_mouse_action**

```
int set_mouse_action(string name)
```

Set keyboard mouse handler.

Description

The *set_mouse_button()* primitive is reserved for future BRIEF compatibility.

The *set_mouse_action()* primitive sets the name of the mouse action handler within the current keyboard.

Parameters

name A string containing the name of the function to be associated with the current keyboard.

Returns

Returns the previous mouse action.

Portability

n/a

See Also[inq_mouse_action](#), [inq_keyboard](#)**set_mouse_type**

```
int set_mouse_type()
```

Sets the mouse type.

Description

The *set_mouse_type()* primitive sets the mouse type.

Parameters

type Integer stating the current mouse type.

Value	Description
0	No mouse.
1	One-button mouse.
2	Two-button mouse.
3	Three-button mouse.

button1 Option integer indicates which button shall be treated as the first button. A value of zero indicates the left-most, with a value of one indicates the right-most button. By default the left-most button is the mouse button one.

Returns

none

Portability

n/a

See Also[inq_mouse_type](#)**undo**

```
int undo([int move],
        [int pastwrite = -1],
        [int redo = FALSE] )
```

Undo previous edit operations.

Description

The *undo()* primitive undoes buffer modifications on the current buffer.

Executing without arguments undoes the last operation performed on the buffer, including cursor movement, any text modification and marked region modification. If the previous operation on the buffer was a macro, or a complex operation, e.g. global translate, then all the buffer modifications performed are undone in one step.

The *move* option limits the undo operation to the last buffer modification, restoring the cursor to the its location where the buffer was actually modified.

Each buffer maintains their own undo stack, unless disabled (See: [set_buffer_flags](#)). Under BRIEF the undo stack for a particular buffer was cleared when the buffer is written to disk, as such it was not possible to undo any operations performed on the buffer before the last *write_buffer*.

Under **GriefEdit** the undo stack is retained for the duration of the editor lifetime. If *pastwrite* is specified as positive value, the *undo* shall perform undo's beyond the last *write_buffer*; by default the user is prompted as follows

Undo past saved file mark?

Note:

Unlike BRIEF, it is possible to call *undo* from within a macro, but this use is highly dubious. It is normally called by the user directly from one of the key assignments.

Parameters

<i>move</i>	Optional boolean value, if true then all buffer operations up the last buffer modification are undone.
<i>pastwrite</i>	Option integer stating the <i>write_buffer undo</i> logic. Zero (0) disables undo's past the last write, a negative value (-1) prompts the user whether to continue otherwise and a positive value (1) permits undo's without prompt.
<i>redo</i>	Optional boolean value, if true the previous <i>undo</i> action shall be undone.

Returns

The *undo()* primitive returns the number of operations *undo*; this is **GriefEdit** extension.

Portability

n/a

See Also

[redo](#)

use_local_keyboard

```
int use_local_keyboard(int kbdid)
```

Associate a keyboard with a buffer.

Description

The *use_local_keyboard()* primitive cause the current keyboard to be associated with the current buffer.

Parameters

kbdid Keyboard identifier.

Returns

The *use_local_keyboard* primitive return 0 on success otherwise -1 on error.

Portability

n/a

See Also

[inq_local_keyboard](#), [keyboard_pop](#), [keyboard_push](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

List Primitives

Summary

List Primitives

Macros

<code>car</code>	Retrieve the first list element.
<code>cdr</code>	Retrieve all secondary list elements.
<code>command_list</code>	Retrieve list of built-in and active macros.
<code>delete_nth</code>	Remove one or more elements from a list.
<code>file_glob</code>	Return names of files that match patterns.
<code>get_nth</code>	Retrieve a list element.
<code>length_of_list</code>	Determine list element count.
<code>list</code>	Declare a list symbol.
<code>list_each</code>	Iterator though the list elements.
<code>list_extract</code>	Extract list elements.
<code>list_reset</code>	Reset the list iterator.
<code>make_list</code>	Build an evaluated list.
<code>nth</code>	Extract the indexed list item.
<code>pop</code>	Pop the last element.
<code>push</code>	Add an element onto a list.
<code>put_nth</code>	Modify a list element.
<code>quote_list</code>	Build an unevaluated list.
<code>search_list</code>	Search list contents.
<code>shift</code>	Shift the first list element.
<code>sort_list</code>	Sort list.
<code>splice</code>	Splice a list, removing and/or adding elements.
<code>strlen</code>	String length.
<code>strnlen</code>	String length limited to an explicit maximum.
<code>unshift</code>	Shift the first list element.

Macros

car

```
declare car(list lst)
```

Retrieve the first list element.

Description

The `car()` primitive retrieves a copy of the first element (also known as an atom) in the list `lst`; effectively removing all elements after the first.

`car` is non-destructive. It does not remove any elements from the source list only returning a copy of what is the first element, as such the source list is unchanged.

Note:

As lists may contain any data type, it is advised to assigned the result to a polymorphic variable, so that its type can be ascertained.

Example

TODO

Returns

The `car()` primitive returns value of the first element from the source.

Like `cdr`, the `car` primitive is built on their Lisp counter parts, which are also non-destructive; that is, neither modify or change lists to which they are applied.

Portability

n/a

See Also

`cdr`, `nth`, `put_nth`, `splice`, `list`, `pop`, `push`

cdr

```
list cdr(list lst)
```

Retrieve all secondary list elements.

Description

The `cdr()` primitive retrieves a copy of everything but the first element (also known as an atom) in the list `lst`; effectively removing the first element.

`cdr` is non-destructive. It does not remove any elements from the source list only returning a copy of what are the second and subsequent elements, as such the source list is unchanged.

Note:

The primitives `car` and `cdr` can be utilised to manipulate all elements on a list. However, it is more efficient to use `nth`, `pop`, `splice` to extract individual elements, since internally it avoids having to copy sub-lists.

Example

TODO

Returns

The `cdr()` primitive returns a list containing the second and subsequent elements from the source.

Like `car`, the `cdr` primitive is built on their Lisp counter parts, which also non-destructive; that is, neither modify or change lists to which they are applied.

Portability

n/a

See Also

`car`, `nth`, `put_nth`, `splice`, `list`, `pop`, `push`

command_list

```
list command_list( [int nomacros = FALSE],  
                   [string pattern] )
```

Retrieve list of built-in and active macros.

Description

The `command_list()` primitive returns a sorted list of all built-in primitives and the optionally all currently defined macros.

If `nomacros` is non-zero only built-in primitives are retrieved. otherwise the list includes all macros and command primitives.

The optional `pattern` is a regular expression similar to that accepted by the command line shells (See: `file_match`), providing an inclusion filter.

Parameters

`nomacros` Optional boolean value, if stated as true then only built-ins primitives are retrieved, filtering any run-time loaded macros.

`pattern` Optional string value containing the macro-name pattern to be applied against each name, only returning the matching. A macro-name expression is a shell style regular expression (See: `file_match`) (e.g. * for wildcard, ? for wild-character, and [...] to select a range of characters).

Returns

The `command_list()` primitive returns a list of strings each containing the name of a primitive or macro.

Portability

n/a

See Also

`macro_list`, `file_match`

delete_nth

```
void delete_nth( list lst,
                 [int offset = 0],
                 [int length = 1] )
```

Remove one or more elements from a list.

Description

The `delete_nth()` primitive deletes `length` elements from the specified list `lst` starting at the index `offset`.

Parameters

- `lst` List reference which shall have elements removed.
- `offset` Optional integer offset within list, if omitted defaults to the first element being offset 0.
- `length` Optional integer length, states the number of elements to be removed; if omitted only one element is removed. If 0 or less, than no elements will be removed.

Returns

`nothing`

Portability

n/a

See Also

[get_nth](#), [nth](#), [splice](#)

file_glob

```
list file_glob(string files)
```

Return names of files that match patterns.

Description

The `file_glob()` primitive performs file name *globbing* in a fashion similar to the *csh* shell. It returns a list of the files whose names match any of the pattern arguments.

No particular order is guaranteed in the list, so if a sorted list is required the caller should use [sort_list](#).

The pattern arguments may contain any of the following special characters:

- `~[user/]` Home directory of either the current or the specified user.
- `?` Matches any single character.
- `*` Matches any sequence of zero or more characters.
- `[ch]` Matches any single character in chars. If ch's contains a sequence of the form *a-b* then any character between *a* and *b* (inclusive) will match.
- `\x` Matches the character *x*.

Returns

The `file_glob()` primitive returns a list of strings corresponding to the filenames which match the wild card expression in string. When no matches or error, an empty list is returned.

See Also

[glob](#), [file_canon](#), [file_pattern](#), [find_file](#), [expandpath](#)

get_nth

```
declare get_nth(int idx,
               list expr)
```

Retrieve a list element.

Description

The `get_nth()` primitive retrieves the element positioned at offset `idx` within the specified list expression `expr`.

Note:

This primitive is generally not utilised directly as the GRIEF language supports list offset operator of the form `[idx]`.

Returns

The `get_nth()` primitive returns the value of the referenced element, otherwise NULL if the index is out of bounds.

Portability

n/a

See Also

`car`, `cdr`, `nth`, `put_nth`, `splice`, `list`, `pop`, `push`

length_of_list

```
int length_of_list(list lst)
```

Determine list element count.

Description

The `length_of_list()` primitive retrieves the number of top-level elements (atoms) within the list `lst`; with sub-lists being counted as single elements.

Returns

The `length_of_list()` primitive returns the top-level element count.

Portability

n/a

See Also

`make_list`, `quote_list`, `list_each`, `list_extract`

list

```
list sym1, sym2 ...;
```

Declare a list symbol.

Description

The `list` statement declares a list being a container of atoms. More precisely, a list is either an empty container NULL, or a sequential list of values.

A list can store any other data type, including lists, allowing the creation of complex types. A list may generally only be extended at its end, by appending data; elements can be updated using `put_nth` and replaced using `splice`.

Any element may be referenced by specifying its ordinal position in the list using `<get_th>` and array subscripts. Lists are not the most efficient data types, since subscript element access involves iterating from the head until the associated value is referenced, whereas `list_each` primitive allows effective iteration.

Returns

nothing

Portability

n/a

See Also

`Types`, `int`, `string`, `float`, `double`, `array`

list_each

```
int list_each( list      lst,
               declare &value,
               [int increment = 1])
```

Iterator though the list elements.

Description

The `list_each()` primitive iterates over a normal list value and sets the variable `value` to be each element of the list in turn.

Example

Split the flag list specification `spec` into its components and iterator the results.

```
const list flags = split(spec, ",");
string flag;

while (list_each(flags, flag) >= 0) {
    process_flag(flag);
}
```

Warning:

If any part of the list is modified, foreach may become very confused. Adding or removing elements within the loop body, for example with splice shall result in unpredictable results possibility crashing the editor.

Secondary list iteration continues until the last element within the list is consumed; if the loop is terminated prior to the end condition, list_reset should be used to reset the list cursor for subsequent iterations.

Parameters

`lst` List expression.
`value` Variable populated with the element value.
`increment` Optional integer iterator loop increment, if omitted defaults to one element.

Returns

The `list_each()` primitive returns the element index retrieved, otherwise -1 upon an end-of-list condition or error.

Portability

A **GriefEdit** extension.

See Also

`make_list`, `quote_list`, `length_of_list`, `list_reset`, `list_extract`

list_extract

```
list list_extract( list lst,
                  int start,
                  [int end],
                  [int increment])
```

Extract list elements.

Description

The `list_extract()` primitive iterates over the list values generating a new list of the referenced elements within the list.

Parameters

`lst` List expression.
`start` Optional integer index, if specified states the first element index at which the iterations begins.
When omitted the iterations begins from the start of the list.
`end` Optional integer index, if specified states the last element index at which the iterations terminates.
When omitted the iterations completes at the end of the list.
`increment` Optional integer iterator loop increment, if omitted defaults to one element.

Returns

The `list_extract()` primitive returns a list containing the referenced elements, otherwise a NULL list on error.

Portability

n/a

See Also

`make_list`, `quote_list`, `length_of_list`, `list_each`

list_reset

```
int list_reset(list lst)
```

Reset the list iterator.

Description

The `list_reset()` primitive resets the list iteration cursor.

Parameters

`lst` List expression.

Returns

The `list_reset()` primitive returns the selected element index.

Portability

A **GriefEdit** extension.

See Also

[make_list](#), [quote_list](#), [length_of_list](#), [list_each](#), [list_extract](#)

make_list

```
list make_list(...)
```

Build an evaluated list.

Description

The `make_list()` primitive builds a new list constructed from the specified arguments. Each of the specified arguments shall be evaluated and appended the end of the list. This is in contrast to the [quote_list](#) primitive which does not evaluate any of the arguments.

Parameters

... One or more values to be made into list elements.

Returns

The `make_list()` primitive returns the built list.

Portability

n/a

See Also

[quote_list](#), [length_of_list](#), [list_each](#), [list_extract](#)

nth

```
declare nth(list expr,
           int idx,
           [int idx2 ...])
```

Extract the indexed list item.

Description

The `nth()` primitive retrieves the value of the referenced list element at the index `idx` from the list expression `expr`. The first element of a list is element 0; the `nth` primitive allows lists to be treated like arrays. The last element of a list is [length_of_list](#) minus one.

The `nth()` primitive is an efficient way of accessing random elements within a list, than for example using the lists primitive `car` and `cdr`. In addition `nth` can access multidimensional lists as a single operation.

Note:

This primitive is the underlying implementation of the list offset operator `[]`, which are converted by the GRIEF Macro Compiler.

Furthermore, this interface should be considered as an **internal** primitive. As this primitive is not compatible with the `nth` macro of CRISP™ its direct usage is not recommended, instead rely on the offset operator `[]`.

Parameters

`expr` List expression.

`idx` Integer index.

... None or more integer indexs of sub-elements.

Returns

The *nth()* primitive returns the value of the specified element from the referenced list, otherwise NULL on error.

Portability

GRIEF uses an alternative prototype to support multidimensional lists, unlike CRISP™ which only supports a single dimension.

```
nth(expr, list_expr)
```

As such for portability *nth* use should be restricted.

See Also

[car](#), [cdr](#), [put_nth](#), [splice](#), [list](#), [pop](#), [push](#)

pop

```
declare pop(list expr)
```

Pop the last element.

Description

The *pop()* primitive removes and returns the last value of the list, shortening the list by one element.

Returns

The *pop()* primitive returns the element removed, otherwise NULL on error.

Example

Iterator the list from the end, removing and then printing each element in the process;

```
list l = {"one", "two", "three", "four"};
while (length_of_list(l)) {
    message("%s", pop(l));
}
```

the resulting output

```
four
three
two
one
```

Portability

A **GriefEdit** extension

See Also

[push](#), [shift](#), [splice](#), [car](#), [cdr](#)

push

```
declare push(      list lst,
                  declare value ... )
```

Add an element onto a list.

Description

The *push()* primitive adds one or more elements *value* to the end of the list.

Returns

The *push()* primitive returns a copy of the value added.

Portability

A **GriefEdit** extension.

See Also

[pop](#), [shift](#), [splice](#), [car](#), [cdr](#)

put_nth

```
declare put_nth(symbol,
    ... )
```

Modify a list element.

Description

The *put_nth()* primitive modifies the element within the specified list.

Note:

This primitive is generally not utilised directly as the GRIEF language supports a list offset operator of the form *[idx]*.

Returns

The *put_nth()* primitive returns the value of the referenced element, otherwise NULL if the index is out of bounds.

Portability

Standard CRISP™ implementation utilises an alternative prototype which does not support multidimensional lists.

```
put_nth(expr, list_expr, value)
```

As such for portability *put_nth* use should be restricted.

See Also

[car](#), [cdr](#), [nth](#), [get_nth](#), [splice](#), [list](#), [pop](#), [push](#)

quote_list

```
list quote_list(...)
```

Build an unevaluated list.

Description

The *quote_list()* primitive builds a list of the specified arguments unchanged and unevaluated. This is in contrast to the [make_list](#) primitive which shall evaluate which of the arguments.

quote_list is the one of the mechanisms for creating list literal constants, also see [make_list](#). It is normally used for assigning initial values to lists, or for passing an anonymous macro to another macro.

Example

An example usage results in the list contains two strings followed by two integers.

```
list l;
l = quote_list("first", "second", 1, 2);
```

As an alternative the following syntax implies the use of *quote_list* by the GRIEF Macro compiler.

```
list l = {"first", "second", 1, 2};
```

Parameters

... One or more values to be made into list elements.

Returns

The *quote_list()* primitive returns the built list.

Portability

n/a

See Also[make_list](#), [length_of_list](#), [list_each](#), [list_extract](#)**search_list**

```
int search_list(      [int start],
                     string pattern,
                     list   expr,
                     [int re],
                     [int case] )
```

Search list contents.

Description

The `search_list()` primitive searches the list `expr` against the expression `pattern`. The treatment of the matched pattern may either be a regular-expression or constant string dependent on `re` and case folding based on `case`.

Note:

The `search_string` primitive is provided primary for Crisp compatibility. Since `re` and `case` can reference the current global search states, inconsistent and/or unexpected results may result between usage; as such it is advised that the state-less `re_search` primitive is used by new macros.

Parameters

<code>start</code>	Optional integer index, states the element offset at which search operation shall start. If omitted, the search is started from the beginning of the list.
<code>pattern</code>	A string containing the pattern to be matched.
<code>expr</code>	A list expression containing the list to be search for the specified <code>pattern</code> .
<code>re</code>	Optional integer value. If <code>re</code> is specified and is zero, then <code>pattern</code> is treated as a literal string not as a regular expression; behaviour being the same as index . Otherwise the string shall be treated as a regular expression using the current global syntax, see re_syntax and search_back for additional information.
<code>case</code>	Optional integer value specifying whether case folding should occur. If <code>case</code> is specified and is zero, then the search is done with case insensitive. If omitted the current global setting is referenced.

Returns

The `search_list()` primitive returns the index of the matched element, otherwise -1 if no match.

Portability

n/a

See Also[list](#), [re_search](#)**shift**

```
declare shift(list lst)
```

Shift the first list element.

Description

The `shift()` primitive removes (shifts) the first value of the list `lst` off and returns its, shortening the list by 1 and moving everything down.

If there are no elements in the list, the following is echoed onthe command promot and `shift` returns the null value.

```
shift: empty list.
```

Parameters

`lst` List expression.

Returns

The `shift()` primitive returns the first element on the list.

Examples

Iterator the list from the front, removing and then printing each element in the process;

```
list l = {"one", "two", "three", "four"};
while (length_of_list(l)) {
    message("%s", shift(l));
}
```

the resulting output

```
one
two
three
four
```

Portability

A **GriefEdit** extension

See Also

[pop](#), [push](#), [splice](#), [car](#), [cdr](#)

sort_list

```
list sort_list( list lst,
                [string|int comparator = 0],
                [int type = 3] )
```

Sort list.

Description

The *sort_list()* primitive sorts the list and returns the sorted array value.

By default elements are sorted alphabetically yet the sort can be modified using the user specified macro or using one of the predefined system sort macros.

Sort Method

GriefEdit 2.5.4 and earlier bindly utilised the system supplied quicksort algorithm to implement sort. The characteristics of the algorithm could not be defined as such was normally unstable, plus may have gone quadratic. (Although quicksort's run time is $O(N\log N)$ when averaged over all arrays of length N , the time can be $O(N^{**2})$, quadratic behavior, for some inputs).

Following Perl and Java in 2.5.5 the default sort implementation was replaced with a stable mergesort algorithm whose worst-case behavior is $O(N\log N)$. In addition quicksort defends against quadratic behaviour by shuffling large arrays before sorting.

A stable sort means that for records that compare equal, the original input ordering is preserved. Mergesort is stable, quicksort is not. Stability will matter only if elements that compare equal can be distinguished in some other way. That means that simple numerical and lexical sorts do not profit from stability, since equal elements are indistinguishable. However, with a comparison such as

```
int
mysort(string a, string b)
{
    return (substr(a, 0, 3) <= substr(b, 0, 3));
```

stability might matter because elements that compare equal on the first 3 characters may be distinguished based on subsequent characters. In Perl 5.8 and later, quicksort can be stabilized, but doing so will add overhead, so it should only be done if it matters.

The best algorithm depends on many things. On average, mergesort does fewer comparisons than quicksort, so it may be better when complicated comparison routines are used. Mergesort also takes advantage of pre-existing order, so it would be favored for using *sort()* to merge several sorted arrays. On the other hand, quicksort is often faster for small arrays, and on arrays of a few distinct values, repeated many times. You can force the choice of algorithm with the specification of the third argument. The default algorithm is mergesort, which will be stable even if you do not explicitly demand it. The stability of the default sort is a side-effect that could change in later versions.

```
int
mycmp(string a, string b)
{
    return (a <= b);
}
```

Note:

In the interests of efficiency the normal calling code for subroutines is bypassed, with the following effects: elements to be compared are passed into *a* and *b*, are passed by reference as such do not modify *a* and *b*.

Parameters

list List to be sorted.
comparator Optional string comparison macro or integer direction. A string value states the comparison macro to be executed. Whereas an integer value states the direction, with zero selecting the built "sort_list::backward" comparator and a non-zero value selecting "sort_list::forward". If omitted the comparator defaults to forward.
type Optional integer stating the sort type, being
 1 quicksort.
 2 mergesort
 3 heapsort (default).

Returns

Sorted list.

Portability

Second argument allowing either a sort-order or user specified callback plus type selection are **GriefEdit** extensions.

See Also

[sort_buffer](#)

splice

```
int splice( list lst,
            int offset          = 0,
            [int length],
            [declare value]     )
```

Splice a list, removing and/or adding elements.

Description

The *splice()* primitive removes the elements designated by *offset* and *length* from an *list*, and replaces them with the elements of *value*, if any.

If *offset* is negative then it starts that far from the end of the list.

If *length* is negative, removes the elements from *offset* onward except for *-length* elements at the end of the list.

If *length* is omitted, removes everything from *offset* onward. If both *offset* and *length* are omitted, removes everything.

If *offset* is past the end of the list, warning is issued and splices at the end of the list.

Parameters

lst List expression.
offset Non-negative list offset from the front at which element are to be removed, when negative then it starts from the end. If omitted removes from the head of the list.
length Optional number of elements to be remove.
value Optional value to be inserted at the deletion point.

Examples**Equivalent**

The *splice* primitive provides a general purpose list manipulation tool, which can be as a alternative to a number of specialise primitives.

The following macro primitives `push`, `<+=>`, `pop`, `cdr`, `unshift` use on the left are equivalent to right `splice` usage.

```
push(lst, x)           splice(lst, length_of_list(a), 0, x)
or lst += x            splice(lst, -1)
pop(lst)               splice(lst, 0, 1)
cdr(lst)               splice(lst, 0, 0, x)
unshift(lst, x)        splice(lst, i, 1, y)
lst[i] = y
```

Flatten mode

Splice usage implies the non-flattening of a source list when appending into another, whereas list append operations shall flatten the source list at level 0, as such when used in the following context the result is;

```
list lst = make_list("a", "b");
lst += lst;
// result = "a", "b", "a", "b"
```

The alternative `splice` usage which shall place a list reference within the list.

```
splice(lst, length_of_list(lst), 0, lst);
```

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

`pop`, `push`, `shift`, `car`, `cdr`

strlen

```
int strlen(string|list arg,
          [int step = 1])
```

String length.

Description

The `strlen()` primitive computes the length of `arg`.

For string arguments, computes the length of the string in character.

In the case `arg` is a list, computes the length of the longest string element contained within the list. Non-string elements shall be omitted. If specified and positive, iteration through the list shall only inspect each `step` element, starting with the first element within the list.

Parameters

`arg` String or list.

`step` Optional integer, stating the list iterator step value 1 or greater.

Returns

The `strlen()` primitive returns the number of characters contained within the string.

See Also

`strnlen`, `length_of_list`

strnlen

```
int strnlen(string|list arg,
            int      maxlen,
            [int step = 1])
```

String length limited to an explicit maximum.

Description

The `strnlen()` primitive computes the length of `arg` limited to `maxlen`.

For string arguments, computes the length of the string in character.

In the case `arg` is a list, computes the length of the longest string element contained within the list in characters. Non-string elements shall be omitted. If specified and positive, iteration through the list shall only inspect each `step` element, starting with the first element within the list.

Parameters

- `arg` String or list.
- `maxlen` Upper limit to be applied to the length.
- `step` Optional integer, stating the iterator step value.

Returns

The `strnlen()` primitive returns either the same result as `strlen()` or `maxlen`, whichever is smaller.

Portability

A **GriefEdit** extension.

See Also

[strlen](#)

unshift

```
declare unshift(list lst,
               declare value)
```

Shift the first list element.

Description

The `unshift()` primitive is reserved for future use.

The `unshift()` primitive performs the opposite action to that of a `shift`. It prepends `value` to the front of the list `lst` returns the resulting number of elements within the list.

Parameters

- `lst` List to be modified.
- `value` Value to be prepended.

Returns

The `unshift()` primitive returns the resulting number of list elements.

Portability

A **GriefEdit** extension.

See Also

[pop](#), [push](#), [shift](#), [splice](#), [car](#), [cdr](#)

`$Id:` \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Macro Language Primitives

Summary

Macro Language Primitives

Constants

BPACKAGES	BRIEF packages default.
GRBACKUP	Backup path.
GRDICTIONARIES	Dictionary locales.
GRDICTIONARY	Dictionary search path.
GRFILE	Default empty file name.
GRFLAGS	Default command line arguments.
GRHELP	Help search path.
GRINIT_FILE	GRIEF initialisation name.
GRKBDPATH	Keyboard library search path.
GRLEVEL	GRIEF Nesting level.
GRLOG_FILE	GRIEF diagnostics log file name.
GRPATH	Macro object search path.
GRPROFILE	Profile directory override.
GRRESTORE_FILE	GRIEF restore file name.
GRSTATE_DB	GRIEF state database name.
GRSTATE_FILE	GRIEF state file name.
GRTEMPLATE	Source template search path.
GRTERM	Terminal override.
GRTERMCAP	Terminal capability database.
GRTMP	Temporary dictionary.
GRVERSIONMAJOR	GRIEF major version.
GRVERSIONMINOR	GRIEF minor version.
GRVERSIONS	Backup versions.
errno	Last system errno number.

Macros

__breaksw	Switch break statement.
autoload	Register location of one or more macros.
bless	Associate an object with a class/module.
break	break statement.
call_registered_macro	Invoke registered macro callbacks.
case	Switch case statement.
catch	Catch statement.
continue	Loop continuation.
create_dictionary	Create a dictionary.
delete_dictionary	Destroy a dictionary.
delete_macro	Delete a macro from memory.
dict_clear	Clear a dictionary.
dict_delete	Remove a dictionary item.
dict_each	Iterator a dictionary.
dict_exists	Dictionary item existence check.
dict_keys	Iterator dictionary keys.
dict_list	Retrieve dictionary items.
dict_name	Retrieve a dictionary name.
dict_values	Iterator dictionary values.
do	do statement.
else	else statement.
execute_macro	Invokes a command by name.
exit	Exit current process level.
finally	Finally statement.
first_time	Determine a macros initialisation status.
for	for statement.
foreach	Container iterator.
fstype	File system type.
get_property	Retrieve a dictionary item.
if	if statement.
inq_btn2_action	Retrieve the second button action.
inq_called	Get the name of the calling macro.
inq_macro	Determine whether a macro or primitive exists.
inq_macro_history	Retrieve macro execution history.
inq_module	Retrieve the current module.
inq_msg_level	Get the message level.
list_of_dictionaries	List of created dictionaries.
load_macro	Load a macro object.
macro	Define a macro body.

module	Assign a module identifier.
nothing	Noop.
register_macro	Register a callback procedure.
reload_buffer	Reload the buffer content.
replacement	Overload an existing macro definition.
require	Enforce the use of an external module.
reregister_macro	Register a unique callback procedure.
restore_position	Restore a previously saved position.
return	Return from a macro.
returns	Return an expression from a macro.
save_position	Saves current cursor/buffer state.
set_btn2_action	Set the second button action.
set_calling_name	Set the name of the calling macro.
set_macro_history	Set the macro execution history.
set_property	Set a dictionary item.
switch	Switch statement.
throw	Throw an exception.
try	Try statement.
unregister_macro	Remove a registered macro.
while	while statement.

Constants

BPACKAGES

```
extern const string BPACKAGES;
```

BRIEF packages default.

Description

The *BPACKAGES* global string is used to specify the BRIEF language sensitive editing modes. This is a hangover from the PC version of BRIEF, it is used to configure old-style package support; see the macro source *language.cr*.

BPACKAGES shall be referenced on the first running of GRIEF and then exported into the GRIEF runtime configuration file. The initial value of *BPACKAGES* is either imported from the environment or if not available then the following default is used.

```
.c.cc.CC.cpp.h.H.hpp-c:hilite,t;.default:hilite,template,regular;
```

Package Specification

You can attach these packages so that they are automatically used with program source files that have special file extensions. The packages configuration is used by all file extension dependent packages.

The format of the *package* specification is,

```
package: extension[,extension]...[-equivalent]:\
          package [args],package...;extension...
```

extension

The **extension** element is the file extension that will invoke a specific style. One or more comma separated extension can be specified for each style.

Note: the special extension "default" is used to specify all extensions not specifically included in the packages string.

equivalent

The **equivalent** element is an optional value which specifies that the preceding extensions should be treated the same as the "equivalent extension" by the language sensitive features, an extension override.

All extensions preceding the equivalent extension back to preceding semicolon, another equivalent extension, or the beginning of the packages definition are affected by the command. Hence given:

```
package: .asm:.r;.cpp.hpp.h-c:smart
```

GRIEF shall consider extensions *.cpp*, *.hpp* and *.h* equivalent to the extension *.c*, and would use *.c* the smart editing

package for all three. Note also that since `.c` has not been explicitly specified as an extension, no packages are assigned to it. The following is required to complete the recipe:

```
package: .asm:r;.cpp.hpp.h.c-c:c:smart
```

GRIEF shall first check for the package using the actual extension before for one with the equivalent extension. If in the previous example, *smart* indentation was available for `.cpp` files, GRIEF would use the `.cpp` support over the `.c` support.

This feature is useful when the extension is not automatically recognised and is unable to provide *smart* indenting. Using extension equivalence, informs GRIEF that an unsupported extension should be treated like an equivalent supported one.

```
package: .default:hilite;.c:hilite,t 1 1 0 1
```

The extension `.default` is used as a wild card (or default), to any match an extension not explicitly listed. The above example enables the *hilite* package on implicitly supported file types, plus an explicit configuration for the `.c` extension.

package

Is the macro package attached to the extension. Each extension can have multiple packages associated with it, as long as the package don't conflict.

See Also

[GRTEMPLATE](#)

GRBACKUP

```
extern const string GRBACKUP;
```

Backup path.

Description

The *GRBACKUP* global string is used to specify the default backup directory, when buffer backups are enabled. In the absence of any buffer or global [set_backup_option](#) **BK_DIR** configuration nor the *GRBACKUP* environment variable, GRIEF renames the original file with a `.bak` extension.

Compatible

GRBACKUP is equivalent to the BRIEF *BBACKUP* environment variable.

See Also

[GRVERSIONS](#)

GRDICTIONARIES

```
extern const string GRDICTIONARIES;
```

Dictionary locales.

Description

The *GRDICTIONARIES* global string is used to specify the dictionary locales to be applied during spell check operations, stated as a list of comma separated dictionary names.

Example

```
GRDICTIONARIES=en_GB,en_US
```

The default value assigned when no value is available.

```
GRDICTIONARIES=en_US,en_GB,default
```

The value of *GRDICTIONARIES* can be dumped by running GRIEF with the `--config` command line option.

```
gr --config
```

See Also

[GRPATH](#), [GRDICTIONARY](#)

GRDICTIONARY

```
extern const string GRDICTIONARY;
```

Dictionary search path.

Description

The *GRDICTIONARY* global specifies the directory from which dictionaries are to be sourced during spell check operations.

See Also

[GRPATH](#), [GRDICTIONARIES](#)

GRFILE

```
extern const string GRFILE;
```

Default empty file name.

Description

The *GRFILE* global string is used to specify the file name used on start up as the default buffer name to be created, when no explicit files are stated on the command line.

The initial value of *GRFILE* is either imported from the environment or if not available then a value of *newfile* is used.

```
GRFILE=newfile
```

See Also

[GRFLAGS](#)

GRFLAGS

```
extern const string GRFLAGS;
```

Default command line arguments.

Description

The *GRFLAGS* global string is used to specify the default implied command line arguments. GRIEF processes the switches stated in *GRFLAGS* after processing the explicit flags on the command line.

The *GRFLAGS* variable provides a means to set up private user options to be automatically applied on each execution. For example, your GRIEF profile macro can be automatic loaded on start using:

```
GRFLAGS="-mmyprofile"
```

The initial value of *GRFLAGS* is either imported from the environment or if not available then a system dependent default is used.

Compatible

GRFLAGS is equivalent to the BRIEF *BFLAGS* environment variable.

See Also

[GRFILE](#)

GRHELP

```
extern const string GRHELP;
```

Help search path.

Description

The *GRHELP* global string is used to specify the directory name stating the search path which GRIEF shall utilise to locate the help database. The initial value of *GRHELP* is either imported from the environment or if not available is derived from the location of the running application.

Directly setting this value within the session environment permits an alternative GRIEF installation.

Example

Default Unix definition.

```
/usr/local/grief/help
```

The value of *GRHELP* can be dumped by running GRIEF with the *--config* command line option.

```
gr --config
```

Compatible

GRHELP is equivalent to the BRIEF *BHELP* environment variable.

See Also

[GRPATH](#), [GRBACKUP](#), [GRDICTIONARIES](#)

GRINIT_FILE

```
extern const string GRINIT_FILE;
```

GRIEF initialisation name.

Description

The *GRINIT_FILE* global constant is assigned to the system dependent initialisation file name.

The *GRINIT_FILE* file contains optional runtime configuration settings to initialise GRIEF when it starts.

- On Unix systems, “.grinit”.
- On Windows systems, “_grinit”.

Location

The standard location of the configuration is within the home directory.

- On Unix style systems, the home directory of the current user is specified by the \$HOME environment variable; this is also represented by the ~ directory.

```
~/.grinit
```

- On Windows systems, the home directory is normally derived from the \$HOMEDRIVE and \$HOMEPATH environment variables or system calls.

```
$HOMEDRIVE\$HOMEPATH\_grinit
```

Within all environments the [GRPROFILE](#) override can be used to state an alternative location.

The *sysinfo* macro can be used to see the current home and/or profile specifications.

See Also

[GRRESTORE_FILE](#), [GRSTATE_FILE](#), [GRSTATE_DB](#)

GRKBDPATH

```
extern const string GRKBDPATH;
```

Keyboard library search path.

Description

The *GRKBDPATH* global string is used to specify the directory name stating the location shall utilise to keyboard library definitions. The initial value of GRKBDPATH is either imported from the environment or if not available is derived from *GRTEMPLATE*.

These macros build the global list variable, *kbd_labels*, that is then is used by the [int_to_key](#) to derive the displayable label.

The keyboard library macro is only relevant within several help functions, so that meaningful keyboard labels can be given. For example, you might have a META key rather than an ALT key.

```
list
kbd_labels = {
    AF(1),      "<Meta-F1>",
    AF(2),      "<Meta-F2>",
    AF(3),      "<Meta-F3>",
    AF(4),      "<Meta-F4>",
    AF(5),      "<Meta-F5>",
    AF(6),      "<Meta-F6>",
    AF(7),      "<Meta-F7>",
    AF(8),      "<Meta-F8>",
    AF(9),      "<Meta-F9>",
    AF(10),     "<Meta-F10>",
    AF(11),     "<Meta-F11>",
    AF(12),     "<Meta-F12>"
};
```

See Also

[GRTEMPLATE](#)

GRLEVEL

```
extern const int GRLEVEL;
```

GRIEF Nesting level.

Description

The *GRLEVEL* global constant is set by GRIEF to the current nesting level, stating the number of current invocations of GRIEF within the same session. This value allows the user to determine that there are nested invocations of GRIEF running; see [inq_brief_level](#).

Compatible

GRLEVEL is equivalent to the BRIEF *BLEVEL* environment variable.

It is a hangover from BRIEF which displays the level number in the bottom right corner. Unlike BRIEF the value only represents the number of instances running within the current terminal session; with each terminal maintaining an independent nesting level.

See Also

[GRPATH](#), [GRHELP](#)

GRLOG_FILE

```
extern const string GRLOG_FILE;
```

GRIEF diagnostics log file name.

Description

The *GRLOG_FILE* global constant is assigned to the system dependent default file name:

- On Unix systems, “.grief.log”.
- On Windows systems, “grief.log”.

GRPATH

```
extern const string GRPATH;
```

Macro object search path.

Description

The *GRPATH* global string is used to specify one or more directory names stating the search path which GRIEF shall utilise to locate macro objects during **autoload** and **require** operations. The initial value of *GRPATH* is either imported from the environment or if not available is derived from the location of the running application.

Same as the system *PATH* environment variable, the directories contained within *GRPATH* should use the operating system dependent delimiter generally either a semi-colon (;) for DOS/Windows or a colon (:) for Unix style systems.

Directly setting this value within the session environment permits an alternative GRIEF installation.

Example

Possible Unix definition, allowing access to both user localised and global macros.

```
$HOME/grief:/usr/local/grief/macro
```

The value of *GRPATH* can be dumped by running GRIEF with the **--config** command line option.

```
gr --config
```

Compatible

GRPATH is equivalent to the BRIEF *BPATH* environment variable.

See Also

[GRHELP](#), [GRBACKUP](#), [GRDICTIONARIES](#)

GRPROFILE

```
extern const string GRPROFILE;
```

Profile directory override.

Description

The *GRPROFILE* global string is used to specify an override to the standard users home directory; it is utilised by several macros to source runtime configuration details. *GRPROFILE* provides the user a means of stating an alternative location.

See Also

[GRPATH](#)

GRRESTORE_FILE

```
extern const string GRRESTORE_FILE;
```

GRIEF restore file name.

Description

The *GRRESTORE_FILE* global constant is assigned to the system dependent default file name, for example *.grief*.

See Also

[GRSTATE_FILE](#), [GRSTATE_DB](#), [GRINIT_FILE](#)

GRSTATE_DB

```
extern const string GRSTATE_DB;
```

GRIEF state database name.

Description

The *GRSTATE_DB* global constant is assigned to the system dependent default file name, for example *grstatedb*.

See Also

[GRRESTORE_FILE](#), [GRSTATE_FILE](#), [GRINIT_FILE](#)

GRSTATE_FILE

```
extern const string GRSTATE_FILE;
```

GRIEF state file name.

Description

The *GRSTATE_FILE* global constant is assigned to the system dependent default file name, for example *.grstate*.

See Also

[GRRESTORE_FILE](#), [GRSTATE_DB](#), [GRINIT_FILE](#)

GRTEMPLATE

```
extern const string GRTEMPLATE;
```

Source template search path.

Description

The *GRTEMPLATE* global string is used to specify the language sensitive file and function header generator. The initial value of *GRTEMPLATE* is either imported from the environment or if not available is derived from the location of the running application.

Template syntax examples; see the macro source macro *funchead.cr* for details.

Synopsis:

```
%[1%FTYPE% %FNAME%%FOPEN% %FARGS% %FCLOSE%      // single arg]%
%[2%FTYPE% %FNAME%%FOPEN% %FARGS(13)% %FCLOSE% // multi arg]%
%FTYPE% %FNAME%
%FOPEN%
%FARGS(,72)%
%FCLOSE%
```

Purpose:

```
%FDESC%
```

Parameters:

```
FINPUTS(9)
%FINPUTS(9)%
FINPUTS(0,+10)
%FINPUTS(0,+10)%
FINPUTS(0,+10,-2)
%FINPUTS(0,+10,-2)%
FINPUTS(0,32,,80,1,,1)
%FINPUTS(0,32,,80,1,,1)%
FINPUTS(0,32,-4,60,1,1)
%FINPUTS(0,32,-4,60,1,1)%
FINPUTS(0,-1)
%FINPUTS(0,-1)%
FINPUTS(0,20,,70,1,1,1)
%FINPUTS(0,20,,70,1,1,1)%
```

Returns:

```
%FTYPE% -
```

See Also

[BPACKAGES](#)

GRTERM

```
extern const string GRTERM;
```

Terminal override.

Description

The *GRTERM* global string is used to specify the identifier for the current terminal type as defined by the UNIX terminfo database, similar to the system *TERM* environment variable. *GRTERM* should be considered as an override to the standard system *TERM* specification; it may be used in preference to *TERM* allowing a GRIEF specialisation without affecting other console applications.

The leading component of the *GRTERM* specification consists of the terminal base-name. Following the base-name, you may add any reasonable number of hyphen-separated feature suffixes, for example:

```
xterm-256color
```

Some of the standard features include,

Feature	Description
m, mono	Mono terminal.
256, 256color	Terminal supports 256 colours.
16, 16color	Terminal supports 16 colours.
88	Terminal support 88 colours.

During initialisation GRIEF firstly references *GRTERM* and then secondary the system *TERM* environment variable values. The derived terminal base-name causes the associated *tty* macro to be loaded; this macro has the responsibility for configuring GRIEF's keyboard and screen layout using *set_term_feature* and *set_term_keyboard* plus related primitives.

See Also

[GRTERMCAP](#)

GRTERMCAP

```
extern const string GRTERMCAP;
```

Terminal capability database.

Description

The *GRTERMCAP* global string is used to specify the path to the terminal capability database, similar to the system *TERMCAP* environment variable. *GRTERMCAP* should be considered as an override to the standard system *TERM* specification; it may be used in preference to *TERMCAP* allowing a GRIEF specialisation without affecting other console applications.

See Also

[GRTERM](#)

GRTMP

```
extern const string GRTMP;
```

Temporary dictionary.

Description

The *GRTMP* global string is used to specify the directory within which temporary files are to be created. *GRTMP* should be considered as an override to the standard system temporary directory specification.

GRIEF firstly references *GRTMP*; if this is not defined it tries *BTMP* secondary and then system dependent alternatives, for example *TEMP* and *TEMPDIR*.

If no temporary is not defined, or if it is set to the name of a directory that does not exist, temporary files shall be created within a system dependent default.

Compatible

GRTMP is equivalent to the BRIEF *BTMP* environment variable.

See Also

[GRPATH](#)

GRVERSIONMAJOR

```
extern const int GRVERSIONMAJOR;
```

GRIEF major version.

Description

The *GRVERSIONMAJOR* global constant is set to the minor number of the running GRIEF implementation version.

See Also

[GRVERSIONMINOR](#)

GRVERSIONMINOR

```
extern const int GRVERSIONMINOR;
```

GRIEF minor version.

Description

The *GRVERSIONMINOR* global constant is set to the minor number of the running GRIEF implementation version.

See Also

[GRVERSIONMAJOR](#)

GRVERSIONS

```
extern const int GRVERSIONS;
```

Backup versions.

Description

The *GRVERSIONS* global string is used to specify the number of backup versions to be maintained for each modified file, when buffer backups are enabled. When stated GRIEF shall keep multiple backup copies of each edited file. In the absence of any buffer or global [set_backup_option](#) **BK_VERSIONS** configuration nor the *GRVERSIONS* environment variable, GRIEF simple keeps the previous image.

When enabled, GRIEF creates a set of *n* directories given by *GRVERSION* named 1 through to *n*; the directory name being number of previous edit sessions the files contained within represent.

For example, given **GRBACKUP=~/backups** and **GRVERSIONS=3** the following directory structure shall be maintained.

```
~/backups
~/backups/1
~/backups/2
~/backups/3
```

During backup operations, starting from *n-1* in reverse order the previous images are rolled into the higher sequenced directory; with any images within the last directory being removed. Finally the most recent is saved into the root of the tree. On completion the oldest version of a file can be found in the *nth* directory; the next oldest is in *n - 1*. The most recent backup is the file in the backup directory.

Given the edited file *myfile.txt* against the above configuration the following backup images may exist:

```
~/backups/myfile.txt      [latest]
~/backups/1/myfile.txt
~/backups/2/myfile.txt
~/backups/3/myfile.txt      [oldest]
```

See Also

[GRBACKUP](#)

errno

```
extern int errno;
```

Last system errno number.

Description

When a system call detects an error, it returns an integer value indicating failure (usually -1) and sets the variable `errno` accordingly. This allows interpretation of the failure on receiving a -1 and to take action accordingly.

Successful calls never set `errno`; once set, it remains until another error occurs. It should only be examined after an error. Note that a number of system calls overload the meanings of these error numbers, and that the meanings must be interpreted according to the type and circumstances of the call.

See [Error Codes](#) for the list of manifest constants exported.

Portability

The values reported by `errno` are system and function dependent.

See Also

[perror](#), [strerror](#)

Macros

__breaksw

```
__breaksw;
```

Switch break statement.

Description

The `__breaksw` statement is used to implement the `break` statement within `switch` statements.

Note:

The `__breaksw()` primitive is **internal** to the macro language. It is generated by the GRIEF macro compiler when a `break` statement is encountered within a `switch` statement; it not intended for direct use.

Returns

nothing

See Also

[Jump Statements](#), [switch](#), [while](#), [do](#), [for](#)

autoload

```
void autoload(string filename,
             string function,
             ...       )
```

Register location of one or more macros.

Description

The `autoload()` primitive informs the macro loader about the existence of global macros. The primitive registers the specified macro `function` against the macro object image `filename`, permitting an arbitrary number of macro names after the initial function; as such one or more function references may be associated.

Whenever the macro engine encounters an undefined macro reference, it searches the internal autoload list for a definition. If available, the associated macro object shall be loaded as normal, with the loader searching the **GRPATH** definition for the underlying object image.

If during an autoload driven load the referenced function was not resolved the following diagnostic shall be generated.

```
load_macro: macro 'xxx' not resolved during autoload.
```

A function reference may only be associated with one macro object, with any secondary references ignored generating the following diagnostic message.

```
autoload: 'xxx' already defined.
```

Example

The macros *modeline* and *mode* are registers against the macro object *modeline*.

```
autoload("modeline", "modeline", "mode");
```

Parameters

`filename` String that contains the name of the compiled macro file.
`function` String containing the name of the macro function.
`...` Additional macro function names.

Returns

nothing

Portability

n/a

See Also

[require](#)

bless

```
int bless([int ref],  
         [string classname])
```

Associate an object with a class/module.

Description

The *bless* primitive is reserved for future use.

This primitive informs the dictionary referenced by *ref* that it is now an object in the *classname* package. If *classname* is omitted, the current module is used. Because a *bless* is often the last thing in a constructor, it returns the reference for convenience.

The *bless* primitive enables Perl style objects.

Note:

Consider always blessing objects in *classname*'s that are mixed case, using a leading upper-case; Namespaces with all lowercase names are considered reserved for GRIEF pragmata.

Parameters

n/a

Returns

The *bless* statement returns the dictionary identifier.

Portability

A **GriefEdit** extension.

See Also

[module](#)

break

```
break;
```

break statement.

Description

The *break* statement is used to terminate *switch*, *while*, *do* and *for* loops.

Returns

nothing

See Also

[Jump Statements](#), [switch](#), [while](#), [do](#), [for](#)

call_registered_macro

```
int call_registered_macro(int type)
```

Invoke registered macro callbacks.

Description

The `call_registered_macro()` primitive invokes all of functions which have register against the particular event type `type`.

See [register_macro](#) for the particulars on the different registered event types.

Parameters

`type` Event type to be invoked.

Returns

The `call_registered_macro` returns nothing.

Portability

The set of available events differ between systems.

See Also

[register_macro](#)

case

```
'case' expr:
```

Switch case statement.

Description

The `switch` and `case` statements help control complex conditional and branching operations.

The body of a switch statement may have an arbitrary number of unique `case` labels.

If condition evaluates to the value that is equal to the value of one of `case` expressions, then control is transferred to the statement that is labelled with that expression.

If condition evaluates to the value that does not match any of the `case` labels, and the `default` label is present, control is transferred to the statement labelled as the `default` label.

Parameters

`expr` Case value, which can be an integer, float or string expression.

Returns

nothing

See Also

[Selection Statements](#), [switch](#)

catch

```
catch statement;
```

Catch statement.

Description

The `catch` statement is reserved for future use, shall implement try, catch and finally constructs.

Parameters

none

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[try](#), [finally](#), [throw](#)

continue

```
continue;
```

Loop continuation.

Description

The *continue* statement forces transfer of control to the controlling expression of the smallest enclosing **do**, **for**, or <while> loop.

A *continue* statement may only appear within a loop body, and causes a jump to the inner-most loop-continuation statement (the end of the loop body).

In a **while** loop, the jump is effectively back to the **while**.

In a **do** loop, the jump is effectively down to the **while**

In a **for** statement, the jump is effectively to the closing brace of the compound-statement that is the subject of the **for** loop. The third expression in the **for** statement, which is often an increment or decrement operation, is then executed before control is returned to the loop's conditional expression.

Parameters

none

Returns

nothing.

Portability

n/a

See Also

[Jump Statements](#), [while](#), [for](#), [foreach](#)

create_dictionary

```
int create_dictionary( string ~name,
                      int    ~tablesize,
                      int    ~tablefactor)
```

Create a dictionary.

Description

The *create_dictionary()* primitive creates a new dictionary resource.

A dictionary is a collections of associations. Dictionaries consist of pairs of keys and their corresponding values, both arbitrary data values. Dictionaries are also known as associative arrays or hash tables.

Parameters

name Optional string containing the unique dictionary name, if omitted the dictionary shall be unnamed.
 size Optional position integer specifying the size of the underlying table.
 factor Optional position integer specifying the table fill factor.

Returns

The *create_dictionary()* primitive returns the identifier associated with the new dictionary.

Portability

A **GriefEdit** extension.

See Also

[delete_dictionary](#), [set_property](#), [get_property](#)

delete_dictionary

```
int delete_dictionary(int obj_id)
```

Destroy a dictionary.

Description

The *delete_dictionary()* primitive destroys the specified dictionary *obj_id*.

Parameters

obj_id Dictionary identifier.

Returns

The *delete_dictionary()* primitive returns 0 on success, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#)

delete_macro

```
void delete_macro(string macro)
```

Delete a macro from memory.

Description

The *delete_macro* macro deletes a macro file, which contains one or more macros, from memory; the underlying macro object retains unchanged.

Any keyboard references and variables, both global and buffer will be retained. If any of the macros within the deleted macro object are referenced at a later time, the normal macro load logical is applied which may in turn reload the object.

Note:

This primitive is provided for BRIEF compatibility and is currently a no-op as macros are not directly deletable. The storage allocated to a macro shall be lost when a macro by the same name is loaded from another file.

Parameters

macro String containing the name of a macro file to be deleted. If no extension is supplied, then .cm shall be appended. If omitted the user is prompted.

Returns

nothing

Portability

Not implemented.

See Also

[load_macro](#)

dict_clear

```
int dict_clear(int obj_id)
```

Clear a dictionary.

Description

The *dict_clear()* primitive removes all items of the specified dictionary *obj_id*.

Parameters

obj_id Dictionary identifier.

Returns

The *dict_clear()* primitive returns 0 on success, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [dict_delete](#), [dict_each](#), [dict_keys](#), [dict_values](#)

dict_delete

```
int dict_delete(int    obj_id,
                string key    )
```

Remove a dictionary item.

Description

The *dict_delete()* primitive removes the item associated with the item *key* from the specified dictionary *obj_id*.

Parameters

obj_id Dictionary identifier.
key Item key.

Returns

The *dict_clear()* primitive returns 0 on success, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [dict_each](#), [dict_keys](#), [dict_values](#)

dict_each

```
int dict_each( int obj_id,
               [string key],
               [declare value])
```

Iterator a dictionary.

Description

The *dict_each()* primitive iterates thru a dictionary.

Returns the element index started at one and populates *key* and *value* with the entry details, so one may iterate over all elements within the dictionary.

There is no guarantee of order, entries are returned in an apparently random order; driven by the underlying hashing value.

When the dictionary is entirely read, a zero or **FALSE** value is returned.

You must not modify the dictionary whilst iterating over it. There is a single iterator for each dictionary, shared by *dict_each*, *dict_keys* and *dict_values* primitives, as such care should be taken not to intermix their usage.

The following prints out your environment like the *printenv* program, only in a different order:

```
int index;
while ((index = dict_each(dict, key, value)) >= 0) {
    insertf("%d:%s=%s\n", index, key, value);
}
```

Parameters

obj_id Dictionary identifier.
key Retrieval key.
value Value.

Returns

The primitive returns the positive element index on success, otherwise zero at the end of the dictionary.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [dict_keys](#), [dict_values](#)

dict_exists

```
int dict_exists( int    obj_id,
                 string key )
```

Dictionary item existence check.

Description

The *dict_exists()* primitive determines the existence of an item within the specified dictionary *obj_id*.

Parameters

obj_id Dictionary identifier.
key Item key.

Returns

The *dict_exists()* primitive returns TRUE on success otherwise FALSE.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [dict_each](#), [dict_keys](#), [dict_values](#)

dict_keys

```
int dict_keys( int obj_id,
               [string key])
```

Iterator dictionary keys.

Description

The *dict_keys()* primitive iterates thru a dictionary.

Returns the element index started at one and populates *key* with the entry details, so one may iterate over all elements within the dictionary.

There is no guarantee of order, entries are returned in an apparently random order; driven by the underlying hashing value.

When the dictionary is entirely read, a zero or **FALSE** value is returned.

You must not modify the dictionary whilst iterating over it. There is a single iterator for each dictionary, shared by [dict_each](#), [dict_keys](#) and [dict_values](#) primitives, as such care should be taken not to intermix their usage.

The following prints out your environment like the `printenv` program, only in a different order:

```
int index;
while ((index = dict_keys(dict, key)) >= 0) {
    insertf("%d:%s\n", index, key);
}
```

Parameters

obj_id Dictionary identifier.
key Retrieval key.

Returns

The primitive returns the positive element index on success, otherwise zero at the end of the dictionary.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [dict_each](#), [dict_values](#)

dict_list

```
list dict_list(int obj_id)
```

Retrieve dictionary items.

Description

The `dict_list()` primitive retrieves the keys or values contained within the specified dictionary `obj_id`.

Parameters

`obj_id` Dictionary identifier.

`keys` Optional boolean value unless **FALSE** key values are retrieved otherwise values are retrieved.

Returns

The `dict_list()` primitive returns a list of values containing all the keys or values within the specified dictionary, otherwise a null list.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#)

dict_name

```
string dict_name(int obj_id)
```

Retrieve a dictionary name.

Description

The `dict_name()` primitive retrieves the name associated with the specified dictionary `obj_id`.

Parameters

`obj_id` Dictionary identifier.

Returns

The `dict_name()` primitive returns the name of the dictionary otherwise an empty string.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#)

dict_values

```
int dict_values( int obj_id,
                 [declare value])
```

Iterator dictionary values.

Description

The `dict_values()` primitive iterates thru a dictionary.

Returns the element index started at one and populates `value` with the entry details, so one may iterate over all elements within the dictionary.

There is no guarantee of order, entries are returned in an apparently random order; driven by the underlying hashing value.

When the dictionary is entirely read, a zero or **FALSE** value is returned.

You must not modify the dictionary whilst iterating over it. There is a single iterator for each dictionary, shared by `dict_each`, `dict_keys` and `dict_values` primitives, as such care should be taken not to intermix their usage.

The following prints out your environment like the `printenv` program, only in a different order:

```

int index;
while ((index = dict_each(dict, value)) >= 0) {
    insertf("%d=%s\n", index, value);
}

```

Parameters

`obj_id` Dictionary identifier.
`value` Value.

Returns

The primitive returns the positive element index on success, otherwise zero at the end of the dictionary.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [dict_each](#), [dict_keys](#)

do

```
do statement; while (condition);
```

do statement.

Description

The `do` statement implements the *do-while* loop construct.

```

do {
    statements;
} while (condition);

```

Executes statements one or more times until *condition* is non-zero. The condition is tested after each iteration of the loop.

Portability

n/a

See Also

[Iteration Statements](#), [while](#), [break](#), [continue](#), [for](#), [if](#)

else

```
if (expr) true-body else false-body
```

else statement.

Description

The `else` statement implements the secondary condition in the if-then-else construct.

See Also

[Selection Statements](#)

execute_macro

```
declare execute_macro([string cmd],
                     ...)
```

Invokes a command by name.

Description

The `execute_macro()` primitive invokes the specified command *cmd*, which if omitted the user shall be prompted as follows:

Command:

When an invoked command is undefined, GRIEF shall attempt to load the associated macro file sourced from the directories listed in the **GRPATH**. If the macro file is successfully loaded and the command is found, the macro is then executed.

Note:

The *execute_macro* primitive cannot be overloaded with a replacement macro.

Parameters

- cmd** Optional string buffer containing the command to be executed. Otherwise if omitted the user shall be prompted for the command.
- ... Depending on the macro, the executed command may expect a sequence of additional arguments.

Returns

The *execute_macro()* primitive returns the result of the executed command, otherwise upon an error -2 shall be returned..

Portability

n/a

See Also

[load_macro](#)

exit

```
void exit([string y_or_w])
```

Exit current process level.

Description

The *exit()* primitive signals to leave the current process loop, which if the top level loop causes **GriefEdit** to terminate to the operating system.

Upon leaving the top level prior to exiting when modified buffers are present the user shall be prompted as to whether or not the buffers should be saved, as follows;

1 buffer has not been saved. Exit [ynw]?

The user may have all modified buffers saved prior to terminating by replying with either "W" or "w", otherwise terminate without saving any buffers using "Y" or "y". Alternatively the user may reject the exit signal altogether with a reply of "N" or "n".

Parameters

- y_or_w** Optional string that is applied to the answer regarding the action to occur when modified buffers are detected; **Y** or **y** **GriefEdit** shall exit without saving any modifiers buffers, whereas **W** or **w** all modified buffers are written.
Any other values other than "YyWW" shall cause the user to be prompted. In addition, the parameter is ignored if *exit* is being applied to a secondary process loop, which would not cause **GriefEdit** to terminate.

Returns

nothing

Portability

n/a

See Also

[abort](#), [process](#)

finally

```
finally statement;
```

Finally statement.

Description

The *finally* statement is reserved for future use, shall implement try, catch and finally constructs.

Parameters

none

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[try](#), [catch](#), [throw](#)

first_time

```
int first_time()
```

Determine a macros initialisation status.

Description

The *first_time()* primitive returns **true** if this is the first invocation of the calling macro since loading.

Each macro maintains a *first_time* status, which during the first execution of the macro is **true** and all subsequent calls shall be **false**. This primitive is generally used to perform run-time subsystem initialisation, for example.

```
if (first_time()) {
    initialise();
}
```

The loading of macro resets the *first_time* flag, is effect deleting the macro and reloading will look like it was never loaded.

Note:

A GRIEF macro compiler utilises this primitive to perform run-time initialisation of function **static** variables.

Parameters

none

Returns

The *first_time()* primitive returns **true** (non-zero) if the first invocation of the associated macro, zero otherwise.

Portability

n/a

See Also

[static](#)

for

```
for ([initialise];
     [condition];
     [increment] ) statements;
```

for statement.

Description

The *for* statement implements the *for* loop construct.

```
for (initialise; condition; increment)
{
    statements;
}
```

Repeatedly performing *statements* while the *condition* is **true** (non-zero).

When the for function starts, *initialise* is evaluated once. It evaluates *condition* and if the result is non-zero, it evaluates the *statements*. Finally, *increment* is evaluated, and control returns to the condition evaluation. The process is repeated from here until the condition is evaluated zero.

Portability

n/a

See Also

[Iteration Statements](#), [break](#), [continue](#), [while](#), [do](#)

foreach

```
void foreach(<body>,
            <expr>,
            <key>,
            <value>,
            <index>,
            <increment>)
```

Container iterator.

Description

The *foreach* statement is reserved for future use.

Parameters

n/a

Returns

The *foreach* statement returns the element index retrieved, otherwise -1 upon an end-of-source condition or error.

Portability

n/a

See Also

[list_each](#), [dict_each](#)

fstype

```
int fstype([string path])
```

File system type.

Description

The *fstype()* primitive retrieves the underlying file-system type.

Parameters

path Optional string containing path on the file-system to be tested, if omitted the current working directory is referenced.

Returns

The *fstype()* primitive returns a system dependent file-system identifier, otherwise -1 on error or not supported.

Portability

A **GriefEdit** extension.

See Also

[stat](#)

get_property

```
declare get_property(int obj_id,
                    string key )
```

Retrieve a dictionary item.

Description

The *get_property()* primitive retrieves the value of an item from the specified dictionary *obj_id*.

This primitive is generally not required within the GRIEF Macro Language. When the user accesses an object member, the compiler converts the construct into calls to this macro.

The following are equivalent;

```
get_property(object, "member");
and:
object.member;
```

Parameters

obj_id Dictionary identifier.

key Item key.

Returns

The *get_property()* primitive returns the item value otherwise NULL.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [set_property](#), [dict_exists](#), [dict_each](#)

if

```
if (expr) true-body
```

if statement.

Description

The *if* statement implements the *if-then-else* construct. The *else* is optional. Evaluates the condition *expr*; if the condition is non-zero, executes the *if_statements*; otherwise, it executes the *else_statements*.

```
if (condition)
  { if_statements }
[else]
  { else_statements }
```

See Also

[Selection Statements](#), [else](#), [while](#)

inq_btn2_action

```
int inq_btn2_action()
```

Retrieve the second button action.

Description

The *inq_btn2_action()* primitive is reserved for future BRIEF compatibility.

The *inq_btn2_action()* primitive retrieves the current action status of the second mouse button. The function can be used to assign a mouse action handler into a newly pushed keyboard, which shall be automatically removed upon the keyboard destruction.

Parameters

none

Returns

Returns the current action button status.

Portability

n/a

inq_called

```
string inq_called()
```

Get the name of the calling macro.

Description

The *inq_called()* primitive returns the callers name, allowing macros to differentiate between being called directly from the command prompt, or from another macro.

This primitive is provided to support [replacement](#) macros and to allow macros to determine whether user prompts [get_parm](#) are suitable and [message](#) output may be required.

Returns

The *inq_calling()* primitive returns a string containing the name of the macro which called the current macro, otherwise an empty string where invoked from the keyboard.

The [set_calling_name](#) can be used to modify the value returned in-turn controlling the behaviour of the macros which are then invoked.

See Also

[set_calling_name](#), [replacement](#)

inq_macro

```
int inq_macro(string name,
              [int mode = 0])
```

Determine whether a macro or primitive exists.

Description

The *inq_macro()* primitive tests whether the specified macro *name* exists within a stated namespace(s) defined by *mode*.

Parameters

name String containing the macro name to be tested.

mode Optional integer test type, if omitted defaults to 0, see table for mode behaviours.

Tests

The following test behaviours are available given by *mode*;

Value	Description
0x00	Default behaviour if <i>mode</i> is omitted. inq_macro tests specified macro name against both runtime loaded macros and built-in primitive. It returns 1 if the macro exists (loaded, autoload or a primitive replacement), 0 if it is a built-in primitive, otherwise -1 if neither a built-in primitive nor macro. Note: The returns are incompatible with the original BRIEF implementation.
0x01	inq_macro only tests whether the specified macro <i>name</i> is runtime loaded macro, ignoring built-in primitives. It returns 1 if the macro is loaded, 2 if it exists as an autoload but has yet to be loaded otherwise 0 if the macro does not exist.
0x02	inq_macro returns the assigned positive module number (See: inq_module) if the macro has been loaded, 0 if it is an autoload, otherwise -1 if the macro does not exist.
0x03	inq_macro only tests whether the specified macro <i>name</i> is a primitive, ignoring runtime loaded macros It returns 1 if the primitive exists, 2 if it has been replaced (See: replacement) otherwise 0 if the primitive does not exist.

Returns

The *inq_macro()* primitive returns a mode specified value, otherwise -1 if the stated *mode* was invalid.

Portability

The autoload visibility under mode 0x01 and mode 0x03 are **GriefEdit** extensions.

See Also[load_macro](#)**inq_macro_history**

```
string inq_macro_history([int index = 0])
```

Retrieve macro execution history.

Description

The *inq_macro_history()* primitive retrieves the name of the previously executed macro as a result of a keyboard binding.

If *index* is specified, it specifies the history index starting at an offset of zero, otherwise the most recent (i.e. index 0) is returns.

The *home* and *end* macros are good examples of its usage, which modify their behaviour based upon previous operations.

Parameters

index Optional int index.

Returns

The *inq_macro_history()* primitive returns the name of the macro invoked, otherwise an empty string if the item does not exist.

Portability

A **GriefEdit** extension, matching similar CRISP Edit functionality of the same name.

See Also[set_macro_history](#), [inq_command](#)**inq_module**

```
string inq_module([int test = 1])
```

Retrieve the current module.

Description

The *inq_module()* primitive retrieves the module name assigned with the calling macro. When loaded macros are automatically assigned with a unique module name or can be specifically assigned using the [module](#) primitive.

Module names are used to implement name spaces for static macro functions or to provide a way of creating a macro package split amongst source files.

Example**The following example setups a static scoped callback function**

```
void
main(void)
{
    register_callback( inq_module() + "::callback" );
}

static void
callback(void)
{}
```

Returns

The *inq_module()* primitive returns the name of the current module.

Portability

A **GriefEdit** extension.

See Also[module](#), [static](#), <scope>

inq_msg_level

```
int inq_msg_level()
```

Get the message level.

Description

The *inq_msg_level()* primitive retrieves the current message level.

- 0 All messages are enabled; the default value.
- 1 Normal messages are not displayed, error messages still display.
- 2 Error messages are suppressed.
- 3 Suppress all messages, both message and error.

Parameters

none

Returns

The *inq_msg_level()* primitive returns the message level.

Portability

n/a

See Also

[set_msg_level](#)

list_of_dictionaries

```
list list_of_dictionaries( [bool nonempty = false],  
                           [bool named = false]      )
```

List of created dictionaries.

Description

The *list_of_dictionaries()* primitive retrieves a list of dictionary identifiers created by the [create_dictionary](#) primitive.

Parameters

- nonempty* Empty diction filter. Optional integer boolean flag. If specified and is not-zero, then empty dictionaries shall be filtered.
named Named dictionary filter. Optional integer boolean flag. If specified and is not-zero, then unnamed dictionaries shall be filtered.

Returns

The *list_of_dictionaries()* primitive retrieves a list of dictionaries.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#)

load_macro

```
int load_macro(string filename,  
               [int reload = 1])
```

Load a macro object.

Description

The *load_macro()* primitive loads the specified macro object *filename*, resolving all external references in the file.

If *filename* does not contain a path specification, then all directories contained within the **GRPATH** environment variable are searched.

If *reload* is either omitted or non-zero, the macro shall be reloaded regardless of whether the object image has already been loaded using *load_macro* or a **autoload** reference. Otherwise if zero the image shall not be reloaded if a image with the same name was loaded previously.

Once a successful load the macro objects **main** entry point is executed.

Note:

Upon loading pre-compiled macro objects (.cm), several interface checks occur confirming whether the object is compatible with the current GRIEF version, for example engine version number and primitive count. Check failure results in a load error and suitable diagnostics being echoed on the command prompt.

Parameters

- `filename` String containing the macro object to be loaded. If the specified file has an extension, then an exact match is located. Otherwise files matching one of the following extensions are .cm, .cr, and .m are located.
- `reload` Optional boolean flag, if given as **false** and the same macro object was previously loaded, nothing shall be loaded.

Returns

The `load_macro()` primitive returns 1 if macro file was successfully loaded; return 2 if `reload` is non-zero and a loaded image already exists; otherwise 0 on failure.

Portability

Unlike BRIEF, macros in source form (.m files) can be loaded with this macro.

The `reload` option is an **GriefEdit** extension.

See Also

[autoload](#), [inq_macro](#), [main](#)

macro

```
declare macro(list declaration)
```

Define a macro body.

Description

The `macro` internal primitive is the mechanism by which macros are imported during the loading of a macro object.

Parameters

- `decl` List containing the macro declaration, including scoping flags, macro name and the function body.

Note:

This interface should be considered as a **internal** primitive, reserved for exclusive use by the GRIEF Macro Compiler and may change without notice. Management of macro definitions plus any associated argument binding shall be handled automatically by the compiler.

Registering an ill-formed macro declaration shall have undefined effects, most likely resulting in GRIEF crashes.

Returns

Result of any `main` and/or associated internal `_init` execution.

Portability

n/a

See Also

[replacement](#)

module

```
int module(string modulename)
```

Assign a module identifier.

Description

GRIEF provides a mechanism for alternative namespaces, being an abstract container providing context for the items, to protect modules from accessing on each other's variables.

The purpose of this functionality is to provide data hiding within a set of common macro files (objects), by allowed separate function and variables namespaces. Each namespace creates a container for a set of symbols, providing a level of indirection to specific identifiers, thus making it possible to distinguish between identifiers with the same

exact name, in effect reducing naming conflicts in unrelated objects (See: [Scope](#)).

The module statement declares the object as being in the given namespace. The scope of the module declaration is from the declaration itself and effects all current and future declarations within the associated object.

The intended usage of the module() primitive is the within the main function in all of the related objects.

The namespace specification should be a string containing a valid sequence of identifier symbols of the form

[A-Za-z_] [A-Za-z_0-9]*

Namespaces are in conjunction with static scoping of members ((see [static](#))). If you are writing a set of macros some of which are internal and some for external use, then you can use the *static* declaration specifier to restrict the visibility of a member variable or function.

However as static functions are hidden from usage outside their own macro file (or module), this can present a problem with functionality which involves the usage of callbacks (e.g. `assign_to_key`). In this case, the `::` (scope resolution) operator is used to qualify hidden names so that they can still be used.

Multiple objects can contained within the same namespace. The module primitive allows you to refer to static functions defined in another macro file explicitly, using the `::` naming modifier, and also allows static macro functions to be accessed in callbacks. Upon a module being associated, it is possible to refer use the syntax `<module-name>::<function>` to refer to a function in callbacks.

Example

```
void
main()
{
    module("my_module");
    assign_to_key("<Alt-D>", "my_module::doit");

    // which to has the identical behaviour as
    assign_to_key("<Alt-D>", "::doit");

    // and
    assign_to_key("<Alt-D>", inq_module() + "::doit");
}

static void
doit()
{
    :
}
```

Returns

The *module* primitive returns 0 on success, otherwise 1 if *module* already existed or -1 if called outside the context of any macro file.

Portability

A [GriefEdit](#) extension.

See Also

[inq_module](#), [<scope>](#), [static](#), [assign_to_key](#)

nothing

void nothing()

Noop.

Description

The *nothing()* primitive is a no operation, noop for short, This primitive it acts as a place holder when a dummy function is required, effectively does nothing at all.

Parameters

none

Returns

nothing

Portability

n/a

See Also[error](#)**register_macro**

```
int register_macro( int type,
                    string macro,
                    [int local = FALSE])
```

Register a callback procedure.

Description

The `registered_macro()` primitive registers a function to be invoked upon the trigger of the event type `type`. Multiple macros may be associated against any particular event type, in which case upon execution they are called in **FIFO** order.

The argument `macro` is the name of a macro to be invoked upon the event being triggered. If `local` is specified and is non-zero, then the macro is only registered against the current buffer, otherwise the event may be triggered even when the current buffer is not selected.

Registered macros may be invoked using `call_registered_macro` and removed by using the `unregister_macro` primitive.

Constant	Description
REG_TYPED	Character typed.
REG_EDIT	Different file edited.
REG_ALT_H	ALT-H pressed in response to a prompt.
REG_UNASSIGNED	Unassigned key pressed.
REG_IDLE	Idle time expired.
REG_EXIT	About to exit.
REG_NEW	New file edited and readin.
REG_CTRL_C	CTRL-C (SIGINT) pressed during macro.
REG_INVALID	Invalid key pressed during response input.
REG_INTERNAL	Internal error.
REG_MOUSE	Mouse callback.
REG_PROC_INPUT	Process input available.
REG_KEYBOARD	Keyboard buffer empty.
REG_STARTUP	Startup complete.
REG_BOOKMARK	Bookmark dropped/deleted.
REG_INSERT_MODE	Insert mode has changed.
REG_BUFFER_MOD	Buffer has been modified.
REG_BUFFER_WRITE	Buffer write operation.
REG_BUFFER_RENAME	Buffer rename operation.
REG_BUFFER_DELETE	buffer delete operation.
REG_FILE_SAVE	File write request.
REG_FILE_WRITTEN	File write completion.
REG_FILE_CHANGE	File external change.
REG_SIGUSR1	SIGUSR1 signal trap.
REG_SIGUSR2	SIGUSR2 signal trap.

Parameters

- `type` Event type against which to register.
- `name` Name of the macro to be registered.
- `local` Optional int, Whether the trigger is of local or global scope. Note currently local is only effective on REG_TYPED.

Returns

The `registered_macro()` primitive returns 1 if the macro was successfully registered, 0 if already registered, otherwise -1 on error.

Portability

The set of available events differ between systems.

See Also

reload_buffer

```
int reload_buffer([int bufnum],
                 [string encoding])
```

Reload the buffer content.

Description

The *reload_buffer()* primitive is reserved for future use.

Parameters

bufnum	Optional buffer number, if omitted the current buffer shall be referenced.
encoding	Optional string containing the character encoding to be applied to the source file.

Returns

n/a

Portability

n/a

See Also

[edit_file](#)

replacement

```
replacement <macro-definition>
```

Overload an existing macro definition.

Description

The *replacement* macro modifier is used to explicitly declare overloaded interfaces, which is a macro that supersedes (or complements) either another macro or directly executable function of the same name.

Replacement macros are used to intercept calls to other macros or functions and are usually to enhance existing the functionality of those commands without rewriting them completely.

For example the following macro replaces *edit_file*;

```
edit_file()
{
    edit_file();
    beep();
}
```

Any reference to *edit_file()* in the replacement macro refers to the function which is replaced; hence in this example *edit_file()*, then calls the original *edit_file()* and then beeps() on its return.

During a compile when a macro which overloads a built-in macro is encountered, the compiler shall generate a warning message. To suppress the warning the keyword *replacement* should be stated prior to the function name, as follows:

```
replacement
edit_file()
{
    edit_file();
    beep();
}
```

Note that the only behaviour effected by the *replacement* keyword is the warning suppression and defining a function the same as built-in shall always result in an implicit function overloading. As such the two examples are functionality equivalent.

Returns

n/a

Portability

n/a

See Also

[macro](#), [static](#), [extern](#), [global](#)

require

```
int require(string filename)
```

Enforce the use of an external module.

Description

The *require()* primitive enforces that the specified macro object *filename* be loaded if it has not already been loaded.

This primitive is similar to using [load_macro](#) with a *reload* option of zero, but is provided as a more convenient and explicit form.

The best practice of this primitive, use *require* within the *main* function of any given macro object.

```
void main(void)
{
    require("utils"); // our external dependency
}
```

Note:

[autoload](#) and *require* usage should generally be exclusive for any given macro object.

Within CRISPEdit™ documentation it is stated that the results are undefined due to the tables maintained for each are separate which can result in the macro being mistakenly loaded more than once. As such the side-effect of the reload shall be any global state the macro was maintained shall be lost.

Under GRIEF their usage **may** be mixed yet for compatibility this style of usage should be avoided.

Returns

Returns 0 if already loaded; 1 if the macro as loaded. Otherwise -1 if the macro file could not be loaded.

Portability

n/a

See Also

[load_macro](#), [autoload](#)

reregister_macro

```
int reregister_macro( int type,
                      string macro,
                      [int local = FALSE])
```

Register a unique callback procedure.

Description

The *reregistered_macro()* primitive registers a unique function to be invoked upon the trigger of the event type *type*. Similar to [register_macro](#) yet only permits a single instance of the given function tp be registered.

This primitive allows macros to unconditionally register handlers without need to know whether a previous instance has been installed, unlike [register_macro](#) which shall permit multiple instances to exist.

See [register_macro](#) for the particulars on the different registered event types.

Parameters

- type* Event type against which to register.
- name* Name of the macro to be registered.
- local* Optional int, Whether the trigger is of local or global scope. Note currently local is only effective on REG_TYPED.

Returns

The *reregistered_macro()* primitive returns 1 if the macro was uniquely registered, 0 if already registered, otherwise -1 on error.

Portability

The set of available events differ between systems.

See Also

[register_macro](#)

restore_position

```
int restore_position([int what = 1])
```

Restore a previously saved position.

Description

The *restore_position()* primitive restores a previously saved position from the position stack. The argument *what* is an optional integer expression which controls the state that is restored.

When states *what*, specifies the information which is restored. If omitted what is equivalent to one;

- 0 Nothing, with the save information being discarded.
- 1 The cursor position is restored.
- 2 The buffer is restored, with the cursor located in the current window at its previous position.
- 3 Reserved.
- 4 The previous buffer and window are restored, with the cursor located at its previous position.

Parameters

what Optional integer states the what elements of the position state to be restored.

Returns

The *restore_position()* primitive returns 1 on success, otherwise zero or less on error.

Portability

n/a

See Also

[save_position](#)

return

```
return [<expression>];
```

Return from a macro.

Description

The *return()* primitive terminate the current macro and returns the given value *expression*. If specified *expression* may be an integer, string or list expression which matches the return type of the associated macro.

If **return** or **returns** is not called, the return value for a macro is indeterminate.

Parameters

expression Optional value to be returned, which should match the return type of the associated macro.

Returns

nothing

See Also

[returns](#), [exit](#), *

returns

```
returns (expression);
```

Return an expression from a macro.

Description

The *returns()* primitive sets the return value of the current macro to *expression*, which may be an integer, string or list expression which matches the return type of the associated macro.

This primitive is similar to the **return** statement, except it does not cause the current macro to terminate. It simply sets **GRIEF's** internal accumulator with the value of the expression; as such any following may overwrite the value.

Note:

If `return` or `returns` is not called, the return value for a macro is indeterminate.

Note:

This primitive is not strictly compatible with the `returns` macro of BRIEF and is not generally recommended as statements following may have side effects.

Parameters

`expression` Value to be returned, which should match the return type of the associated macro.

Returns

`nothing`

See Also

[Jump Statements](#), [return](#), [exit](#)

save_position

```
void save_position()
```

Saves current cursor/buffer state.

Description

The `save_position()` primitive pushes the current buffer, window and cursor position into the position stack, allowing them to be restored later using `restore_position`.

As these states are maintained by a stack in LIFO (Last In First Out) order each invocation of `save_position` should have a corresponding invocation of `restore_position`, otherwise saved positions shall accumulate consuming system resources.

Note:

That the position stack is an independent of any buffer permitting `restore_position` to be called to pop off the top entry of the stack even when the current buffer is not the same as buffer referenced by the top of the saved position stack.

Parameters

`none`

Returns

`nothing`

Portability

`n/a`

See Also

[restore_position](#)

set_btn2_action

```
int set_btn2_action([int action])
```

Set the second button action.

Description

The `set_btn2_action()` primitive is reserved for future BRIEF compatibility.

The `set_btn2_action()` primitive sets or toggles the default action for the second mouse button. If `action` is zero, the Quick-Edit action will be the default. If `action` is non-zero, the Quick-Menu action will be the default. If `action` is omitted, the current setting shall be toggled.

Parameters

`action` Optional integer, the new Quick-Menu button action.

Returns

Returns the previous action button status.

Portability

n/a

set_calling_name

```
void set_calling_name(string name = NULL)
```

Set the name of the calling macro.

Description

The *set_calling_name()* primitive sets the calling name of the current macro, which is the name that would be returned by *inq_name()* within any macros called. If omitted then the name is cleared and returned to the original name.

set_calling_name is used to modify the value returned in-turn controlling the behaviour of the macros which are then invoked.

Another common use is within replacement macros, to forward the original callers name onto the next macro.

```
set_calling_name(inq_called());
```

Returns

Nothing

See Also

[inq_called](#), [replacement](#)

set_macro_history

```
string set_macro_history( [int index = 0],  
                         [string value] )
```

Set the macro execution history.

Description

The *set_macro_history()* primitive replaces an entry within the macro execution history.

index specifies the history index starting at an offset of zero. *value* is new replace name.

Note:

GRIEF maintains the history to the depth of 16.

Parameters

index Optional int index.

Returns

The *set_macro_history()* primitive returns the previous value name of entry replaced, otherwise an empty string if the item does not exist.

Portability

A **GriefEdit** extension, matching similar CRISPEdit functionality of the same name.

See Also

[inq_macro_history](#)

set_property

```
int set_property( int    obj_id,  
                  string key,  
                  [declare value])
```

Set a dictionary item.

Description

The `set_property()` primitive sets the value of an item within the specified dictionary `obj_id`.

This primitive is generally not required within the GRIEF Macro Language. When the user accesses an object member, the compiler converts the construct into calls to this macro.

The following are equivalent;

```
set_property(object, "member", value);
and:
object.member = value;
```

Parameters

`obj_id` Dictionary identifier.
`key` Item key.
`value` Item value.

Returns

The `set_property()` primitive returns TRUE on success otherwise FALSE.

Portability

A **GriefEdit** extension.

See Also

[create_dictionary](#), [get_property](#), [dict_exists](#), [dict_each](#)

switch

```
switch(expr) statement
```

Switch statement.

Description

The `switch` and `case` statements help control complex conditional and branching operations. The `switch` statement transfers control to a statement within its body. The body of a `switch` statement may have an arbitrary number of unique `case` labels, for example.

```
switch (value) {
    case 1:
        w = "one";
        break;
    case 2:
        w = "two";
        break;
    default:
        w = "others";
        break;
}
```

If condition evaluates to the value that is equal to the value of one of case expressions, then control is transferred to the statement that is labelled with that expression.

If condition evaluates to the value that does not match any of the `case` labels, and the `default` label is present, control is transferred to the statement labelled as the `default` label.

The `break` statement, when encountered in statement exits the `switch` statement.

Parameters

`expr` Switch value, which can be an integer, float or string expression.

Returns

nothing

Portability

n/a

See Also

[Selection Statements](#), `case`, `break`

throw

```
throw expr;
```

Throw an exception.

Description

The *throw* statement is reserved for future use.

Exceptions are used to indicate that an error has occurred while running the program. Exception objects that describe an error are created and then thrown with the *throw* primitive. The runtime then searches for the most compatible exception handler.

Parameters

expr Exception value.

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[try](#), [catch](#), [finally](#)

try

```
try statement; catch statement; finally statement;
```

Try statement.

Description

The *try* statement is reserved for future use, which shall implement try, catch and finally exception constructs.

Exceptions are used to indicate that an error has occurred while running the program. Exception objects that describe an error are created and then thrown with the *throw* primitive. The runtime then searches for the most compatible exception handler.

```
try {
    statement;

} catch type {
    statement;

} finally {
    statement;
}
```

Parameters

none

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[catch](#), [finally](#), [throw](#)

unregister_macro

```
int unregister_macro( int      type,
                      string macro,
                      [int local = FALSE])
```

Remove a registered macro.

Description

The *unregistered_macro()* primitive removes a previously registered macro.

If a particular macro has been registered multiple times than for each successful registration a corresponding unregister must occur to remove all instances; unregister_macro may be called in a loop until all instances are removed.

Parameters

- `type` Event type against which to unregister.
- `name` Name of the macro to be unregistered.
- `local` Optional int, Whether the trigger is of local or global scope. Note currently local is only effective on REG_TYPED.

Returns

The *unregistered_macro()* primitive returns 1 if macro was registered and has now been unregistered, otherwise 0.

Portability

The set of available events differ between systems.

See Also

[register_macro](#)

while

```
while ([condition]) statements;
```

while statement.

Description

The *while* statement implements the *while* loop construct.

```
while (condition)
{
    statements;
}
```

Repeatedly performs *statements* while the *condition* is **true** (non-zero). In each iteration of the loop, it first tests the condition, and if it is **true** it executes statement. This cycle is continued until condition is **false** (zero).

Portability

n/a

See Also

[Iteration Statements](#), [break](#), [continue](#), [for](#), [do](#)

`$Id: $`

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Miscellaneous Primitives

Summary

Miscellaneous Primitives

Macros

<code>_regress_op</code>	Regression operations.
<code>_regress_replacement</code>	Replacment regression testing.
<code>beep</code>	Issue a beep sound.
<code>color_index</code>	Border color background color index.
<code>date</code>	Get current system date.
<code>gmtime</code>	Convert a time value to UTC time components.
<code>grief_version</code>	GRIEF version.
<code>inq_clock</code>	Retrieve the user identifier.
<code>localtime</code>	Convert a time value to local time components.
<code>process_mouse</code>	Process mouse event.
<code>rand</code>	Generate a random number.
<code>set_msg_level</code>	Set level of informational messages.
<code>srand</code>	Seed the random number generator.
<code>time</code>	Get the current system time.
<code>version</code>	Version information.

Macros

`_regress_op`

```
int __regress_op(...)
```

Regression operations.

Description

The `__regress_op()` primitive is an **INTERNAL** function utilised using system regression testing, providing access to miscellaneous system library functions.

Returns

n/a

Warning:

Direct use of `__regress_op()` is not advised and is only documented for completeness.

Functionality may change or be removed without notice.

Portability

GriefEdit specific, functionality may change without notice.

See Also

`<regress>`

`_regress_replacement`

```
declare __regress_replacement(...)
```

Replacment regression testing.

Description

The `__regress_replacement()` primitive is an **internal** function utilised by the regress macro to assert replacement macro features.

Returns

n/a

Warning:

Direct use of `__regress_replacement()` is not advised and is only documented for completeness.

Functionality may change or be removed without notice.

Portability

A **GriefEdit** specific, functionality may change without notice.

See Also

`<regress>`, `__regress_op`

beep

```
int beep([int freq,
         [int duration])
```

Issue a beep sound.

Description

The `beep()` primitive sends a bell or beep to the screen causing an audible sound.

The function is synchronous; it performs an alertable wait and does not return control to its caller until the sound finishes.

`freq` and `duration` are system dependent values.

Parameters

`freq` The frequency of the sound, in hertz. This parameter must be in the range 37 through 32,767.

`duration` The duration of the sound, in milliseconds.

Returns

The `beep()` primitive returns 0 on success, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

`cursor`

color_index

```
int color_index([int index])
```

Border color background color index.

Description

The `color_index()` primitive sets the current value of the color index. The index controls the color that shall be assigned as the borderless background color to the next window created. On assignment the color index shall be automatically incremented and contained within the range 0 .. 16.

When borders are disabled, this color shall be used as the background of the associated window allowing one to distinguish between individual views.

Parameters

`index` Optional integer index between the ranges of 0 and 16, if omitted the current index is returned without effecting any change effectively behaving like a `inq_color_index` function.

Returns

Returns the previous value of the color index.

Portability

A **GriefEdit** extension.

See Also

`borders`, `create_buffer`

date

```
int date( [int &year],
          [int &month],
          [int &day],
          [string &monname],
          [string &dayname] )
```

Get current system date.

Description

The *date()* primitive retrieves the current date in local time.

The following numeric components are returned.

year Year, in the range [1900-2099].

month Month of the year in the range [1-12].

day Day of the month, in the range [1-31].

in addition if supplied the following string values are populated

monname Name of the month (e.g. "January").

dayname Name of the day (e.g. "Monday").

Returns

The *date* function returns the current value of time in seconds since the Epoch which the components represent.

Example

Displays the current date

```
int year, month, days;
string dayname;

date(year, month, day, NULL, dayname);
message("%s, %d/%d/%d", dayname, year, month, day);
```

See Also

[time](#), [localtime](#), [gmtime](#)

gmtime

```
int gmtime( [int time = NULL],
            [int &year],
            [int &mon],
            [int &mday],
            [string &monname],
            [string &dayname],
            [int &hour],
            [int &min],
            [int &sec] )
```

Convert a time value to UTC time components.

Description

The *gmtime()* primitive converts the *time* in seconds since the Epoch (1970/1/1) into its components, expressed as Coordinated Universal Time (UTC). If *time* is omitted the current time is broken-down into components.

Returns

The *gmtime* function returns the value of time in seconds since the Epoch which the components represent.

Portability

A **GriefEdit** extension

See Also

[time](#), [localtime](#)

grief_version

```
int grief_version()
```

GRIEF version.

Description

The *grief_version()* primitive retrieves the current GRIEF version.

Parameters

none

Returns

The *grief_version* primitive returns the current version multiplied by 100 plus the minor; for example *101* represents version *1.1*.

Portability

A **GriefEdit** extension.

See Also

[version](#)

inq_clock

```
int inq_clock()
```

Retrieve the user identifier.

Description

The *inq_clock()* primitive retrieves an approximation of the current CPU time used by the current edit session.

This primitive is an interface to the system *clock()* function, yet normalises the returned value to microseconds; to determine the number of seconds used, divide by the system constant **CLOCKS_PER_SEC**.

Note loops every 72 minutes.

Parameters

none

Returns

The *inq_clock* returns the time in microseconds since start of the current **GriefEdit** instance.

If the processor time used is not available or its value cannot be represented, the function returns the value -1.

Portability

n/a

See Also

[time](#), [sleep](#)

localtime

```
int localtime( [int time = NULL],
               [int &year],
               [int &mon],
               [int &mday],
               [string &monname],
               [string &dayname],
               [int &hour],
               [int &min],
               [int &sec] )
```

Convert a time value to local time components.

Description

The *localtime()* primitive converts the *time* in seconds since the Epoch (1970/1/1) into its components, expressed as local time also known as wall-clock. If *time* is omitted the current time is broken-down into components.

Returns

The *localtime* function returns the value of time in seconds since the Epoch which the components represent.

Portability

A **GriefEdit** extension.

See Also

[time](#), [gmtime](#)

process_mouse

```
void process_mouse(      [int b1],
                        [int b2],
                        [int b3],
                        int x,
                        int y      )
```

Process mouse event.

Description

The *process_mouse()* primitive allows an external mouse event to be processed; some mice types are handled internally whereas others require macro support.

Parameters

b1, b2, b3 Button states, zero for up otherwise non-zero for down.
x, y Screen coordinates.

Returns

nothing

Portability

n/a

See Also

[translate_pos](#)

rand

```
int rand([int upper])
```

Generate a random number.

Description

The *rand()* primitive computes a sequence of pseudo-random integers in the range 0 to *RAND_MAX*.

rand will by default produce a sequence of numbers that can be duplicated by calling *srand()* with 1 as the seed.

The *srand* primitive can be used to set/reset to random seed plus modify the generator table depth. This implementation uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $(2^{31})-1$.

The period of this random number generator is very large, approximately $16 * ((2^{31})-1)$.

Parameters

upper Optional integer stating the upper range of the returned random number, if omitted 2^{32} .

Returns

The *rand* primitive returns a random number in the range 0.. 2^{31} or 0..*upper* if a positive *upper* value is stated.

See Also

[srand](#)

set_msg_level

```
int set_msg_level(int level)
```

Set level of informational messages.

Description

The *set_msg_level()* primitive sets the message level for the duration of the current command.

The message level controls what type of messages are to be made visible on the status line; it allows macros to suppress messages from macros.

By default the message level is a value of 1 whenever a command is invoked from the keyboard or a registered macro, and set to zero when the command completes.

The specified *level* is a value in the range 0-3 with the following effects:

- 0 All messages are enabled; the default value.
- 1 Normal messages are not displayed, error messages still display.

- 2 Error messages are suppressed.
- 3 Suppress all messages, both message and error.

Parameters

`level` Integer value specifying the new message level.

Returns

The `set_msg_level()` primitive returns the previous message level.

Portability

n/a

See Also

`inq_msg_level`, `error`, `message`

strand

```
int strand([int seed = time()],
           [int depth] )
```

Seed the random number generator.

Description

The `strand()` primitive initialises the random number generator based on the given seeds.

The `seed` is used to prime the generator running at the specified `depth`.

By default, the package runs with 128 bytes of state information and generates far better random numbers than a linear congruential generator. If the amount of state information is less than 32 bytes, a simple linear congruential R.N.G. is used.

Parameters

`seed` Basic initial primer, if omitted defaults to the current value of `time`.

`depth` Optional integer depth controlling how sophisticated the random number generator shall be.

Current “optimal” values for the amount of state information are 8, 32, 64, 128, and 256 bytes; other amounts will be rounded down to the nearest known amount. Using less than 8 bytes will cause an error.

Returns

The `strand()` primitive returns 0 on success, otherwise -1 on error.

See Also

`rand`

time

```
int time([int &hour],
         [int &min],
         [int &sec],
         [int &msec] )
```

Get the current system time.

Description

The `time()` primitive retrieves the current time in local time.

The following numeric components are returned

`hours` Hour of the day, in the range [0-23].

`mins` Minutes of the hour, in the range [0-59].

`secs` Seconds of the minute, in the range [0-60].

`msecs` Milliseconds, in the range [0-9999]

Returns

The `time()` primitive returns the value of time in seconds since the Epoch (1970/1/1).

Example

Displays the current date

```

int hour, min, sec, msec;

time(hour, min, sec, msec);
message ("time, %d:%d:%d.%d", hour, min, sec, msec);

```

Portability

The *msec* parameter is a **GriefEdit** extension; the BRIEF version returned only hundredths of seconds.

See Also

[date](#), [localtime](#), [gmtime](#)

version

```

int version( [int major | string machtype],
             [int min],
             [int edit],
             [int release],
             [string machtype],
             [string compile],
             [int cmversion],
             [string features],
             [string build]
           )

```

Version information.

Description

The *version()* primitive retrieves the version information associated with the current GRIEF installation.

If the first argument is omitted, displays the current version and build information on the command prompt, for example:

```
GRIEF v3.2.0 compiled Aug 20 2014 20:05:04
```

Parameters

The first parameter may be either an integer or string variable, which shall be populated with the major *version* or the machine type representatively.

All additional parameters are either integer or string variable references which are populated with their associated value.

major	Integer major <i>version</i> number.
min	Integer minor <i>version</i> number.
edit	Integer sub <i>version</i> number.
machtype	Machine type labels, value include "DOS", "OS/2", "UNIX" and "VMS".
release	Reserved for future use.
compiled	GRIEF engine compilation timestamp.
cmversion	Macro compiler language <i>version</i> .
features	String of comma separated built-in features (reserved for future use).
build	Populated with the host build label.

Returns

The *version()* primitive returns the current *version* multiplied by 100, plus the minor; for example 301 represents *version 3.1*.

Portability

All the arguments are extensions.

See Also

[grief_version](#)

\$Id: \$

Movement Primitives

Summary

Movement Primitives

Macros

<code>backspace</code>	Delete character to the left of the cursor.
<code>beginning_of_line</code>	Goto beginning of line.
<code>bookmark_list</code>	Retrieve existing bookmark list.
<code>delete_bookmark</code>	Delete a bookmark.
<code>down</code>	Move position down one line.
<code>drop_bookmark</code>	Create or update a bookmark.
<code>end_of_buffer</code>	Move cursor to end of current buffer.
<code>end_of_line</code>	Goto end of line.
<code>end_of_window</code>	Goto end of the current window.
<code>goto_line</code>	Move to a particular line.
<code>goto_old_line</code>	Move to line before buffer modification.
<code>left</code>	Move position left one character.
<code>move_abs</code>	Move to an absolute location in the buffer.
<code>move_rel</code>	Move to a relative location in the buffer.
<code>next_char</code>	Move to the next character.
<code>page_down</code>	Move position down a page.
<code>page_up</code>	Move position up a page.
<code>parse_filename</code>	Parse a file into its components.
<code>prev_char</code>	Move to the previous character.
<code>swap_anchor</code>	Swaps the mark with the current position.
<code>top_of_buffer</code>	Move cursor to start of current buffer.
<code>top_of_window</code>	Goto top of the current window.

Macros

`backspace`

```
void backspace([int num = 1])
```

Delete character to the left of the cursor.

Description

The `backspace()` primitive moves the cursor and deletes the character to the left of the cursor in the current buffer.

The actions of `backspace` are dependent on the current insert mode.

Insert Mode

If insert mode backspaces moves the cursor and deletes the previous character, all characters to the right move one character to the left.

If the cursor is at the beginning of the line then the current line is appended to the end of the previous line.

Overtake Mode

If overstrike mode backspaces moves the cursor and deletes the previous character, replacing it with a space.

If the previous character is a tab, it moves over virtual spaces between the current position and the tab character when moving back.

If the cursor is at the beginning of the line then the cursor is moved to end of the previous line.

Parameters

`num` Optional integer, if stated specifies the number of characters the cursor to be moved backwards, if omitted only a single character position is moved.

Returns

The `backspace()` primitive returns non-zero if successful implying the cursor moved, otherwise zero.

Portability

The `num` option is a **GriefEdit** extension.

See Also

[delete_char, left](#)

beginning_of_line

```
int beginning_of_line()
```

Goto beginning of line.

Description

The *beginning_of_line()* primitive moves the buffer cursor to the first character of the current line.

Parameters

none

Returns

The *beginning_of_line()* primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

[end_of_line](#)

bookmark_list

```
list bookmark_list()
```

Retrieve existing bookmark list.

Description

The *bookmark_list()* primitive creates a list containing all currently defined bookmarks. For each definition the list shall contain a 4 integer record representing the bookmark as follows:

```
{ bookid, bufnum, line, column }
```

- bookid - Bookmark identifier.
- bufnum - buffer number.
- line - buffer line number.
- column - buffer column number.

If no bookmarks exist then a **NULL** list shall be returned.

Parameters

none

Returns

The *bookmark_list()* primitive returns a list containing the bookmark definitions otherwise NULL if no bookmarks exist.

Portability

n/a

See Also

[delete_bookmark, drop_bookmark, goto_bookmark](#)

delete_bookmark

```
void delete_bookmark(int bookid)
```

Delete a bookmark.

Description

The *delete_bookmark()* primitive deletes the bookmark *bookid*.

Upon successful completion, the **REG_BOOKMARK** event shall be triggered (See: [register_macro](#)).

Parameters

`bookid` Bookmark identifier.

Returns

nothing

Portability

n/a

See Also

[bookmark_list](#), [drop_bookmark](#), [goto_bookmark](#)

down

```
int down([int lines = 1])
```

Move position down one line.

Description

The `down()` primitive moves the cursor down one line to the same column on the next line.

Parameters

`lines` Optional number of lines to move the cursor; may be negative in which case the cursor moves backwards behaving like `up`.

Returns

The `down()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

[up](#), [left](#), [right](#)

drop_bookmark

```
int drop_bookmark( [int bookid],
                   [string yesno],
                   [int bufnum],
                   [int line],
                   [int column],
                   [int local = FALSE])
```

Create or update a bookmark.

Description

The `drop_bookmark()` primitive either creates a new or updates an existing bookmark. A bookmark is a named place holder with a buffer, representing a specific physical location within that buffer.

`bookid` is the unique identifier to be associated with the bookmark; any valid integer may be used as the identifier. The bookmark shall be associated with the buffer `bufnum`, `line` and `column`, if any are omitted the current buffer and location with that shall be used.

Upon there being an existing definition against the specified bookmark identifier, the user shall be prompted as follows, asking whether or not the bookmark should be replaced:

Overwrite existing bookmark [y/n]?

The `yesno` argument if given disables the user prompt. If supplied with either "y" or "yes" the bookmark shall automatically be replaced, otherwise the bookmark is retained without change with the user informed as follows:

Bookmark already exists.

Upon successful completion, the user shall be informed as follows regardless of whether a new or updated definition resulted:

Bookmark dropped.

Parameters

- `bookid` Optional bookmark identifier, if omitted a new unique book mark identifier shall be generated.
- `yesno` Optional string buffer containing the answer to be applied upon the bookmark pre-existing, if given as "y[es]" the bookmark shall be overridden otherwise is shall be related. Otherwise if omitted upon a preexisting bookmark the user shall be prompted.
- `bufnum` Optional buffer number, if omitted the current buffer shall be referenced.
- `line` Optional integer line number within the buffer, if omitted shall default to the top of the buffer (1).
- `column` Optional integer column number within the buffer, if omitted shall default to the left of the buffer (1).
- `local` Reserved for future use.

Returns

The `drop_bookmark()` primitive returns the associated book mark identifier, otherwise 0 on error.

Portability

n/a

TODO

Support buffer local bookmark identifiers; improves Vim compatibility.

See Also

[bookmark_list](#), [delete_bookmark](#), [goto_bookmark](#)

end_of_buffer

```
int end_of_buffer()
```

Move cursor to end of current buffer.

Description

The `end_of_buffer()` primitive moves the buffer cursor to the end of the last line of the current buffer.

Parameters

none

Returns

The `end_of_buffer()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

[top_of_buffer](#), [move_abs](#)

end_of_line

```
int end_of_line()
```

Goto end of line.

Description

The `end_of_line()` primitive moves the buffer cursor to the last character of the current line.

Parameters

none

Returns

The `end_of_line()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

[beginning_of_line](#)

end_of_window

```
int end_of_window()
```

Goto end of the current window.

Description

The *end_of_window()* primitive moves the buffer cursor to the last line of the current window.

Parameters

none

Returns

Returns non-zero if the current cursor position moved, otherwise zero if already positioned at the end of the window.

Portability

n/a

See Also

[top_of_window](#)

goto_line

```
int goto_line([int lineno])
```

Move to a particular line.

Description

The *goto_line()* primitive repositions the cursor to the beginning of the specified line *lineno*.

Parameters

lineno Specifies the line number which to relocate the cursor, if omitted the user is prompted.

Returns

The *goto_line()* primitive returns **true** if successful, otherwise zero or less if unsuccessful.

Portability

n/a

See Also

[goto_old_line](#), [move_abs](#)

goto_old_line

```
int goto_old_line([int oldlineno])
```

Move to line before buffer modification.

Description

The *goto_old_line()* primitive repositions the cursor as close as possible to the beginning of the specified line *oldlineno*, representing the line prior to any buffer modifications.

For each buffer the previous line numbers are retained for any one edit session, that is they are maintained until the buffer is saved, at which point line references are reset to the resulting new image.

This primitive is provided for seeking lines within a buffer that are referred to in an external listing. For example within a previous compiler error report yet since that time inserts and/or line deletes have occurred to the source, yet despite these edits the original line can still be addressed.

Parameters

oldlineno Specifies the old line number which to relocate the cursor, if omitted the user is prompted.

Returns

The *goto_old_line()* primitive returns **true** if successful, otherwise zero or less if unsuccessful.

Portability

n/a

See Also

[goto_line](#), [move_abs](#)

left

```
int left([int columns = 1],  
        [int wrap = TRUE] )
```

Move position left one character.

Description

The *left()* primitive moves the cursor left one column retaining the current line; unless at the beginning of the line, in which case the cursor moves to the end of the previous line.

Parameters

- columns* Optional number of columns to move the cursor; negative in which case the cursor movement is reversed behaving like [right](#).
- wrap* Optional boolean value controlling whether the cursor wraps when positioned at the beginning of line. If **FALSE** line wrapping shall be disabled.

Returns

The *left()* primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

Unlike BRIEF, if the cursor is moved past the beginning of the current line, then the cursor wraps around to the end of the previous line.

wrap is a **GriefEdit** extension.

See Also

[right](#), [up](#), [down](#)

move_abs

```
int move_abs( [int line = -1],  
             [int column = -1],  
             [int bufnum],  
             [int clip = FALSE])
```

Move to an absolute location in the buffer.

Description

The *move_abs()* primitive moves the buffer cursor to an absolute location, with top of the buffer being position (1,1).

If a parameter is 0 or omitted, the corresponding line or column coordinate is unchanged; positive values set the line and/or column.

Parameters

- line* Optional integer specifying the line number, if positive the cursor is set to the specified line.
- column* Optional integer specifying the column number, if positive the cursor is set to the specified column.
- bufnum* Optional buffer number, if specified the associated buffer is affected, otherwise the current buffer.
- clip* Optional int flag, if non-zero the resulting buffer cursor shall be clipped to the buffer size.

Returns

The *move_abs()* primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

bufnum and *clip* are extensions.

See Also

[move_rel](#)

move_rel

```
int move_rel([int lines = 1],
             [int cols = 1] )
```

Move to a relative location in the buffer.

Description

The *move_rel()* primitive moves the buffer cursor to a new position relative to the current position.

If a parameter is 0 or omitted, the corresponding line or column coordinate is unchanged. Positive values move the cursor towards the end of the line and/or column, likewise negative values move towards the beginning of the line and/or column.

Parameters

- `lines` Optional integer specifying the line number, if positive the cursor moves forwards the end of the file, likewise a negative moves to backwards towards the top.
- `cols` Optional integer specifying the column number, if positive the cursor moves forwards the front of the line, likewise a negative moves to backwards towards the beginning.

Returns

The *mov_rel()* primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

[move_abs](#)

next_char

```
int next_char([int characters = 1])
```

Move to the next character.

Description

The *next_char()* primitive moves the current buffer position forward to the next character, wrapping around line ends when encountered.

The primitive is similar to [right](#) except it moves physical characters as opposed to logic characters, as the result tabs and newlines are both treated as one character.

Within navigating binary files newlines are not implied, as such are not counted within the character count.

Parameters

- `characters` Optional number of characters to move forward in the buffer, which if omitted is 1.

Returns

The *next_char()* primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

[prev_char](#), [right](#)

page_down

```
int page_down([int pages = 1])
```

Move position down a page.

Description

The *page_down()* primitive moves the buffer position down or forward one or more pages down, with a page being the current window size in lines.

Parameters

- `pages` If supplied, states the number of pages to move the cursor forward, otherwise the cursor is moved 1 page.

Returns

The `page_down()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

`pages` is a **GriefEdit** extension.

See Also

`page_up`, `down`

page_up

```
int page_up([int pages = 1])
```

Move position up a page.

Description

The `page_up()` primitive moves the buffer position up or backwards one or more pages up, with a page being the current window size in lines.

Parameters

`pages` If supplied, states the number of pages to move the cursor backwards, otherwise the cursor is moved 1 page.

Returns

The `page_up()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

`pages` is a **GriefEdit** extension.

See Also

`page_down`, `up`

parse_filename

```
int parse_filename( string fullname,
                    [string &drive],
                    [string &path],
                    [string &filename],
                    [string &ext]      )
```

Parse a file into its components.

Description

The `parse_filename()` primitive parses and breaks the file name *fullname* into its components.

Note:

Since this primitive is not portable outside of a DOS/Windows environment, its use is not advised.

Parameters

<code>fullname</code>	A string containing the file-name to be parsed.
<code>drive</code>	Optional string variable when supplied to be populated with the drive component, if any.
<code>path</code>	Optional string variable when supplied to be populated with the path component.
<code>filename</code>	Optional string variable when supplied to be populated with the file-name component.
<code>ext</code>	Optional string variable when supplied to be populated with the file extension.

Returns

The `parse_filename()` primitive returns non-zero on success denoted the *fullname* was parsed, otherwise zero was unsuccessful and -1 if an empty *fullname* was supplied.

Portability

Provided for BRIEF compatibility.

See Also

`dirname`, `basename`

prev_char

```
int prev_char([int characters = 1])
```

Move to the previous character.

Description

The `prev_char()` primitive moves the current buffer position backward to the previous character, wrapping around line ends when encountered.

The primitive is similar to `left` except it moves physical characters as opposed to logic characters, as the result tabs and newlines are both treated as one character.

Within navigating binary files newlines are not implied, as such are not counted within the character count.

Parameters

`characters` Optional number of characters to move backward in the buffer, which if omitted is 1.

Returns

The `prev_char()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

`next_char`, `left`

swap_anchor

```
int swap_anchor()
```

Swaps the mark with the current position.

Description

The `swap_anchor()` primitive swaps the current cursor position with the start of the marked region, without changing the mark type.

Parameters

none

Returns

Returns `true` on success, otherwise `false`.

Portability

n/a

See Also

`drop_anchor`, `mark`

top_of_buffer

```
int top_of_buffer()
```

Move cursor to start of current buffer.

Description

The `top_of_buffer()` primitive moves the buffer cursor to the start of the first line of the current buffer; this is equivalent to using `move_abs` as follows.

```
move_abs(1,1);
```

Parameters

none

Returns

The `top_of_buffer()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also[end_of_buffer](#), [inq_position](#), [move_abs](#), [move_rel](#)**top_of_window**

```
int top_of_window()
```

Goto top of the current window.

Description

The *top_of_window()* primitive moves the buffer cursor to the first line of the current window.

Parameters

none

Returns

Returns non-zero if the current cursor position moved, otherwise zero if already positioned at the top of the window.

Portability

n/a

See Also[end_of_buffer](#), [inq_position](#), [move_abs](#), [move_rel](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Process Management Primitives

Summary

Process Management Primitives

Macros

connect	Attach a process to a process.
disconnect	Disconnect a buffer from a process.
dos	Create a sub-shell.
inq_connection	Connection information.
insert_process	Send string to a attached process.
perror	Print error.
send_signal	Send signal to a process buffer.
shell	Spawn a sub-shell process.
sleep	Suspend execution for an interval of time.
sterror	String error.
strsignal	Return string describing signal.
suspend	Suspend current process.
wait	Wait for attached process to terminate.
wait_for	Wait for process output.
Signals	Signals are a limited form of inter-process communication used in Unix, Unix-like, and other POSIX compliant operating systems.

Macros

connect

```
int connect(int mode,
           string shell = NULL,
           string cwd   = NULL )
```

Attach a process to a process.

Description

The `connect()` primitive creates a sub-process and attaches it to the current buffer. The buffer behaviour is similar to that of a virtual terminal with the sub-process standard input and output streams redirected to the buffer. The virtual terminal implementation is system dependent yet most utilise pseudo-devices otherwise pipes.

All text inserted into the buffer is automatically forwarded to the underlying sub-process, insertion methods include `self_insert`, `insert` and `paste`. In addition the primitive `insert_process` allows for explicit text to be forwarded without echo.

Likewise output from the underlying sub-process is sent to the buffer. All output from the sub-process is automatically inserted into the buffer, at the process insertion position; see `inq_process_position` and `set_process_position`.

The `mode` argument specifies the control flags the behaviour of the connection, if omitted defaults to `PF_ECHO`.

By default, the process created is a shell process, and the shell is got from the `SHELL` environment variable. If `shell` is specified, then it is taken as the pathname of a shell to execute.

Once attached `connect()` may be called to change the `mode` flag.

Modes

Control flags

Constant	Value	Description
PF_ECHO	0x0001	Reserved for macro use, denotes all typed key strokes are redirected to the process using a <code>REG_TYPE</code> signal handler (See: <code>register_macro</code>).
PF_NOINSERT	0x0002	Reserved for macro compatibility, disables automatic insertion of sub-process output into the associated buffer; not implemented.
PF_NONINTERACTIVE	0x0004	Non-interactive mode command shell.

Constant	Value	Description
PF_LOCALECHO	0x1000	Local echo mode, characters written to the buffer using either <code>insert</code> or <code>self_insert</code> shall be automatically sent to the process.
PF_OVERWRITE	0x4000	Overwrite process input (CR/LF conversion). Indicates whether output from a process overwrites text in the buffer or inserts and shifts text over as it does so. Needed for effectively allow type-ahead to not be destroyed but allow escape sequences to cause data in the buffer to be overwritten when running termcap oriented programs.
PF_WAITING	0x8000	Waiting for text. Normally when a buffer is created, the output from the subprocess is inserted directly into the buffer. Setting this bit causes the output from the process to be held onto, until the calling macro issues a <code>wait</code> or <code>wait_for</code> .

Parameters

- `mode` Optional integer control flags.
`shell` Optional string containing the SHELL specification is be utilised, if omitted the environment value of SHELL is used.
`cwd` Optional string containing current working directory.

Returns

The `connect()` primitive returns the positive IP identifier associated identifier with the created connection, 0 if the buffer is already connected, otherwise -1 on error.

Portability

n/a

See Also

`insert_process`, `inq_connection`, `disconnect`, `wait`, `shell`, `dos`

disconnect

```
int disconnect()
```

Disconnect a buffer from a process.

Description

The `disconnect()` primitive disconnects the current buffer from any attached process, if any, terminating the subprocess.

Note:

Under Unix the terminated subprocess is sent a SIGTERM followed by a SIGKILL signal.

Parameters

none

Returns

Returns 1 on success otherwise 0.

Portability

n/a

See Also

`connect`, `send_signal`

dos

```
int dos([string cmd],
       [int use_shell],
       [string callback])
```

Create a sub-shell.

Description

The *dos()* primitive executes the specified command *cmd*, if omitted starts an interactive command shell.

This function is provided for compatibility using the [shell](#) interface; see the [shell](#) and the [brief](#) macro module for details.

Parameters

<code>cmd</code>	String containing the command to be executed.
<code>shell</code>	Optional integer stating whether a shell should be utilised, if non-zero use of a shell shall be omitted if feasible.
<code>callback</code>	Optional string containing the name of a macro to be executed on the completion of the task. If stated the command is intended to run as a background task, otherwise in the foreground.

Returns

The *dos()* primitive returns the completion value of the corresponding command.

A negative value denotes an execution failure; on failure *errno* contains a value indicating the type of error that has been detected including.

Constant	Description
<code>E2BIG</code>	Combined Size of environment and argument list is too large.
<code>EACCES</code>	Search permission is denied on a component of the path prefix of filename or the name of a script interpreter.
<code>EACCSE</code>	The file or a script interpreter is not a regular file.
<code>EACCSE</code>	Execute permission is denied.
<code>EIO</code>	An I/O error occurred.
<code>ENAMETOOLONG</code>	Path is too long.
<code>ENOENT</code>	The command or one of its components does not exist.
<code>ENOEXEC</code>	An executable is not in a recognized format, is for the wrong architecture, or has some other format error that means it cannot be executed.
<code>ENOMEM</code>	Insufficient kernel memory was available.

Portability

n/a

See Also

[shell](#)

inq_connection

```
int inq_connection(int cid,
                   int &flags,
                   int &pid )
```

Connection information.

Description

The *inq_connection()* primitive retrieves information regarding the connection *cid*.

Parameters

<code>cid</code>	Integer connection identifier returned from connect.
<code>flags</code>	Integer variable to be populated with the connection flags.
<code>pid</code>	Integer variable to be populated with the connection process identifier.

Returns

nothing

Portability

A [GriefEdit](#) extension.

See Also

[connect](#), [disconnect](#)

insert_process

```
int insert_process(string|int val,
                  [int num = 1])
```

Send string to a attached process.

Description

The *insert_process()* primitive inserts the specified string or integer character value *val* into the process attached to the current buffer. The value shall be inserted *num* times, which if omitted defaults to 1.

Parameters

val String or integer character value to be inserted.

num Option integer number stating the repeat count, if specified then the string is inserted the given number of times. If omitted, it defaults to 1.

Returns

The *insert_process()* primitive returns the number of characters inserted.

Portability

n/a

See Also

[insert](#), [insertf](#), [insert_buffer](#)

perror

```
string perror(      [int errnum = errno],
                  string format,
                  ... )
```

Print error.

Description

The *perror()* primitive shall map the error number specified by *errnum* to a language-dependent error message, which shall be written to the standard error stream as follows:

```
<message> : <error - description>
```

Parameters

errnum Integer value of the error condition to be decoded, if omitted the current system *errno* is decoded.

Returns

The *perror* function returns a string containing the formatted message with a trailing description of the the error value.

Portability

A [GriefEdit](#) extenions.

See Also

[errno](#), [strerror](#)

send_signal

```
int send_signal(int signal)
```

Send signal to a process buffer.

Description

The *send_signal()* primitive send a signal to a process or a group of processes attached to the current buffer. The signal to be sent is specified by *signo* and is either one from the list given in [Signals](#) or 0.

If *signo* is 0 or omitted, then the null signal is sent, error checking is performed but no signal is actually sent.

See the unix *kill* system function is more details.

Parameters

signo Optional integer signal number, if omitted defaults to the *null* signal.

Returns

The `send_signal()` primitive upon successful completion returns 0 and 1 when no buffer is attached. Otherwise -1 shall be returned and `errno` set to indicate the error.

Portability

n/a

See Also

`connect`, `disconnect`

shell

```
int shell( [string cmd],
           [int use_shell],
           [string completion],
           [string stdin],
           [string stdout],
           [string stderr],
           [int mode],
           [string spec] )
```

Spawn a sub-shell process.

Description

The `shell` primitive performs executes a command specified in `cmd` by creating a system dependent shell, and returns after the command has been completed.

The `shell` primitive can be used to create a subshell without exiting the current **GriefEdit** session; placing **GriefEdit** into the background. Without any arguments an interactive shell is created. The user terminates the sub-shell by typing the usual ^D or exit.

```
shell()
```

Alternatively the `shell` primitive can be used to execute an explicit command, as in the following examples.

The following example performs a `uname(1)` command redirecting into a output in a temporary working file for use on completion; the `TRUE` informs **GriefEdit** not to bother repainting the screen.

```
shell("uname 2>&1 >/tmp/uname.tmp", TRUE);
```

The following example performs a `gmake` command redirecting output plus registers the macro `gmake_complete` to be executed on completion.

```
shell("gmake 2>&1 >/tmp/gmake.tmp", TRUE, "gmake_completion");
```

associated completion macro.

```
void
gmake_completion(int status)
{
    message("gmake done, status = %d", status);
}
```

Meta Characters

As the command is passed on to a command processor care should be taken with special characters (including \$, ?, *, [, and ;) since the command wild-card and environment expansion will occur.

Although there is no definite syntax for wildcard operations, common features include:

Wildcard	Description
?	Matches any single character.
*	Matches none or more characters.

Wildcard	Description
[seq]	Matches any one of the characters within the sequence. Sequences generally support ranges; two characters separated by - denote a range. (e.g. [A-Fa-f0-9] is equivalent to [ABCDEFabcdef0123456789].)
[!seq]	Matches any one of the characters not contained within the sequence.

Redirection

In addition input/output redirection is system dependent and/or *shell* command interpreter specific. Despite this fact the following are normally supported across all supported **GriefEdit** targets.

Redirection	Description
>	Writes the command output to a file or a device, such as a printer, instead of the Command Prompt window.
<	Reads the command input from a file, instead of reading input from the keyboard.
>>	Appends the command output to the end of a file without deleting the information that is already in the file.
>&	Writes the output from one handle to the input of another handle; the standard handle assignments are 0=stdin, 1=stdout and 2=stderr.
<&	Reads the input from one handle and writes it to the output of another handle.

Note:

If there is any doubt regarding portability of redirection operators it is advised to use the explicit *stdin*, *stdout* and *stderr* arguments.

Parameters

cmd	Optional string containing the command to be passed to the host environment to be executed by a command processor in an implementation-dependent manner. If omitted a interactive sub-shell is created.
use_shell	Optional boolean value, if true forces the original display and terminal settings to be restored, and than initialised on completion of the sub-process.
completion	Optional string containing the name of macro to be executed on the termination of the sub-process. The completion routine is called with the first parameter set to the return status from the underlying process. Any other positional parameters are shifted up one.
stdin	Option string, specifies the name of the file/device from which standard input shall be source. If omitted the sub-shell standard input remains unchanged.
stdout	Option string, specifies the name of the file/device to which standard output shall be redirected. If omitted the sub-shell standard output remains unchanged.
stderr	Option string, specifies the name of the file/device to which standard error shall be redirected. If omitted the sub-shell standard error remains unchanged.
mode	Optional mode flags, specifies the creation mode to be utilised during stream creation. If omitted 0644 is applied.
spec	Optional string, reserved for future use.

Returns

The *shell* primitive returns the exit status from the executed command (0 .. 256), otherwise -1 on error and sets **errno** to indicate the error.

Constant	Description
E2BIG	Combined Size of environment and argument list is too large.
EACCES	Search permission is denied on a component of the path prefix of filename or the name of a script interpreter.
EACCES	The file or a script interpreter is not a regular file.
EACCES	Execute permission is denied.
EIO	An I/O error occurred.
ENAMETOOLONG	Path is too long.
ENOENT	The command or one of its components does not exist.
ENOEXEC	An executable is not in a recognized format, is for the wrong architecture, or has some other format error that means it cannot be executed.
ENOMEM	Insufficient kernel memory was available.

Portability

A arguments differ from the original implementation.

See Also

[dos](#), [connect](#)

sleep

```
void sleep([int seconds = 1],  
          [int milliseconds = 0])
```

Suspend execution for an interval of time.

Description

The *sleep()* primitive causes the caller to be suspended from execution until either the specified interval specified by *seconds* and *milliseconds* has elapsed.

Note:

The suspension time may be longer than requested due to the scheduling of other activity by the system.

Parameters

seconds Optional positive integer stating the time-out interval seconds component, if omitted defaults to 1 second.
milliseconds Option positive integer stating the time-out interval milliseconds component.

Returns

nothing

Portability

n/a

See Also

[inq_clock](#), [time](#)

strerror

```
string strerror( [int errnum = errno],  
                [string &manifest],  
                [int multi = FALSE] )
```

String error.

Description

The *strerror()* primitive maps the error number in *errnum* to a locale-dependent error message and returns a string containing the mapped condition.

Parameters

errnum Integer value of the error condition to be decoded, if omitted the current system *errno* is decoded.
manifest Optional string variable which is specified shall be populated with the signal manifest.

Returns

The *strerror* function returns a string describing the error value, otherwise an empty string if undefined.

Portability

The *manifest* and *multi* options are **GriefEdit** extensions.

See Also

[errno](#), [perror](#)

strsignal

```
int strsignal( int signo,  
               [string &manifest],  
               [int multi = FALSE])
```

Return string describing signal.

Description

The *strsignal()* primitive returns a string describing the signal number passed in the argument *signo*.

Parameters

signo Integer value of the signal number to be decoded.

manifest Optional string variable which is specified shall be populated with the signal manifest.

Returns

The *strsignal()* primitive returns the appropriate description string, or an unknown signal message if the signal number is invalid.

Portability

A **GriefEdit** extension.

See Also

[strerror](#)

suspend

```
void suspend()
```

Suspend current process.

Description

The *suspend()* primitive pretends user typed *ctrl-z* by sending a **SIGTSTP** to the controlled processing, effectively suspending **GriefEdit** by placing it in the background.

Note:

This primitive shall only function on systems which support job control, for example *unix*.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[process](#), [exit](#)

wait

```
int wait([int &status])
```

Wait for attached process to terminate.

Description

The *wait()* primitive suspends execution until for the process attached to the current buffer terminates, returning status information for the terminated child, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

This primitive may be aborted by pressing a space.

Parameters

status Optional integer variable populated with the process status.

Returns

The *wait()* primitive returns 0 if the process has completed, -2 if the user aborted, otherwise -1 if no process was attached.

Portability

n/a

See Also

[wait_for](#), [inq_process_position](#), [connect](#)

wait_for

```
int wait_for(          [int timeout],
                  list|string pattern,
                  [int flags = 0])
```

Wait for process output.

Description

The `wait_for()` primitive waits for a specific string of characters to be output by the current buffers attached sub-process for up to the specified time `timeout`.

The character sequence is in the form of regular expression `pattern` using the search flags `flags`. The pattern is either a single string expression or a list of string expressions. When a list is given, then a parallel match is performed as each character is read from the buffer.

Parameters

- `timeout` Optional integer timeout in seconds, if omitted or zero the primitive blocks indefinitely until the sequence is encountered or the sub-process terminates.
- `pattern` Character sequences to be matched, as either a single string containing a regular expression or a list of string expressions.
- `flags` Optional integer stating the search flags to be applied (See: [re_search](#)).

Returns

The `wait_for()` primitive returns a non-negative return on a success match, otherwise -1 if there is no process currently attached to the current buffer, or the user interrupted.

On a single string expression `wait_for` returns 1, otherwise upon a list of string expressions returns the matching index.

Portability

n/a

See Also

[wait](#), [inq_process_position](#), [connect](#)

Signals

Signals are a limited form of inter-process communication used in Unix, Unix-like, and other POSIX compliant operating systems. A signal is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred.

The following manifest constants are the general set of signals which may be supported representing their positive signal number, which if not supported by the underlying operating system shall have an assigned value of -1.

Constant	Description
SIGHUP	Hangup (POSIX).
SIGINT	Interrupt (ANSI).
SIGQUIT	Quit (POSIX).
SIGILL	Illegal instruction (ANSI).
SIGTRAP	Trace trap (POSIX).
SIGABRT	Abort (ANSI).
SIGIOT	IOT trap (4.2 BSD).
SIGBUS	BUS error (4.2 BSD).
SIGFPE	Floating-point exception (ANSI).
SIGKILL	Kill, unblockable (POSIX).
SIGUSR1	User-defined signal 1 (POSIX).
SIGSEGV	Segmentation violation (ANSI).
SIGUSR2	User-defined signal 2 (POSIX).
SIGPIPE	Broken pipe (POSIX).
SIGALRM	Alarm clock (POSIX).
SIGTERM	Termination (ANSI).
SIGSTKFLT	Stack fault.
SIGCHLD	Child status has changed (POSIX).
SIGCLD	Same as SIGCHLD (System V).
SIGCONT	Continue (POSIX).
SIGSTOP	Stop, unblockable (POSIX).
SIGTSTP	Keyboard stop (POSIX).
SIGTTIN	Background read from tty (POSIX).
SIGTTOU	Background write to tty (POSIX).

Constant	Description
SIGURG	Urgent condition on socket (4.2 BSD).
SIGXCPU	CPU limit exceeded (4.2 BSD).
SIGXFSZ	File size limit exceeded (4.2 BSD).
SIGVTALRM	Virtual alarm clock (4.2 BSD).
SIGPROF	Profiling alarm clock (4.2 BSD).
SIGWINCH	Window size change (4.3 BSD, Sun).
SIGPOLL	Pollable event occurred (System V).
SIGIO	I/O now possible (4.2 BSD).
SIGPWR	Power failure restart (System V).
SIGSYS	Bad system call.
SIGUNKNOWN	Unknown error

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Scrap Primitives

Summary

Scrap Primitives

Macros

<code>copy</code>	Copy marked area to scrap.
<code>cut</code>	Cut marked area to scrap.
<code>delete_block</code>	Deleted selected region.
<code>drop_anchor</code>	Start marking a selection.
<code>end_anchor</code>	Set the end of the anchor.
<code>get_region</code>	Retrieve marked region content.
<code>inq_mark_size</code>	Retrieve size of marked region.
<code>inq_marked</code>	Determine the current marked region.
<code>inq_marked_size</code>	Size the marked region.
<code>inq_scrap</code>	Obtain the scrap buffer identifier.
<code>mark</code>	Toggle the anchor status.
<code>paste</code>	Insert scrap buffer at cursor location.
<code>raise_anchor</code>	Raise the last dropped mark.
<code>redo</code>	Redo an undo operation.
<code>set_scrap_info</code>	Set the scrap buffer details.
<code>transfer</code>	Buffer to buffer transfer.
<code>write_block</code>	Write selected region.

Macros

copy

```
int copy([int append = FALSE],  
       [int keep = FALSE] )
```

Copy marked area to scrap.

Description

The `copy()` primitive copies the content of the currently marked region to the scrap buffer, and releases the marked region on completion.

Parameters

- `append` Optional integer, if non-zero the copied region shall be appended to the scrap instead of replacing the scrap content.
- `keep` Optional integer, if non-zero the marked region is retained otherwise the anchor is released.

Returns

The `copy()` primitive returns 1 on success, otherwise 0 on error.

Portability

n/a

See Also

[cut](#), [delete_block](#), [inq_scrap](#), [paste](#), [transfer](#)

cut

```
int cut([int append = FALSE],  
       [int syscopy = FALSE])
```

Cut marked area to scrap.

Description

The `cut()` primitive moves the content of the currently marked region to the scrap buffer, and releases the marked region on completion.

Parameters

`append` Optional integer, if non-zero the copied region shall be appended to the scrap instead of replacing the scrap content.

`keep` Optional integer, if non-zero the marked region is retained otherwise the anchor is released.

Returns

The `cut()` primitive returns 1 on success, otherwise 0 on error.

Portability

n/a

See Also

[paste](#), [delete_block](#), [inq_scrap](#), [paste](#), [transfer](#)

delete_block

```
int delete_block()
```

Deleted selected region.

Description

The `delete_block()` primitive deleted the current marked block, leaving the cursor position on the last line of the marked region.

The characters included in the mark depend on the its type, and once deleted the mark is removed.

On completion the following is echoed on the command prompt.

Block deleted.

In the event of no active region the following is echoed.

No marked block.

Parameters

none

Returns

The `delete_block()` primitive returns one if the block was delete, otherwise zero.

See Also

[cut](#), [copy](#), [delete_char](#)

drop_anchor

```
int drop_anchor([int type = MK_NORMAL])
```

Start marking a selection.

Description

The `drop_anchor()` primitive starts marking a block of the specified `type` at the current position in the current buffer.

Marked regions are highlighted to stand out on the screen either in a different colour or in reverse video dependent on the available display capacities.

The following are the available region types.

Value	Constant	Description
1	MK_NORMAL	A normal mark.
2	MK_COLUMN	A column mark.
3	MK_LINE	A line mark.
4	MK_NONINC	A non-inclusive mark.

- MK_NORMAL** A normal mark is a region which encompasses from the place where the anchor was dropped up to and including the current cursor position.
- MK_COLUMN** A column mark allows a rectangular section of the buffer to be marked, highlighting the inclusive columns between the left and right boundaries.
- MK_LINE** A line mark selects entire lines, and allows for easy movement of text from one part of a buffer to another.
- MK_NONINC** A non-inclusive mark, like a normal, is a region which encompasses from the place where the anchor was dropped up to and but *does not* include the current cursor position.

Regions are nestable, in that a *drop_anchor* may be issued without an intervening *raise_anchor*. Each mark is pushed into a LIFO (last-in, first-out) stack, allowing multiple marks to exist simultaneously; each mark must however be eventually raised. The most recent mark shall be the one displayed by the buffer.

The current active marked region can queried using [inq_marked](#).

The marked region can be cleared by calling [raise_anchor](#) or performing a high level [copy](#) or [cut](#) buffer operations plus a numebr of the lower level functions, for example [delete_block](#).

Parameters

`type` Optional anchor type to be dropped Otherwise if omitted a **MK_NORMAL** anchor shall created.

Returns

The *drop_anchor* returns 1 if successful, otherwise 0 on error.

Portability

n/a

See Also

[mark](#), [raise_anchor](#)

end_anchor

```
int end_anchor([int line],  
              [int column])
```

Set the end of the anchor.

Description

The *end_anchor()* primitive sets the end of the current marked region without the need to move the cursor.

Parameters

`line` Optional new line, if omitted the line shall be unchanged.

`column` Optional new column, if omitted the column shall be unchanged.

Returns

The *end_anchor* returns 1 if successful, otherwise 0 on error.

Portability

n/a

See Also

[mark](#), [drop_anchor](#)

get_region

```
int get_region([int bufnum])
```

Retrieve marked region content.

Description

The *get_region()* primitive is similar to *copy* yet is copies the content of the current marked region and returns the result as a string.

Parameters

`bufnum` Optional window identifier, if omitted the current buffer is referenced.

Returns

The *get_region()* primitive returns the content of the current marked region as a string, otherwise an empty string.

Portability

n/a

See Also[cut](#), [paste](#), [copy](#), [inq_marked](#)**inq_mark_size**

```
int inq_mark_size()
```

Retrieve size of marked region.

Description

The *inq_mark_size()* primitive determines the number of characters what are in the currently marked region; this represents the the length of a string which would be necessary to hold its content, see [get_region](#).

Parameters

none

Returns

The *inq_mark_size()* primitive returns the number of characters in the currently marked region, otherwise 0 if there is no current region.

Portability

A **GriefEdit** extension.

See Also[mark](#), [drop_anchor](#), [inq_marked](#)**inq_marked**

```
int inq_marked( [int &start_line],
                 [int &start_col],
                 [int &end_line],
                 [int &end_col] )
```

Determine the current marked region.

Description

The *inq_marked()* primitive retrieves the current mark type and the associated coordinates of the marked area.

Region Types

Value	Constant	Description
0	MK_NONE	No mark is set
1	MK_NORMAL	Normal mark
2	MK_COLUMN	Column mark
3	MK_LINE	Line mark
4	MK_NONINC	Non-inclusive

Parameters

- `start_line` If specified the integer parameter is set to the line number at the top of the marked region.
- `start_col` If specified the integer parameter is set to the column number at the beginning of the marked region.
- `end_line` If specified the integer parameter is set to the line number marking the bottom of the marked region.
- `end_col` If specified the integer parameter is set to the column number at the end of the marked region.

Returns

The *inq_marked()* primitive returns the current region type, otherwise 0 if no mark is active.

Portability

n/a

See Also[mark](#)

inq_marked_size

```
int inq_marked_size()
```

Size the marked region.

Description

The *inq_marked_size()* primitive is reserved for future compatibility.

The *inq_marked_size()* primitive determines the number of characters contained within the current marked region.

Parameters**Returns**

Returns the character count within the marked region.

Portability

Function is currently a no-op, returning -1.

See Also

[inq_marked](#)

inq_scrap

```
int inq_scrap([int &last],  
             [int &type] )
```

Obtain the scrap buffer identifier.

Description

The *inq_scrap()* primitive retrieves the buffer identifier of the current scrap buffer.

Parameters

last Not used; provided for BRIEF compatibility. If **false** the last newline in the scrap is not considered part of the scrap, otherwise **true** it is considered part of the scrap.

type Optional integer variable reference, if stated shall be populated with the buffers mark type.

Returns

The *inq_scrap()* primitive returns the scrap buffer identifier.

Portability

n/a

See Also

[copy](#), [cut](#), [paste](#), [set_scrap_info](#)

mark

```
int mark([int type = MK_NORMAL])
```

Toggle the anchor status.

Description

The *mark()* primitive toggles the status of the marked region. If the anchor is currently dropped it shall be raised, and if raised it shall be dropped.

To be independent of the current state macros should generally utilise [drop_anchor](#) and [raise_anchor](#), with mark reserved for keyboard bindings.

Parameters

type Optional anchor type to be dropped, if omitted a MK_NORMAL (1) mark shall be dropped.

Value	Constant	Description
1	MK_NORMAL	A normal mark.
2	MK_COLUMN	A column mark.
3	MK_LINE	A line mark.
4	MK_NONINC	A non-inclusive mark.

Returns

The *mark* returns 1 if successful, otherwise 0 on error.

Portability

Unlike BRIEF the anchor status is always toggled.

BRIEF logic was equivalent to the following. If *type* was stated and an anchor of a different type current exists, the anchor is converted.

```
if (! inq_marked() || type) {
    drop_anchor(type);
} else {
    raise_anchor();
}
```

See Also

[inq_marked](#), [raise_anchor](#), [drop_anchor](#)

paste

```
int paste([int syspaste = FALSE])
```

Insert scrap buffer at cursor location.

Description

The *paste()* primitive insert the contents of the scrap buffer at the current cursor location.

The buffer is pasted according to the type of the mark which created it:

- Normal or non-exclusive mark, the scrape is inserted before the current position.
- Line mark, the scrape is inserted before the current line.

Parameters

syspaste Optional boolean flag, is specified and is non-zero, then the text contents of the operating system buffer shall (e.g. clipboard under WIN32) shall be inserted into the current buffer where the cursor is located, if the feature is supported/available.

Returns

The *paste()* primitive returns 1 on success, otherwise 0 on failure.

Portability

n/a

See Also

[cut](#), [copy](#), [inq_scrap](#), [set_scrap_info](#)

raise_anchor

```
int raise_anchor()
```

Raise the last dropped mark.

Description

The *raise_anchor()* primitive raises the last anchor mark that was dropped.

If no mark is dropped, this primitive has no effect.

Parameters

none

Returns

The *raise_anchor* returns 1 if successful, otherwise 0 on error.

Portability

n/a

See Also

[mark](#), [drop_anchor](#)

redo

```
void redo()
```

Redo an undo operation.

Description

The *redo()* primitive redoes the last undone operation.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[undo](#)

set_scrap_info

```
void set_scrap_info( [int last],
                     [int type],
                     [int bufnum])
```

Set the scrap buffer details.

Description

The *set_scrap_info()* primitive specifies the mark type and newline handling to be applied against the scrap buffer.

Parameters

last Not used; provided for BRIEF compatibility. If **false** the last newline in the scrap is not considered part of the scrap, otherwise **true** it is considered part of the scrap.
type Optional integer value, specifies the mark-type to be applied to the scrap buffer.
bufnum Optional integer buffer identifier, if stated changes the current scrap buffer to the specified buffer. The resulting buffer shall be marked with the buffer flag **BF_SCRAPBUF**.

Returns

The *set_scrap_info()* primitive returns 0 on success, otherwise -1 on error, for example in invalid buffer.

Portability

n/a

See Also

[copy](#), [cut](#), [paste](#), [inq_scrap](#)

transfer

```
int transfer( int bufnum,
              int sline,
              [int scolumn],
              int eline,
              [int ecolumn] )
```

Buffer to buffer transfer.

Description

The *transfer()* primitive moves the specified region from the buffer *bufnum* into the current buffer.

All characters in the source buffer between the start and end positions inclusive are included in the block.

The primitive has two forms determined by the number of coordinates arguments being either four or two, as followings:

- start line/column *sline*, *scol* and end line/column *eline* and *ecol*.
- start line *sline* and end line *eline*.

Unlike the original BRIEF implementation, this version is more forgiving and is provided for compatibility; it is usually far easier to use the region macros.

Parameters

bufnum Buffer identifier of the source.
 sline Starting line.
 scolumn Starting column.
 eline Ending line.
 ecolumn Ending column.

Returns

The *transfer()* primitive returns non-zero on success, otherwise zero or less on error.

Portability

The short form of *transfer* is an extension yet less error prone and more convenient interface.

See Also

[copy](#), [cut](#), [paste](#)

write_block

```
int write_block( [string filename],
                 [int append = FALSE],
                 [int keep = FALSE],
                 [int pause = TRUE] )
```

Write selected region.

Description

The *write_block()* primitive writes out the currently marked region to the file *filename*. If *filename* is not specified, then it is prompted for as follows

Write marked area as:

The characters included in the mark depend on the its type, and once written the mark is removed unless *keep* is specified.

Writing out a marked region does not affect the backup flag or the undo information flag; see [undo](#) and [set_backup](#).

On completion the following is echoed on the command prompt.

Write successful.

In the event of no active region the following is echoed.

No marked block.

File Name

Several special leading characters within the stated filename act as modifiers on the output mode of the file.

' | ' data is written to a pipe instead of a file. The string content after the | is passed as an argument to `popen()`.
 '>', '>>' data shall be appended to the specified file following the >; same effect as stated *append*.

Parameters

filename Optional string value containing the path of the destination filename. If omitted the user shall be prompted.
 append Optional boolean value, if true the block is appended to the end of the file; otherwise the file content is replaced.
 keep Optional boolean value, if true the marked region is retainined, otherwise on completion the region is cleared.
 pause Optional boolean value control pipe completion handling. During pipe operations the command may destroy the screen. If omitted or is non-zero then the user is prompted to hit <Enter> before continuing.

Returns

The *write_block()* primitive returns one on success, otherwise zero on error.

Portability

The *filename* and *append* options are **GriefEdit** extensions to BRIEF.

See Also

`delete_block`, `write_buffer`, `undo`, `set_backup`

`$Id:` \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Screen Primitives

Summary

Screen Primitives

Macros

borders	Set window border status.
color	Set the basic colors.
copy_screen	Copy the current screen.
cursor	Control cursor display.
display_mode	Set/retrieve display control flags.
display_windows	Control window display.
echo_line	Set echo line flags.
ega	Terminal line display.
get_color	Retrieve screen colors.
inq_borders	Retrieve the border status.
inq_cmd_line	Retrieve the command line message.
inq_color	Retrieve the basic colors.
inq_command	Retrieve name of last keyboard command.
inq_display_mode	Inquire display control flags.
inq_echo_format	Retrieve the echo-line format.
inq_echo_line	Retrieve the echo-line flags.
inq_font	Inquire the current window fonts.
inq_line_col	Retrieve the echo_line content.
inq_message	Retrieve the prompt line.
inq_prompt	Retrieve the prompt status.
inq_screen_size	Determine screen dimensions.
inq_window_color	Retrieve the window attribute.
inq_window_priority	Retrieve the windows display priority.
inq_window_size	Retrieve the window size.
redraw	Redraw screen.
refresh	Update the display.
screen_dump	Dump an image of the screen.
set_color	Set screen colors.
set_echo_format	Set the echo line format.
set_window_priority	Set the window display priority.
use_tab_char	Configure use of hard/soft tabs.
view_screen	View the content of underlying screen.
window_color	Set the window attribute.

Macros

borders

```
int borders([int borders])
```

Set window border status.

Description

The *borders()* primitive either sets or toggles the tiled window border status.

Disabling borders can improve display performance on slow systems, yet shall disable scroll bars, title bars and may make working with multiple windows difficult.

Parameters

borders An optional integer stated on (1) or off (0). If omitted, the current value is toggled. A value of -1, acts the same as *inq_borders()* only returning the current status without effecting any change.

Returns

An integer boolean value representing the previous border state.

Portability

n/a

See Also

inq_borders, *_chg_properties*

color

```
int color( [int background],
           [int normal],
           [int selected],
           [int message],
           [int error],
           [int hilite],
           [int hilite_fg],
           ... )
```

Set the basic colors.

Description

The *color()* primitive configures the standard color attributes. *color* is only intended for use with the basic color set and is provided only for backward compatibility.

If *background* is omitted, all colors shall be prompted. Colors are numeric values in the range *0..15* or their symbolic name as follows:

Basic Colors

Value	Name	Value	Name
0	Black	8	Dark-grey
1	Blue	9	Bright-blue
2	Green	10	Bright-green
3	Cyan	11	Bright-cyan
4	Red	12	Bright-red
5	Magenta	13	Bright-magenta
6	Brown	14	Yellow
7	White	15	bright-white

Additional colors maybe modified, yet would advice only the use of symbolic *color* names and not *color* values to avoid issues when dealing with terminals which support greater then 16 colors.

The alternative interfaces of [set_color](#) and [set_color_pair](#) are the preferred interfaces for new macros.

Parameters

- background* The background *color*.
- normal* The normal text *color*.
- selected* Highlighting *color* for the selected window title.
- message* The *color* for normal messages.
- error* The *color* for error messages.
- hilite* Color of marked areas. The value either states both the foreground and background of the marked areas as high nibble is background and the low is foreground. Unless *hilite_fg* is stated in which case only the foreground.
- hilite_fg* Foreground *color* of marked areas.

Returns

The *color()* primitive returns 1 on success, otherwise 0 if one or more of the colors is invalid.

Example

Set the background *color* to blue and the foreground color to white, and active window title to bright cyan.

```
color(1, 7, 11);
```

Portability

Colors definitions beyond *frame* are system dependent.

See Also

[get_color](#), [get_color_pair](#)

copy_screen

```
void copy_screen()
```

Copy the current screen.

Description

The `copy_screen()` primitive exports an image of the current editor windows to the current buffer.

Parameters

none

Returns

nothing

Portability

A **GriefEdit** extension; this primitive is subject for removal, and has been removed from recent CrispEdit™ releases.

See Also

[transfer](#)

cursor

```
int cursor(int state)
```

Control cursor display.

Description

The `cursor()` primitive controls whether the cursor is visible, by default the cursor is enabled.

The cursor visibility is set to `state` with a non-zero value enabling and a zero value disabling, if omitted then the current value is toggled.

Parameters

`state` Optional boolean cursor status, if omitted the current status is toggled.

Returns

The `cursor()` primitive returns 0 on success, otherwise non-zero.

Portability

A **GriefEdit** extension.

See Also

[inq_screen_size](#), [borders](#)

display_mode

```
int display_mode( [int or_mask|string set-list],
                  [int and_mask|string clear-list],
                  [int scroll_cols],
                  [int scroll_rows],
                  [int visible_cols],
                  [int visible_rows],
                  [int number_cols]
                )
```

Set/retrieve display control flags.

Description

The `display_mode()` primitive controls primary features of the display interface. Features are either stated using their manifest constant or in the case of values an explicit parameters

If specified one or more flags shall be cleared using the `and_mask`, in addition one or more flags are set using the `or_mask`.

Constant	Name	Description
DC_WINDOW	window	Running under a windowing system (ro).
DC_MOUSE	mouse	Mouse enabled/available (ro).
DC_READONLY	readonly	Read-only mode (ro).
DC_CHARMODE	charmode	Character-mode with basic GUI features (r)
DC_SHADOW	shadow	Display shadow around popups.
DC_SHADOW_SHOWTHRU	showthru	Show-thru shadow around popups.
DC_STATUSLINE	statusline	Status line.
DC_UNICODE	unicode	UNICODE character encoding available (ro)
DC_ASCIIONLY	asciionly	ASCII only characters within UI/dialogs.

Constant	Name	Description
DC_ROSUFFIX	rosuffix	Read-only suffix on titles.
DC_MODSUFFIX	modsuffix	Modified suffix.
DC_EOF_DISPLAY	eof_display	Show <EOF> marker.
DC_TILDE_DISPLAY	tilde_display	Show ~ marker.
DC_EOL_HILITE	eol_hilite	Limit hilites to EOL.
DC_HIMODIFIED	himodified	Hilite modified lines.
DC_HIADDITIONAL	hiadditional	Hilite additional lines.

Note:

Items marked as (ro) are Read-Only with any specified changes against the attribute shall be quietly ignored.

The optional parameters *scroll_cols* and *scroll_rows* define the screen distance each scroll operation shall employ. Values greater than one result in scroll jumps, which may be desired on slower terminals.

The optional parameters *visual_cols* and *visual_rows* define the smallest display arena which shall be permitted, with *number_cols* defining the number line arena width.

Parameters

set_mask	Optional mask of flags to set. May either be an integer of AND'ed together flag constants, or alternatively a string of comma separated flag names.
clear_mask	Optional mask of flags to clear. May either be an integer of AND'ed together flag constants, or alternatively a string of comma separated flag names.
scroll_cols	Optional integer value, if stated sets the number of screen columns scroll operations shall employ. A value of zero shall clear the scroll override, defaulting the value to 1. Upon a negative value, the current override shall be returned.
scroll_rows	Optional integer value, if stated sets the number of screen rows scroll operations shall employ. A value of zero shall clear the scroll override, defaulting the value to 1. Upon a negative value, the current override shall be returned.
visible_cols	Optional integer value, if stated sets the lower bounds of the display arena width. Upon a negative value, the current override shall be returned.
visible_rows	Optional integer value, if stated sets the lower bounds of the display arena height. Upon a negative value, the current override shall be returned.
number_cols	Optional integer value, if stated as a positive integer sets the width of the number-line column within windows, disabling dynamic width selection. A value of zero shall clear the width override, enabling the default dynamic width based upon the buffer length. Upon a negative value, the current override shall be returned.

Returns

The *display_mode()* primitive by default returns the previous value of the display control flags prior to any modifications.

If one of the integer parameters is a negative value (e.g. -1) then *display_mode* returns the current value of the associated parameter. If more than one is negative, then the value of last shall be returned.

Portability

The string mask variants and *set* parameter are GRIEF extensions.

Many of the flags are GRIEF specific; CRISP™ has a similar primitive yet as the two were developed independently features differ.

See Also

[inq_display_mode](#), [set_font](#), [inq_font](#)

display_windows

```
int display_windows([int mode])
```

Control window display.

Description

The *display_window()* primitive control the state of the display driver.

This primitive should be called prior to the creation of any windows using [create_tiled_window](#) enabling the display driver. If any tiled windows exist when enabled the following error shall be echoed, denoting incorrect system initialisation.

```
display_window: overlapping window exist
```

The original functionality was intended to initialise a set of tiled windows, between two successive calls; firstly with disable (0) which cleared all windows and secondary on the completion of the window creation with enable (1). This implementation only obeys the first enable command, with any disable requests silently ignored.

Parameters

`mode` Optional integer, if specified states the new display mode, otherwise if omitted the current display mode is toggled.

Returns

The `display_window()` primitive returns the previous status.

Portability

Unlike BRIEF existing tiled windows are not destroyed upon the initial mode change.

See Also

[create_tiled_window](#)

echo_line

```
int echo_line([int flags])
```

Set echo line flags.

Description

The `echo_line()` primitive controls the fields which are to be visible within the status area.

The status area layout can be defined by one of two means. The `echo_line` specification controls the elements visible using a fixed layout whereas `set_echo_format` allows a user defined format specification.

The default echo_line configuration is.

```
E_CHARVALUE | E_VIRTUAL | E_LINE | E_COL | \
E_CURSOR | E_REMEMBER | E_TIME
```

Parameters

`flags` Optional integer flags, one or more of the following flags OR'ed together control the components to be reported against each attribute. If omitted then only the current flag values are returned without any change.

Flags

Constant	Order	Definition
E_CHARVALUE	1	Character value.
E_VIRTUAL	2	Virtual character indicator.
E_LINE	3	Line: field.
E_COL	4	Col: field.
E_PERCENT	5	nn%
E_CURSOR	7	Insert/overstrike status (OV or blank).
E_REMEMBER	6	Remember status (RE and PA).
E_TIME	8	HH:MM a/pm
E_TIME24	8	Time in 24hour form (HH:MM).
E_FORMAT	n/a	Format override active.
E_FROZEN	n/a	Echo line is frozen, ie not updated.

When an status area user specified format is active then the **E_FORMAT** flag is enabled (See: [set_echo_format](#)). Setting a format also sets **E_FORMAT** similarly clearing the format clears **E_FORMAT**.

The **E_FROZEN** flag disables status area updates whilst set. **E_FROZEN** can be utilised by macros to reduce screen updates during buffer operations.

Active Character

The **E_CHARVALUE** element identifies the character under the cursor. Normal printable characters are enclosed within a set of square brackets, with non-printable characters represented by their hexadecimal value. When the

cursor is positioned past the end of the current line, *EOL* is displayed, and when past the end of file, *EOF* is displayed.

Virtual Character

The **E_VIRTUAL** element indicates whether the current character is either a physical or virtual character in the case of tabs, EOL and EOF conditions.

The virtual character status is represented by one of the following otherwise blank if a normal character.

- ✗ Virtual space, for example logical space created as the result of tab expansion.
- ✗ End of line.
- ✗ Position is past the end-of-line.
- ✗ Is a Unicode encoded character.
- ✗ Is an illegal Unicode character code.

Buffer Coordinates

The **E_LINE** and **E_COL** elements represents the line (row) and column where the cursor is located. Unless invoked with restore enabled, when GRIEF is started the cursor is located at top of the current buffer, being line one(1) and column one(1).

Cursor Mode

The **E_CURSOR** element represents the optional cursor mode.

On systems which have means of controlling the cursor, the current insert/overstrike mode is represented by the cursor shape; when overstrike mode a large/block cursor is used wheras insert mode shall utilise a small/underline cursor.

Otherwise on systems without cursor control a mode indicator shall be displayed, *OV* when overstrike is active otherwise blank when in insert mode.

Remember Status

The **E_REMEMBER** element represents the remember status.

When macro recording is active or paused, *RE* and *PA* respectively are displayed.

Time

Last item in the status area is the time, which is displayed in hours and minutes, with a colon as a separator either using a 12-hour format **E_TIME** or a 24-hour format **E_TIME24**.

Returns

The *echo_line()* primitive returns the previous flags value.

Portability

n/a

See Also

[inq_line_col](#), [inq_prompt](#), [set_echo_format](#), [inq_echo_format](#)

ega

```
int ega(int lines)
```

Terminal line display.

Description

The *ega()* primitive attempts to configure the console size to the specified number of *lines* with an implied width of 80 columns on supporting terminals.

Possible screen dimensions include.

- 60 x 80
- 50 x 80
- 43 x 80
- 25 x 80

Parameters

lines Optional integer specifying the required number of console lines. Under WIN32 if -1 the console size shall toggled bewteen minimized and maximised. If omitted, then only the current state is returned.

Notes!

When running within a windows console, before Windows 7 one could press <Alt+Enter> to run the application in full screen. As of Windows 7 this functionality is no longer available resulting in the following.

This system does not support full screen mode

The `ega()` primitive can be used to emulate this functionality, using the special -1 flag, which shall toggle between a minimized or maximised sized console.

```
ega(-1)
```

Returns

The `ega()` primitive returns the previous status, otherwise -1 upon an error.

Portability

A **GriefEdit** extension.

See Also

[set_term_feature](#)

get_color

```
list get_color([int flags = 0])
```

Retrieve screen colors.

Description

The `get_color()` primitive retrieves the current display color scheme as a list of strings, each string containing the specification of an individual color attribute.

The reported specifications following the form based upon the optional `flags`.

[<id>,] [<flags>,] [<name>=<spec>

`id` Numeric identifier of the attribute.

`flags` Control flags.

`name` Attribute name.

`spec` Color specification.

Color Specification

Each color specification follows the following form.

```
attribute=
  sticky@<attribute>|none
  | link@<attribute>|none
  | <foreground-color>[,<background-color>][|font][:style ...]
  | clear
```

Where colors take one of the following forms, see [set_color](#) for a list of the reported attributes.

```
color-name|none
foreground|fg|background|bg|dynamic_fg|dynamic_bg
decimal (xx), octal (0xxx) or hex (0x###)
#RRGGBB
color[#]ddd
```

style specifications are in the form :<style> [,<style> ...]

```
bold
inverse
underline
blink
italic
reverse
standout
dim
undercurl
```

Parameters

`flags` Optional integer flags, one or more of the following flags OR'ed together control the components to be reported against each attribute.

Flags

Constant	Definition
<code>COLORGET_FVALUE</code>	Attribute numeric value/identifier.
<code>COLORGET_FFLAGS</code>	Type flags.
<code>COLORGET_FNAME</code>	Name.

Returns

Returns a list of colors associated with the color attributes.

If `flags` was specified then the color attribute flags are returned in a list, instead of the color names.

Example

Example list content using `COLORGET_FNAME`

```
"background=none"
"normal=none"
"select=light-cyan"
"message=light-green"
"error=light-red"
"hilite=red"
"hilite_fg=light-white"
"frame=white"
"inscursor=black"
"ovcursor=black"
"shadow=clear"
"prompt=light-magenta,dynamic-bg:link@message"
"question=clear:link@message"
"echo_line=yellow,dynamic-bg"
"statuscolumn=clear:link@message"
"linenocolumn=clear:link@message"
"nonbuffer=brown,dynamic-bg:link@message"
:
:
```

Portability

A **GriefEdit** extension.

See Also

`color`, `inq_color`, `set_color`

inq_borders

```
int inq_borders()
```

Retrieve the border status.

Description

The `inq_borders()` primitive retrieves the border status which controls whether or not tiled windows are displayed with borders.

Disabling borders can improve display performance on slow systems, yet shall disable scroll bars, title bars and may make working with multiple windows difficult.

Parameters

none

Returns

Returns non-zero if windows borders are enabled, otherwise zero if disabled.

Portability

n/a

See Also

[borders](#), [color_index](#)

inq_cmd_line

```
string inq_cmd_line()
```

Retrieve the command line message.

Description

The *inq_cmd_line()* primitive returns the current prompt response displayed on the message line.

Parameters

none

Returns

The *inq_cmd_line* returns a string containing the current prompt.

Portability

n/a

See Also

[inq_message](#)

inq_color

```
string inq_color( [int &background],
                  [int &normal],
                  [int &selected],
                  [int &messages],
                  [int &errors],
                  [int &hilite],
                  [int &hilite_fg] )
```

Retrieve the basic colors.

Description

The *inq_color()* primitive retrieves color details of the primary color attributes. *inq_color* is only intended for use with the basic color set and is provided only for backward compatibility.

The alternative interfaces of [get_color](#), [set_color](#), [set_color_pair](#) and [get_color_pair](#) are the preferred interfaces for new macros.

Parameters

background	Background color.
normal	Normal text color.
selected	Highlighting color for the selected window title.
message	Normal messages.
error	Error messages.
hilite	Color of marked areas. The value either states both the foreground and background of the marked areas as high nibble is background and the low is foreground. Unless hilite_fg is stated in which case only the foreground.
hilite_fg	Foreground color of marked areas.
frame	Window frame.

Returns

The *inq_color()* primitive returns a string containing the current color specification, being a space seperated list of color names

Portability

n/a

See Also[color](#), [get_color](#)**inq_command**

```
string inq_command()
```

Retrieve name of last keyboard command.

Description

The *inq_command()* primitive retrieves the name of last command invoked from keyboard.

Commands with names beginning with an underscore (_) are ignored.

Parameters

none

Returns

The *inq_command()* primitive returns a string containing the name of the last macro called by the user.

Portability

n/a

See Also[inq_message](#)**inq_display_mode**

```
int inq_display_mode([string flagname],
                     [string ~flags] )
```

Inquire display control flags.

Description

The *inq_display_mode()* primitive retrieves the value of the associated display attribute, given by the parameter *flagname*.

Name	Constant	Description
window	DC_WINDOW	Running under a windowing system (ro).
mouse	DC_MOUSE	Mouse enabled/available (ro).
readonly	DC_READONLY	Read-only mode (ro).
charmode	DC_CHARMODE	Character-mode with basic GUI features (ro)
shadow	DC_SHADOW	Display shadow around popups.
showthru	DC_SHADOW_SHOWTHRU	Show-thru shadow around popups.
statusline	DC_STATUSLINE	Status line.
unicode	DC_UNICODE	UNICODE character encoding available (ro).
asciionly	DC_ASCIIONLY	ASCII only characters within UI/dialogs.
rosuffix	DC_ROSUFFIX	Read-only suffix on titles.
modsuffix	DC_MODSUFFIX	Modified suffix.
eof_display	DC_EOF_DISPLAY	Show <EOF> marker.
tilde_display	DC_TILDE_DISPLAY	Show ~ marker.
eol_hilite	DC_EOL_HILITE	Limit hilites to EOL.
himodified	DC_HIMODIFIED	Hilite modified lines.
hiadditional	DC_HIADDITIONAL	Hilite additional lines.
scroll_cols	n/a	Scroll jump column override.
scroll_rows	n/a	Scroll Jump row override.
visible_cols	n/a	Visible display window width lower bounds.
visible_rows	n/a	Display window height lower bounds.
number_cols	n/a	Line number column width override.

Parameter

flagname Optional string parameter, if stated gives the name of the attribute to be retrieved (See: [display_mode](#)). If omitted the display mode are retrieved.

flags Optional string parameter, if stated shall be populated with a comma separated list of active attribute names.

Returns

The *inq_display_mode()* primitive returns the value of the associated attribute, otherwise -1 on error.

Portability

A **GriefEdit** extension.

For system portability use of the manifest constants is advised.

See Also

[display_mode](#), [set_font](#), [inq_font](#)

inq_echo_format

```
int inq_echo_format()
```

Retrieve the echo-line format.

Description

The *inq_echo_format()* primitive retrieves the current echo-line format specification.

Parameters

none

Returns

The *inq_echo_format* returns the current echo line format specification string, otherwise an empty string.

Portability

A **GriefEdit** extension.

See Also

[set_echo_format](#), [echo_line](#), [inq_echo_line](#)

inq_echo_line

```
int inq_echo_line()
```

Retrieve the echo-line flags.

Description

The *inq_echo_line()* primitive retrieves the current echo-line flags.

Parameters

none

Returns

The *inq_echo_line()* primitive returns the current echo_line flags.

Portability

A **GriefEdit** extension.

See Also

[set_echo_format](#), [inq_echo_format](#), [echo_line](#)

inq_font

```
int inq_font(string &normalfont,
             [string &italicfont])
```

Inquire the current window fonts.

Description

The *inq_font()* primitive retrieves the active normal and/or italic font of the current running **GriefEdit** image.

Note:

Only available when running under a suitable windowing system, otherwise this primitive is a no-op.

Parameters

normalfont Optional string variable reference to be populated with the normal text font name.
italicfont Optional string variable reference to be populated with the italic text font name.

Returns

The *inq_font()* primitive returns zero or greater on success, otherwise -1 on error.

Portability

GriefEdit extended.

See Also

[set_font](#)

inq_line_col

```
string inq_line_col()
```

Retrieve the echo_line content.

Description

The *inq_line_col()* primitive retrieves the current content of the echo_line.

Parameters

none

Returns

The *inq_line_col()* primitive returns the current echo line status.

Portability

n/a

See Also

[inq_cmd_line](#)

inq_message

```
string inq_message()
```

Retrieve the prompt line.

Description

The *inq_message()* primitive returns the string that is currently displayed on the command prompt.

Parameters

none

Returns

The *inq_message()* primitive returns a string being what is currently displayed on the message line.

Portability

n/a

See Also

[inq_cmd_line, error, message](#)

inq_prompt

```
int inq_prompt()
```

Retrieve the prompt status.

Description

The *inq_prompt()* primitive retrieves the prompt status flag, which can be used to determine whether the user prompt is current active.

Parameters

none

Returns

The *inq_prompt()* primitive returns a boolean value stating whether or not the user is currently being prompted for input on the command line.

Portability

n/a

See Also

[get_parm](#)

inq_screen_size

```
int inq_screen_size([int &rows],
                    [int &cols],
                    [int &colors])
```

Determine screen dimensions.

Description

The *inq_screen_size()* primitive retrieves the screen dimensions, being the number of text rows and character columns.

Parameters

rows Optional integer reference populated with the number of text rows.
cols Optional integer reference to be populated with the number of character columns.
colors Optional integer reference populated with the color depth supported by the display.

Returns

The *inq_screen_size()* primitive returns zero if terminal is configured as a monochrome screen, or non-zero if in color mode.

Portability

GriefEdit extended.

See Also

[display_mode](#)

inq_window_color

```
int inq_window_color([int winnum])
```

Retrieve the window attribute.

Description

The *inq_window_color* returns the background and foreground colors of a window. If *winnum* is not specified then the colors of the current window are returned.

If the window color is set to < 0, then the default background color will be used. If the color is set to >= 0 then the specified color will be used.

Parameters

winnum Optional window identifier, if omitted the current window shall be referenced.

Returns

Returns the current window attribute, otherwise -1 on error.

Portability

Unlike BRIEF the assigned attributes is returned, whereas BRIEF returned an encoded colorl the foreground were the lower 4 bits (nibble) and the background was the upper 4 bits.

See Also

[get_color_pair](#), [window_color](#)

inq_window_priority

```
int inq_window_priority([int winnum])
```

Retrieve the windows display priority.

Description

The *inq_window_priority()* primitive retrieves the display priority of the specified window, , if omitted the current window.

Parameters

`winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

Window priority, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[set_window_priority](#)

inq_window_size

```
int inq_window_size()
```

Retrieve the window size.

Description

The *inq_window_size()* primitive determines the size of the current window.

Parameters

<code>height</code>	An integer variable which shall be populated with the window height in lines.
<code>width</code>	An integer variable which shall be populated with the window width height in lines.
<code>left_offset</code>	An integer variable which shall be populated with the number of columns that the window has been scroll horizontally, being the number of columns to the left.
<code>lmargin</code>	An integer variable which shall be populated with the window left margin in lines.
<code>rmargin</code>	An integer variable which shall be populated with the window right margin in lines.

Returns

The *inq_window_size()* primitive returns the window height in rows, otherwise -1 if there is no current window.

Portability

The margins left and right are **GriefEdit** extensions.

See Also

[create_window](#), [inq_window](#), [inq_window_info](#), [inq_window_color](#)

redraw

```
void redraw([int winch])
```

Redraw screen.

Description

The *redraw()* primitive redisplays the screen.

Parameters

`none`

Returns

`nothing`

Portability

`n/a`

See Also

[refresh](#)

refresh

```
void refresh()
```

Update the display.

Description

The *refresh()* primitive updates the screen flushing any pending changes.

Parameters

none

Returns

nothing

Portability

n/a

See Also

[redraw](#)

screen_dump

```
int screen_dump(string filename)
```

Dump an image of the screen.

Description

The *screen_dump()* primitive dumps a text representation of the current screen image to the specified file. The resulting image omits both attribute and character-map associations with the view, with frame characters mapped to their ASCII equivalent.

Parameters

`filename` Optional string containing the full path of the output filename, if omitted the path `/tmp/griefscreen.###` shall be used; where `###` represents the next available sequential filename.

Returns

Returns 0 on success, otherwise -1 on error. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

Portability

n/a

See Also

[print](#)

set_color

```
int set_color([list|string spec],
             [int create = TRUE])
```

Set screen colors.

Description

The *set_color()* primitive controls the colors that are utilised on a color display; the values are ignored for monochrome displays.

It is designed to compliment and enhance the functionality of the *color* primitive, with *color()* being a user level interface and *set_color()* being a script level interface.

Parameters

`spec` Color specification, either a list or comma separated set of attributes. The specification may use one of two forms, firstly an explicit set of attributes pairs or secondary as an implicit list of colors, see below.

`create` Optional boolean flag, if either omitted or `true` non-existent attributes shall be created.

Color Specification

The specification may take one of the following forms. For details on available attributes and possible colors, see the *Color Attributes* and *Color Name* sections below.

Explicit:

A list of strings containing attribute color pairs Simple attributes are assigned a color delimited by a = take the form:

```
"attribute=color"
```

For the highlight and syntax attributes an extended form permits explicit selection of the background over the implied default of *background* in addition to optional *style* settings, as follows:

```
"attr=color[:style][,background]"
```

Implicit:

An ordered list of strings corresponding to each color attribute to be set, using the form:

```
"color-name color-name ..."
```

A value of NULL in the list means to skip the assignment of the related attribute. An integer value within the list allows the index reference to be explicitly set for subsequent attribute assignment.

Note, provided primary for CRISP compatibility as the explicit interface is the recommended use of the [set_color](#) primitive to guard against future enhancements/color attributes.

Also note as the color interfaces have developed independently the COL enumeration values differ.

Scheme Specification

A special attribute *scheme* sets the default color scheme, taking the form:

```
scheme=d[ark]|l[ight]
```

Color Names

The following color names are recognised which are case-insensitive, with the color number used which are available on most systems. The first name listed shall be the primary name as returned from inquiry functions ((see [get_color](#))) with the additional as aliases.

Color	Constant	Aliases
Black	BLACK	
Blue	BLUE	
green	GREEN	
Cyan	CYAN	
Red	RED	
Magenta	MAGENTA	
Brown	BROWN	
White	WHITE	
Grey	GREY	Gray
Light-blue	LTBLUE	LightBlue
Light-green	LTGREEN	LightGreen
Light-cyan	LTCYAN	LightCyan
Light-red	LTRED	LightRed
Light-magenta	LTMAGENTA	LightMagenta
Yellow	YELLOW	
Light-white	LTWHITE	LightWhite
Bright-white	LTWHITE	
Light-grey	LTWHITE	LightGrey
Light-gray	LTWHITE	LightGray
Dark-grey	DKGREY	DarkGrey
Dark-gray	DKGREY	DarkGray
Dark-blue	DKBLUE	DarkBlue
Dark-green	DKGREEN	DarkGreen
Dark-cyan	DKCYAN	Darkcyan
Dark-red	DKRED	DarkRed
Dark-magenta	DKMAGENTA	DarkMagenta
Dark-yellow	DKYELLOW	DarkYellow
Light-yellow	LYELLOW	LightYellow

plus the following specials to support black-white and terminal features:

Color	Description
Normal	Normal terminal text.
Inverse	Inverse contrast.
Blink	Blinking text.
Reverse	Reverse contrast.
Standout	Stand-out terminal text.
Dim	Dim colors.

Color Codes

Under environments which support a larger color range, examples include xterm256 and WIN32+, the following additional system colors are available. Either by its color values (e.g. 0x32) or by its Red, Green and Blue (RGB) value.

The format is "#rrggbb" being hexadecimal value in range 00-ff, where:

- rr is the Red value.
- gg is the Green value.
- bb is the Blue value.

Color Attributes

There are a number of display attributes which can be assigned specific colors. These attributes represent a number of different display objects from basis editing, syntax hiliting and dialog.

The table below lists all the attributes, their manifest-constant within the `set_color` and `get_color` interface and description:

Name	Constant	Description
background	COL_BACKGROUND	Background color.
normal	COL_NORMAL_FG	Normal text.
select	COL_SELECT_FG	Title of selected window.
message	COL_MESSAGE_FG	Prompt, messages and status fields.
error	COL_ERROR_FG	Error messages.
hilite	COL_HILITE_BG	Highlighted/markd background.
hilite	COL_HILITE_FG	Highlighted foreground.
frame	COL_BORDERS	Window frame/borders.
cursor_insert	COL_INSERT_CURSOR	Insert mode cursor.
cursor_overtype	COL_OVERTYPE_CURSOR	Over-type mode cursor.
cursor	n/a	
cursor_row	n/a	
cursor_col	n/a	
shadow	COL_SHADOW	
prompt	COL_PROMPT	
prompt_standout	n/a	
prompt_complete	COL_COMPLETION	
question	COL_QUESTION	
echo_line	COL_ECHOLINE	
standout	COL_STANDOUT	
literalchar	n/a	
whitespace	n/a	Highlighted white-space/tabs.
scrollbar	n/a	
scrollbar_thumb	n/a	
column_status	n/a	
column_lineno	n/a	Line numbers.
nonbuffer	n/a	
search	n/a	
search_inc	n/a	
search_match	n/a	
ruler	n/a	
ruler_margin	n/a	
ruler_ident	n/a	
ruler_mark	n/a	
ruler_column	n/a	

Name	Constant	Description
popup_normal	n/a	
popup_hilite	n/a	
popup_standout	n/a	
popup_title	n/a	
popop_frame	n/a	
dialog_normal	n/a	
dialog_focus	n/a	
dialog_hilite	n/a	
dialog_greyed	n/a	
dialog_hotkey_normal	n/a	
dialog_hotkey_focus	n/a	
dialog_frame	n/a	
dialog_title	n/a	
dialog_scrollbar	n/a	
dialog_thumb	n/a	
dialog_but_greyed	n/a	
dialog_but_normal	n/a	
dialog_but_focus	n/a	
dialog_but_key_normal	n/a	
dialog_but_key_focus	n/a	
dialog_edit_greyed	n/a	
dialog_edit_normal	n/a	
dialog_edit_focus	n/a	
dialog_edit_complete	n/a	
lsdirectory	n/a	
lsexecute	n/a	
lssymlink	n/a	
lspipe	n/a	
lsspecial	n/a	
lserror	n/a	
lsreadonly	n/a	
lsnormal	n/a	
lsattribute	n/a	
lssize	n/a	
modified	n/a	
additional	n/a	
difftext	n/a	
diffdelete	n/a	
match	n/a	
link	n/a	
tag	n/a	
alert	n/a	
ansi_bold	n/a	
ansi_underline	n/a	
spell	n/a	
spell_local	n/a	
spell_special	n/a	
comment	n/a	Language comments.
comment_standout	n/a	
todo	n/a	
code	n/a	
constant	n/a	
constant_standout	n/a	
string	n/a	Language string elements.
character	n/a	
operator	n/a	Language operator elements.
number	n/a	Language numeric values.
float	n/a	
delimiter	n/a	Language delimiter elements.
word	n/a	
boolean	n/a	
preprocessor	n/a	Language preprocesor elements.
preprocessor_define	n/a	
preprocessor_include	n/a	

Name	Constant	Description
preprocessor_conditional	n/a	
preprocessor_keyword	n/a	
preprocessor_word	n/a	
keyword	n/a	Language keywords, primary.
keyword_function	n/a	
keyword_extension	n/a	
keyword_type	n/a	
keyword_storageclass	n/a	
keyword_definition	n/a	
keyword_conditional	n/a	
keyword_repeat	n/a	
keyword_exception	n/a	
keyword_debug	n/a	
keyword_label	n/a	
keyword_structure	n/a	
keyword_typedef	n/a	
user1	n/a	User specified 1
user2	n/a	User specified 2
user3	n/a	User specified 3
user4	n/a	User specified 4
user5	n/a	User specified 5
user6	n/a	User specified 6
user7	n/a	User specified 7
user8	n/a	User specified 8
user9	n/a	User specified 9
user10	n/a	User specified 10
window1	n/a	
window2	n/a	
window3	n/a	
window4	n/a	
window5	n/a	
window6	n/a	
window7	n/a	
window8	n/a	
window9	n/a	
window10	n/a	
window11	n/a	
window12	n/a	
window13	n/a	
window14	n/a	
window15	n/a	
window16	n/a	

Styles

The following styles are recognised which are case-insensitive.

Featuree	Description
Underline	Underlined text.
Italic	Italic typeface.
Bold	Bold typeface.
Italic	Italic typeface.
Underline	Under lined text.
Undercurl	Curly underline, generally underline.

Returns

The `set_color()` primitive returns 0 on success, otherwise -1 -1 on failure to set colors.

Portability

A **GriefEdit** extension.

See Also

[get_color](#), [color](#), [inq_color](#), [set_color_pair](#)

set_echo_format

```
void set_echo_format([string format = NULL])
```

Set the echo line format.

Description

The `set_echo_format()` primitive sets or clears the current status area format specification. The format shall be applied to the echo-line overriding the fixed layout implied by `echo_line`.

The status area layout can be defined by one of two means. The `echo_line` specification controls the elements visible using a fixed layout whereas `set_echo_format` allows a user defined format specification.

The following example specification

```
"grief %v: %b (%m) %l %c %t"
```

results in

```
grief v2.6.1: (-rw-rw-rw-rw) Line: 165 Col: 1 3:14pm
```

The echo-line format is applied when the `E_FORMAT` flag is enabled (See: `echo_line`). Setting an echo line format implies `E_FORMAT` similarly clearing the format disables `E_FORMAT`.

Note:

When errors are encountered while evaluating the format specification the incorrect section shall be simply echoed within the echo-line; otherwise screen updating would loop.

Parameters

`format` Optional echo line format specified, if omitted the format is cleared the current `echo_line` shall take effect.

Format Specification

A format specification, which consists of optional and required fields, has the following form:

```
%[flags] [modifier]<type>
```

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a type character (for example, `%b`). If a percent sign is followed by a character that has no meaning as a format field, the character is simply copied. For example, to print a percent-sign character, use `%%`.

The optional fields, which appear before the type character, control other aspects of the formatting, as follows:

`type` Required character that determines whether the associated argument is interpreted as a character, a string, or a number.

`flags` Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.

`modifier` Optional number that specifies a format modifier, selected an alternative form of the associated `type`. If omitted the modifier assumes a value of 0.

Following is a description of the possible echo-line attributes. The second character in "item" is the type:

Type	Description									
b	Buffer details. <table border="1"> <thead> <tr> <th>Modifier</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Buffer title.</td> </tr> <tr> <td>1</td> <td>Buffer number as decimal.</td> </tr> <tr> <td>2</td> <td>Buffer number as hex.</td> </tr> </tbody> </table>		Modifier	Value	0	Buffer title.	1	Buffer number as decimal.	2	Buffer number as hex.
Modifier	Value									
0	Buffer title.									
1	Buffer number as decimal.									
2	Buffer number as hex.									

Type	Description																		
f	Buffer flags. <table border="1"> <thead> <tr> <th>Modifier</th><th>Value</th></tr> </thead> <tbody> <tr> <td>0</td><td>Flags0</td></tr> <tr> <td>1</td><td>Flags1</td></tr> <tr> <td>2</td><td>Flags2</td></tr> <tr> <td>3</td><td>Flags3</td></tr> <tr> <td>4</td><td>Flags4</td></tr> </tbody> </table>	Modifier	Value	0	Flags0	1	Flags1	2	Flags2	3	Flags3	4	Flags4						
Modifier	Value																		
0	Flags0																		
1	Flags1																		
2	Flags2																		
3	Flags3																		
4	Flags4																		
n	File name with directory.																		
N	File name without the directory.																		
p	Percent string.																		
c	Column number. <table border="1"> <thead> <tr> <th>Modifier</th><th>Value</th></tr> </thead> <tbody> <tr> <td>0</td><td>Col: xxx</td></tr> <tr> <td>1</td><td>xxx</td></tr> </tbody> </table>	Modifier	Value	0	Col: xxx	1	xxx												
Modifier	Value																		
0	Col: xxx																		
1	xxx																		
m	Mode string (i.e. --rw-rw-rw).																		
o	Overwrite mode, " OV" otherwise "".																		
O	Overwrite/insert flag, like %o yet shows OV/RE.																		
C	Character value.																		
V	Virtual character indicator.																		
r	Remember flag.																		
l	Line number. <table border="1"> <thead> <tr> <th>Modifier</th><th>Value</th></tr> </thead> <tbody> <tr> <td>0</td><td>Line: xxx</td></tr> <tr> <td>1</td><td>xxx</td></tr> </tbody> </table>	Modifier	Value	0	Line: xxx	1	xxx												
Modifier	Value																		
0	Line: xxx																		
1	xxx																		
L	Number of lines in the file.																		
t	Current time. <table border="1"> <thead> <tr> <th>Modifier</th><th>Value</th></tr> </thead> <tbody> <tr> <td>0</td><td>12 hour</td></tr> <tr> <td>1</td><td>24 hour</td></tr> </tbody> </table>	Modifier	Value	0	12 hour	1	24 hour												
Modifier	Value																		
0	12 hour																		
1	24 hour																		
d	Current date. <table border="1"> <thead> <tr> <th>Modifier</th><th>Value</th></tr> </thead> <tbody> <tr> <td>0</td><td>yyyy-mm[m]-dd</td></tr> <tr> <td>1</td><td>yy-mm[m]-dd</td></tr> <tr> <td>2</td><td>dd-mm[m]-yyyy</td></tr> <tr> <td>3</td><td>mm[m]-dd-yyyy</td></tr> <tr> <td>4</td><td>dd-mm[m]-yy</td></tr> <tr> <td>5</td><td>mm[m]-dd-yy</td></tr> <tr> <td>6</td><td>dd-mm[m]</td></tr> <tr> <td>7</td><td>mm-dd</td></tr> </tbody> </table>	Modifier	Value	0	yyyy-mm[m]-dd	1	yy-mm[m]-dd	2	dd-mm[m]-yyyy	3	mm[m]-dd-yyyy	4	dd-mm[m]-yy	5	mm[m]-dd-yy	6	dd-mm[m]	7	mm-dd
Modifier	Value																		
0	yyyy-mm[m]-dd																		
1	yy-mm[m]-dd																		
2	dd-mm[m]-yyyy																		
3	mm[m]-dd-yyyy																		
4	dd-mm[m]-yy																		
5	mm[m]-dd-yy																		
6	dd-mm[m]																		
7	mm-dd																		
v	GRIEF Version.																		
Y	Current Year. <table border="1"> <thead> <tr> <th>Modifier</th><th>Value</th></tr> </thead> <tbody> <tr> <td>0</td><td>YYYY</td></tr> <tr> <td>1</td><td>YY</td></tr> </tbody> </table>	Modifier	Value	0	YYYY	1	YY												
Modifier	Value																		
0	YYYY																		
1	YY																		

Type	Description
M	Current month of the year.
D	Current day of the month.

The following flags are supported;

Flag	Description	Default
-	Left align the result within the given field width.	Right align.
+	Prefix the output value with a sign (+ or -) if the output value is of a signed type.	Sign appears only for negative signed values () .
0	If width is prefixed with 0, zeros are added until the minimum width is reached. If 0 and appear, the 0 is ignored. If 0 is specified with an integer format (i, u, x, X, o, d) the 0 is ignored.	No padding.
<space>	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear.	No blank appears.

Parameters

format Echo line format specification, if omitted or an empty string the specification is cleared.

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[inq_echo_format](#), [echo_line](#), [inq_echo_line](#)

set_window_priority

```
int set_window_priority(int priority,
                       [int winnum])
```

Set the window display priority.

Description

The *set_window_priority()* primitive sets the display priority of the specified window, if omitted the current window. The priority controls the display order of pop-up windows.

Parameters

priority Window priority between the range of 0 .. 127.

winnum Optional window identifier, if omitted the current window shall be referenced.

Returns

Returns the previous window priority, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also[inq_window_priority](#)**use_tab_char**

```
int use_tab_char([int|string yesno],
                 [int global = FALSE])
```

Configure use of hard/soft tabs.

Description

The *use_tab_char()* primitive controls whether hard or soft tabs shall be utilised within buffers. When enabled hard/physical tab characters shall be inserted into the buffer otherwise one or more space characters are inserted up-to the next indentation/tab stop.

The effected tabs setting shall either be the current buffer if *global* is given as **FALSE** or omitted, otherwise the global configuration setting which is referenced during buffer creation defining the default of subsequent buffers.

The argument *yesno* states the new tab setting value as either a string or an integer flag. If omitted the user shall be prompted as follows:

Fill with tab chars [yes,no]?

Parameters

yesno Optional string or integer containing the required status. A string value of *yes* enables with *no* disabling. An integer of *1* enables, *0* disables and *-1* returns the current with effecting any change.

global Optional integer flag states the effected resource, if **FALSE** or omitted then the current buffer otherwise the global buffer default.

Returns

The *use_tab_char()* primitive returns the previous tab setting of the selected resource, either the current buffer or the global.

Portability

n/a

See Also[tabs](#), [inq_tabs](#)**view_screen**

```
int view_screen()
```

View the content of underlying screen.

Description

The *view_screen()* primitive restores the original screen image visible prior to Grief's execution, returning upon a key press.

Parameters

none

Returns

The *view_screen()* primitive returns zero on success otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also[cursor](#)**window_color**

```
int window_color([int color|string color],
                  [int winnum] )
```

Set the window attribute.

Description

The `window_color()` primitive sets the background of the specified window otherwise if omitted the current window to the stated `color`.

When borders are disabled, this color shall be used as the background of the associated window allowing one to distinguish between individual views.

Parameters

- `color` Can be either an integer containing the color numeric value or a string containing the color name. If omitted, the next color within the color index sequence shall be assigned, see [color_index](#) for details.
- `winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

Returns non-zero if the color was changed successfully, otherwise zero.

Portability

n/a

See Also

[inq_window_color](#), [color_index](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Search and Translate Primitives

Summary

Search and Translate Primitives

Macros

quote_regexp	Quote regexp special characters.
re_comp	Compile a regular expression.
re_delete	Delete a compiled expression.
re_result	Retrieve search captures.
re_search	Search for a string.
re_syntax	Set the regular expression mode.
re_translate	Search and replace.
search_back	Backwards buffer search.
search_case	Set the search pattern case mode.
search_fwd	Buffer search.
translate	Buffer search and replace.

Macros

quote_regexp

```
string quote_regexp(string text)
```

Quote regexp special characters.

Description

The `quote_regexp()` primitive quotes (i.e. prefixes with a backslash) the following set of predefined characters in the specified string `text`, representing most of the regular expression special characters.

The predefined characters are;

- asterisk (*)
- backslash (\)
- caret (^)
- curvy brackets ({}')
- dollar sign (\$)
- parenthesis (())
- percentage (%)
- period (.)
- pipe (|)
- plus sign (+)
- question mark (?)
- square brackets ([])'

Parameters

`text` String containing the text to quote, protecting special regular expression characters..

Returns

The `quote_regexp()` primitive returns the resulting quoted string.

Portability

n/a

See Also

[search_fwd](#), [re_search](#)

re_comp

```
list re_comp(      [int flags],
               string pattern,
               [string &error])
```

Compile a regular expression.

Description

The *re_comp()* primitive is reserved for future development.

The *re_comp* primitive converts a regular expression string (RE) into an internal form suitable for pattern matching.

Parameters

<i>flags</i>	Optional integer flags, one or more of the <i>SF_XXX</i> flags OR'ed together control the options to be applied during the search and replace.
<i>pattern</i>	String containing the pattern to translate.
<i>error</i>	Optional string reference, on an error shall be populated with a description of the error.
<i>parms</i>	Optional string containing configuration parameters.

Returns

The *re_comp()* primitive returns list containing the compiled expression, otherwise a null list.

Warning:

DONOT modify the result otherwise you shall encounter undefined results during search operations.

Portability

n/a

See Also

[re_search](#), [re_translate](#), [re_delete](#)

re_delete

```
int re_delete(list handle)
```

Delete a compiled expression.

Description

The *re_delete()* primitive

Parameters

handle -

Returns

The *re_delete()* primitive returns 1 on success, otherwise 0.

Portability

n/a

See Also

[re_comp](#)

re_result

```
int re_result( [int capture],
               [string &value],
               [int &offset],
               [int &length]   )
```

Retrieve search captures.

Description

The *re_result()* primitive retrieves the results of the last regular expression [re_search](#) operation. In addition the prior search must have had **SF_CAPTURES** enabled.

Warning:

The captured regions reference the original searched object, as such the original buffer, list or

string *must not be modified prior to executing *re_result*.

Parameters

- `capture` Integer capture index, see details below.
- `value` String variable reference, is populated with the value of the referenced capture.
- `offset` Optional integer variable reference, is populated with the starting offset of the capture within the original source.

Capture Index

Value	Description
1 .. 9	Xth captured expression.
CAPTURE_BEFORE	Everything prior to matched string
CAPTURE_ARENA	Entire matched string.
CAPTURE_AFTER	Everything after to matched string
CAPTURE_LAST	Last parenthesized pattern match.

Returns

The `re_result()` primitive returns the length of the capture, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

`re_search`, `search_fwd`, `search_back`, `search_string`, `search_list`

re_search

```
int re_search( [int flags],
              [string pattern],
              [declare object],
              [int start],
              [int lensym]      )
```

Search for a string.

Description

The `re_search()` primitive is a generalised search interface, combining the functionality of the more specialised search primitives but presenting consistent and stateless search capabilities.

In addition both the `re` and `case` modes are explicit based upon the `flags` values.

Parameters

- `flags` Optional integer flags, one or more of the `SF_XXX` flags OR'ed together control the options to be applied during the search.
- `pattern` A string containing the pattern to be matched.
- `object` Optional object to be searched. The search depends on the type of `object`. If `object` is a string, then a string search is performed. If `object` is a list then a list search is done. Ifn an integer the buffer associated with the stated buffer number is searched, otherwise if omitted a search on the current buffer is performed.
- `start` Optional integer index for list objects, states the element offset at which search operation shall start. If omitted, the search is started from the beginning of the list.
- `lensym` Optional integer reference for string objects is populated with the length of the matched region.

Flags

General control flags.

Constant	Description
<code>SF_BACKWARDS</code>	Search in backwards direction. By default searches are performed in a forward direction; only meaningful for buffer searches or translates.
<code>SF_IGNORE_CASE</code>	Ignore/fold case.
<code>SF_BLOCK</code>	Restrict search to current block.
<code>SF_LINE</code>	Line mode.
<code>SF_LENGTH</code>	Return length of match.
<code>SF_MAXIMAL</code>	BRIEF maximal search mode.
<code>SF_CAPTURES</code>	Capture elements are retained.
<code>SF QUIET</code>	Do not display progress messages.

Constant	Description
SF_GLOBAL	Global translate (*); re_translate only.
SF_PROMPT	Prompt for translate changes; re_translate only.
SF_AWK	awk(1) style capture references.
SF_PERLVARS	perl(1) style capture references.

(*) PROMPT is given priority over GLOBAL.

Syntax selection, one of the following mutually exclusive options can be stated.

Constant	Description
SF_NOT_REGEXP	Treat as plain test, not as a regular expression.
SF_BRIEF/SF_CRISP	BRIEF/CRISP/GRIEF expressions (default).
SF_UNIX	Unix regular expressions.
SF_EXTENDED	Extended Unix expressions.
SF_PERL	PERL syntax.
SF_RUBY	Ruby syntax.

Returns

The [re_search\(\)](#) primitive return is dependent on the type of the object searched; matching the corresponding specialised primitive [search_fwd](#), [search_list](#) and [search_string](#). These are summarised below.

Object Type	return
buffer	on success the length of the matched text; otherwise 0 if no match or -1 on error.
list	on success index of the matched element, otherwise -1 if no match.
string	on success the index of first character of match, otherwise 0 if no match.

On error [re_search](#) returns -1.

Portability

n/a

See Also

[re_translate](#), [search_fwd](#), [search_back](#), [search_string](#), [search_list](#), [re_syntax](#)

re_syntax

```
int re_syntax([int re],
             [int case],
             [int capture])
```

Set the regular expression mode.

Description

The [re_syntax\(\)](#) primitive configures the global regular expression syntax mode to the mode *re*. By default, the mode is set to the **RE_BRIEF** regular expression syntax.

re_syntax affects the defaults of the following primitives;

- [search_string](#)
- [search_list](#)
- [search_fwd](#) and [search_back](#)
- [translate](#)

The supported regular expression syntax modes.

Value	Constant	Description
0	RE_BRIEF	BRIEF/Crisp Edit mode.
1	RE_UNIX	Basic Unix syntax.
2	RE_EXTENDED	POSIX Extended Unix.
3	RE_PERL	Perl.
4	RE_RUBY	Ruby style syntax.

Parameters

re	Optional integer value, specifying the new regular expression syntax to be applied. If omitted, the current value is unchanged.
case	Optional integer value, specifying the new state of the global case mode. If <i>case</i> is specified and is zero, then searches are performed as case insensitive, otherwise when non-zero they shall be case sensitive. If omitted, the current value is unchanged.
capture	Optional integer value, specifying the new state of the global replacement pattern reference mode. If <i>RE_PERLVAR</i> then Perl style capture references shall be utilised, when <i>RE_AWKVARS</i> AWK style capture references are utilised, otherwise then 0 the style is dependent on the <i>re</i> mode; see translate . If omitted, the current value is unchanged.

Returns

The *re_syntax()* primitive returns the current/resulting regular expression syntax mode.

Portability

The *case* and *re* arguments are **GriefEdit** extensions.

See Also

[search_case](#), [re_search](#)

re_translate

```
int|string re_translate(      [int flags],
                           string pattern,
                           [string replacement],
                           [declare object] )
```

Search and replace.

Description

The *re_translate()* primitive is a generalised search and replace interface, combining the functionality of the more specialised primitives but presenting consistent and stateless search capabilities.

Parameters

flags	Optional integer flags, one or more of the <i>SF_XXX</i> flags OR'ed together control the options to be applied during the search and replace.
pattern	String containing the pattern to translate.
replacement	String containing the replacement expression for each matched pattern. It may contain references to the pattern's capture groups by <i>special variables</i> using one of three forms GRIEF, AWK or Perl, see below.
object	Optional object to be searched. The search depends on the type of <i>object</i> . If <i>object</i> is a string, then a string search is performed; translating a string provides similar functionality to awk's <i>sub()</i> and <i>gsub()</i> functions. If <i>object</i> is a list then a list search is done. If an integer the buffer associated with the stated buffer number is searched, otherwise if omitted a search on the current buffer is performed.

Special Variables

- GRIEF variables - GRIEF style references.

This style take the form of the form "\d", where *d* is a numeric group number started at an index of 0; note escapes need to be preceded by an additional backslash.

Variable	Value
\0, \1 ...	Xth + 1 captured expression.

- AWK Variable - AWK style references are selected using *SF_AWK*.

This style take the form of "\d", where *d* is a numeric group number indexed from 1; note escapes need to be preceded by an additional backslash.

Variable	Value
\0	Matched string.
\1, \2 ...	Xth captured expression.
&	Match string, same as \0.

- PERL Variables - PERL style references are selected using *SF_PERLVAR*; is it also implied under *SF_PERL*, *SF_RUBY* and *SF_TRE* unless *SF_AWK* is stated.

This style is super-set of all three forms. Capture references take the form "\$x" when *x* is a numeric group number indexed from 1 or one of the special variables \$`, \$&, and \$. In addition the AWK "\x" syntax is supported, yet their use should be avoided due to the need to quote the escapes.

Variable	Value
\1, \2 ...	Old style, Xth captured expression.
\$1, \$2 ...	Xth captured expression.
\${xx}	Xth captured expression.
\$+	Last parenthesized pattern match.
\$`	Everything prior to the matched string.
\$&	Entire matched string.
\$'	Everything after the matched string.

Returns

If *object* is a string, then the resulting translated string is returned.

Otherwise the *re_translate()* primitive returns the number of translations which occurred, otherwise -1 on error.

Portability

n/a

See Also

[re_search](#), [translate](#)

search_back

```
int search_back( string pattern,
                 [int re],
                 [int case],
                 [int block],
                 [int length])
```

Backwards buffer search.

Description

The *search_back()* primitive searches backwards within the active buffer from the current cursor position to the buffer top (or block) against the expression *pattern*. The treatment of the matched pattern may either be a regular-expression or constant string dependent on *re* and *case* folding based on *case*.

Note:

The *search_back* primitive is provided primary for BRIEF compatibility. Since *re* and *case* can reference the current global search states, inconsistent and/or unexpected results may result between usage; as such it is advised that the state-less [re_search](#) primitive is used by new macros.

Parameters

- pattern* A string containing the pattern to be matched.
- re* Optional integer value. If *re* is specified and is zero, then *pattern* is treated as a literal string not as a regular expression; behaviour being the same as [index](#). Otherwise the string shall be treated as a regular expression using the current global syntax, see [re_syntax](#) and [search_back](#) for additional information.
- case* Optional integer value specifying whether case folding should occur. If *case* is specified and is zero, then the search is done with case insensitive. If omitted the current global setting is referenced.
- block* Optional integer flag. If *block* is specified and is non-zero, the search operations are limited to the current marked region.
- length* Optional inter flag. If *length* is specified and is non-zero, indicates that the total length of the text should be returned; regardless of any \\c anchors in the pattern.

Returns

The *search_back()* primitive returns the length of the matched text plus one otherwise zero or less if no match was found.

Alternatively, if *length* is specified the total length of the matched text is returned. Otherwise if a \\c anchor was encountered within the pattern, it returns the length of the text from the position of the marker to the end of the matched text plus one.

Portability

n/a

See Also

[search_fwd](#)

search_case

```
int search_case([int case])
```

Set the search pattern case mode.

Description

The `search_case()` primitive sets or toggles the global search case mode, if omitted the current mode is toggled.

On completion the resulting mode is echo on the command prompt, either as

Case sensitivity on

or

Case sensitivity off

By default all searches are case sensitive, that is `A` and `a` are not equivalent, by setting the case mode to a zero value, case sensitivity will be ignored when performing matches.

`search_case` affects the default case sensitivity of the follow primitives.

- `search_string`
- `search_list`
- `search_fwd` and `search_back`
- `translate`

Parameters

`case` Optional integer value, specifying the new state of the case mode. If `case` is omitted, the current value is toggled.

Returns

The `search_case()` primitive returns the previous value of the case mode.

Portability

n/a

See Also

`search_back`, `search_fwd`, `translate`, `re_syntax`

search_fwd

```
int search_fwd( string pattern,
                [int re],
                [int case],
                [int block],
                [int length])
```

Buffer search.

Description

The `search_fwd()` primitive searches forward within the active buffer from the current cursor position to the buffer end (or block) against the expression `pattern`. The treatment of the matched pattern may either be a regular-expression or constant string dependent on `re` and `case` folding based on `case`.

Note:

The `search_fwd` primitive is provided primary for BRIEF compatibility. Since `re` and `case` can reference the current global search states, inconsistent and/or unexpected results may result between usage; as such it is advised that the state-less `re_search` primitive is used by new macros.

Parameters

`pattern` A string containing the pattern to be matched.

<code>re</code>	Optional integer value. If <code>re</code> is specified and is zero, then <code>pattern</code> is treated as a literal string not as a regular expression; behaviour being the same as <code>index</code> . Otherwise the string shall be treated as a regular expression using the current global syntax, see <code>re_syntax</code> and <code>search_back</code> for additional information.
<code>case</code>	Optional integer value specifying whether case folding should occur. If <code>case</code> is specified and is zero, then the search is done with case insensitive. If omitted the current global setting is referenced.
<code>block</code>	Optional integer flag. If <code>block</code> is specified and is non-zero, the search operations are limited to the current marked region.
<code>length</code>	Optional inter flag. If <code>length</code> is specified and is non-zero, indicates that the total length of the text should be returned; regardless of any <code>\c</code> anchors in the pattern.

Returns

The `search_fwd()` primitive returns the length of the matched text plus one otherwise zero or less if no match was found.

Alternatively, if `length` is specified the total length of the matched text is returned. Otherwise if a `\c` anchor was encountered within the pattern, it returns the length of the text from the position of the marker to the end of the matched text plus one.

Portability

n/a

See Also

[search_back](#)

translate

```
int translate( string pattern,
              string replacement,
              [int global],
              [int re],
              [int case],
              [int block],
              [int forward])
```

Buffer search and replace.

Description

The `translate()` primitive perform string translations within the current buffer. The translate starting at the current cursor position, continuing until the end of the buffer or optionally terminated by the end user.

If either the `pattern` or `replacement` is not specified, then the user is prompted for the change.

Unless `global` is specified, during translate operations the user is informed on the first match and prompted for the desired action.

Change [Yes|No|Global|One|Entire|Abort,Top|Center]?

the actions being

- Yes Replace current match with the pattern and continue searching.
- No Do not replace current match with the pattern and continue searching.
- Global Replace all matches from cursor to end of file.
- Entire Replace all matches in the file.
- Abort Stop the translation.
- Top Moves cursor and the associated text for better visibility to top of page and re-prompts.
- Center Centers the cursor and the associated text for better visibility and re-prompts.

Note:

The `translate` primitive is provided primary for BRIEF compatibility. As the values of `re` and `case` can reference the current global state resulting in possible inconsistent results between usage, it is advised that the state-less `re_translate` primitive is used by new macros.

Parameters

- `pattern` String containing the pattern to `translate`.
- `replacement` String containing the replacement expression for each matched pattern. It may contain references to the pattern's capture groups by *special variables* using one of three forms GRIEF, AWK or Perl, see below.

global	Optional integer flag controls user prompts. If specified as non-zero every occurrence of the pattern is replaced, otherwise only the first occurrence is translated. If <i>global</i> is omitted the user is prompted on each match.
re	Optional integer value. If <i>re</i> is specified and is zero, then <i>pattern</i> is treated as a literal string not as a regular expression; behaviour being the same as index . Otherwise the string shall be treated as a regular expression using the current global syntax, see re_syntax and search_back for additional information.
case	Optional integer value specifying whether case folding should occur. If <i>case</i> is specified and is zero, then the search is done with case insensitive. If omitted the current global setting is referenced.
block	Optional integer flag. If <i>block</i> is specified and is non-zero, the search operations are limited to the current marked region.
forward	Optional integer flag specifying the search direction. If <i>forward</i> is specified as non-zero or omitted, the translation operation moves forward. Otherwise the translation moves backwards until the top of buffer.

Returns

The `translate()` primitive returns the number of translations which occurred, otherwise -1 on error.

Portability

n/a

See Also

[search_fwd](#), [search_back](#), [re_search](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Spell Checker Primitives

Summary

Spell Checker Primitives

Macros

<code>spell_buffer</code>	Spell the specified buffer.
<code>spell_control</code>	Spell control.
<code>spell_dictionary</code>	Spell dictionary modifications.
<code>spell_distance</code>	Edit distance.
<code>spell_string</code>	Spell the specified word or line.
<code>spell_suggest</code>	Suggest spelling of the the specified word.

Macros

`spell_buffer`

```
list spell_buffer( int start_line,
                   [int end_line],
                   [int tokenize = 1],
                   [int suggest] )
```

Spell the specified buffer.

Description

The `spell_buffer()` primitive spell checks the specified lines within the current buffer.

The string passed in should only be split on white space characters.

Furthermore, between calls to reset each string should be passed in exactly once and in the order they appeared in the document. Passing in strings out of order, skipping strings or passing them in more than once may lead to undefined results.

Parameters

<code>start_line</code>	Integer line number, being the first line at which spell
<code>end_line</code>	Optional integer line number, denoting the line at which spell check shall complete. If omitted checks are performed until the end-of-buffer.
<code>tokenize</code>	Optional integer flag, if specified as zero the buffer content shall not be split into tokens during parsing.
<code>suggest</code>	Optional integer flag, if specified as non-zero spelling suggestions shall be returned against each misspelt word.

Returns

The `spell_buffer()` primitive returns a list of possible spelling errors in the form:

```
[<word>, <suggest-list|NULL>, <offset>, <column>, <line> [, <count>]]
```

Portability

A **GriefEdit** extension.

See Also

`spell_string`, `spell_suggest`, `spell_control`, `spell_distance`

`spell_control`

```
declare spell_control(int action,
                      ...)
```

Spell control.

Description

The *spell_control()* primitive manipulates the spell engine attribute associated with the specified *action*. Additional arguments are specific to the action or attribute being modified.

Parameters

- action* Integer identifier of the engine attribute to be manipulated.
- ... Action specific value.

Modes

Action	Description
SPELL_DESCRIPTION	Spell implementation description, returns a string containing the name of the current speller.
SPELL_DICTIONARIES	List of available dictionaries, retrieves a list of string each the name of a available dictionary.
SPELL_LOAD	Load a personal dictionary.
SPELL_SAVE	Save to a personal dictionary.
SPELL_ADD	Add to the personal dictionary.
SPELL_IGNORE	Add to the spell ignore list.
SPELL_REPLACE	Add to the spell replacement list.
SPELL_LANG_ADD	n/a
SPELL_LANG_PRIMARY	n/a
SPELL_LANG_REMOVE	n/a

Returns

The *spell_control()* primitive usually, on success returns zero, otherwise -1 on error.

A few *spell_control* requests return non-integer values, either a string or list, based on the attribute that was manipulated by the specified *action*, see table above.

Portability

A **GriefEdit** extension.

See Also

[spell_buffer](#), [spell_string](#), [spell_suggest](#), [spell_distance](#)

spell_dictionary

```
int spell_dictionary(int,
                     string|list)
```

Spell dictionary modifications.

Description

The *spell_dictionary()* primitive is reserved.

Parameters

n/a

Returns

The *spell_dictionary()* primitive returns -0 on success, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[spell_buffer](#), [spell_string](#), [spell_suggest](#), [spell_control](#), [spell_distance](#)

spell_distance

```
int spell_distance(string a,
                  string b )
```

Edit distance.

Description

The *spell_distance()* primitive computes the "distance" between two words by counting the number of insert,

replace, and delete operations required to permute one word into another.

In general the fewer operations required, the closer the match. Some implementations assign varying scores to the insert, replace, and delete operations. Another common variation varies which operations are considered when computing the distance; for example, the replace operation may not be considered, thereby defining the edit distance solely in terms of insert and delete options.

This algorithm uses one of the more popular algorithms in the edit distance known as the "Levenshtein distance".

The costs assigned to each move are equal, that is of the following operations have a cost of 1.

- Delete a character
- Insert a character
- Character substitution.

Parameters

s1 String one.
s2 Second string.

Returns

The *spell_distance()* primitive returns the edit distance between the two words.

Portability

A **GriefEdit** extension.

See Also

[spell_buffer](#), [spell_string](#), [spell_suggest](#), [spell_control](#),

spell_string

```
int spell_string( string word,
                  [int length],
                  [int tokenize = 0],
                  [int suggest = FALSE])
```

Spell the specified word or line.

Description

The *spell_string()* primitive spell checks the specified string *word*

The string passed in should only be split on white space characters.

Furthermore, between calls to reset each string should be passed in exactly once and in the order they appeared in the document. Passing in strings out of order, skipping strings or passing them in more than once may lead to undefined results.

Parameters

word String containing the text to be checked.
length Optional integer, stating the number of characters within the string to be parsed.
tokenize Optional integer flag, if specified as non-zero the string content shall be split into tokens during parsing.
suggest Optional integer flag, if specified as non-zero spelling suggestions shall be returned against each misspelt word.

Returns

The *spell_string()* primitive returns a value dependent on the input mode.

For word checks, 0 on success, other non-zero.

For string or line checks, list of possible incorrect words offset/line + length pairs with optional suggestion list, with offsets starting from 1.

```
[<word>, <suggest-list|NULL>, <offset>, <column>]
```

Portability

A **GriefEdit** extension.

See Also

[spell_buffer](#), [spell_suggest](#), [spell_control](#), [spell_distance](#)

spell_suggest

```
list spell_suggest(string word,  
                  [int length])
```

Suggest spelling of the the specified word.

Description

The *spell_distance()* primitive spell checks the specified *word* building a list of possible suggestions.

Parameters

word xxx
length xxx

Returns

The *spell_suggest()* primitive returns a list of possible suggestions, otherwise null.

Portability

A **GriefEdit** extension.

See Also

[spell_buffer](#), [spell_string](#), [spell_control](#), [spell_distance](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

String Primitives

Summary

String Primitives

Macros

<code>atoi</code>	Convert string to a decimal number.
<code>cftime</code>	Format time and date.
<code>characterat</code>	Retrieve the character value within a string.
<code>compress</code>	Compress repeated instances of white-space characters.
<code>diff_strings</code>	Compare to strings.
<code>firstof</code>	Leftmost character search.
<code>format</code>	Formatted printing.
<code>index</code>	Search string for a leftmost sub-string or character.
<code>isalnum</code>	Alphanumeric character predicate.
<code>isalpha</code>	Alpha character predicate.
<code>isascii</code>	ASCII character predicate.
<code>isblank</code>	Blank character predicate.
<code>iscntrl</code>	Control character predicate.
<code>iscsym</code>	A symbol character predicate.
<code>isdigit</code>	Numeric character predicate.
<code>isgold</code>	Alphanumeric character predicate.
<code>isgraph</code>	Graphic character predicate.
<code>islower</code>	Lowercase character predicate.
<code>isprint</code>	A printable character predicate.
<code>ispunct</code>	Punctuation character predicate.
<code>isspace</code>	Space character predicate.
<code>isupper</code>	Uppercase character predicate.
<code>isword</code>	Word character predicate.
<code>isxdigit</code>	Hexadecimal character predicate.
<code>itoa</code>	Convert an integer into a string.
<code>lastof</code>	Rightmost character search.
<code>lower</code>	Convert string or character to lowercase.
<code>ltrim</code>	Chomp characters from the front of a string.
<code>macro_list</code>	Retrieve list of current macros.
<code>read</code>	Read characters from the buffer.
<code>rindex</code>	Search string for a rightmost sub-string or character.
<code>rtrim</code>	Chomp characters from the end of a string.
<code>search_string</code>	Searches for a pattern in a string.
<code>split</code>	Split a string into tokens.
<code>split_arguments</code>	Argument split.
<code>sprintf</code>	Formatted printing to a string.
<code>sscanf</code>	Read formatted data from string.
<code>strcasecmp</code>	String case insensitive compare.
<code>strcasestr</code>	Locate first occurrence of a case insensitive.
<code>strcmp</code>	String compare.
<code>strfilecmp</code>	Filename comparison.
<code>strftime</code>	Format time and date.
<code>string_count</code>	Count occurrences of characters in a string.
<code>strbrk</code>	Search a string for any of a set of characters.
<code>strupr</code>	Pop the leading character(s).
<code>strrstr</code>	Locate last occurrence of a sub-string.
<code>strstr</code>	Locate first occurrence of a sub-string.
<code>strtod</code>	String to double.
<code>strtof</code>	String to float.
<code>strtol</code>	Convert a string into its numeric value.
<code>strverscmp</code>	Version string compare.
<code>substr</code>	Extract a sub-string.
<code>tokenize</code>	Tokenize a string into token elements.
<code>trim</code>	Chomp characters from a string.
<code>upper</code>	Convert string or character to uppercase.
<code>ctype</code>	Character classes

Macros

atoi

```
int atoi(string str,
         [int svalue = TRUE])
```

Convert string to a decimal number.

Description

The *atoi()* primitive converts the initial portion of the string *str* into its numeric value. This behaviour is equivalent to using [strtol](#) as follows.

```
val = strtol(str, NULL, 10);
```

Optionally *atoi* if *svalue* is specified and zero then the ascii value of the first character in string is returned. This behaviour is equivalent to using [characterat](#) as follows.

```
val = characterat(str, 1);
```

Parameters

str String object.

svalue Optional flag, when stated as FALSE only the value of the leading character is returned.

Returns

The *atoi* function returns integer value of argument string treated as an ascii number or the ascii value of the first character in string if *svalue* is zero.

Portability

A [GriefEdit](#) extension.

See Also

[strtol](#)

cftime

```
string cftime(string format,
              int      time   )
```

Format time and date.

Description

The *cftime()* primitive is an alternative interface to [strftime](#).

Returns

The *cftime* function returns a string containing the formatted time and date.

Portability

Provided for CRISP™ compatibility, see [strftime](#).

See Also

[strftime](#)

characterat

```
int characterat(string str,
                int      index)
```

Retrieve the character value within a string.

Description

The *characterat* function returns the character value within the string *str* located at the specified position *index* starting at offset one.

Parameters

str String object to be searched.

index Character index, starting from on offset of 1 being the first character.

Return

Character value as an integer, otherwise -1.

Portability

A **GriefEdit** extension.

compress

```
string compress(
    string str,
    [int trim = FALSE],
    [string chars = "\t\r\n"],
    [int replacement = ' ']
)
```

Compress repeated instances of white-space characters.

Description

The *compress()* primitive takes a string and removes all multiple white space characters, by converting consecutive white-space characters into a single white-space character.

The default is to compress all tabs, spaces and newline characters. If *trimchars* is specified, then all characters within the *trimchar* string are compressed.

If *trim* is specified then *compress()* acts the same as

```
trim(compress(str,trimchars),trimchars)
```

Parameters

<i>str</i>	String object to be compressed.
<i>trim</i>	Optional flag, when TRUE invokes trim on the compressed result.
<i>chars</i>	Optional string defining the set to characters be to compressed and optionally trimmed.
<i>replacement</i>	Optional replacement character, if given as a non-positive (≤ 0) value when the first character with the consecutive set shall be used.

Returns

Returns a copy of string with all spaces, tabs and newlines mapped to single spaces.

Portability

The *chars* and *replacement* options are a **GriefEdit** extensions.

By default BRIEF preserved the first character in every group of compressed characters, whereas **GriefEdit** replaces them with a single space; unless replacement is stated as a non-positive number (e.g. -1).

See Also

[trim](#), [rtrim](#), [ltrim](#)

diff_strings

```
int diff_strings([int flags],
                string s1,
                string s2)
```

Compare to strings.

Description

The *diff_strings()* primitive determines whether the specified strings are equivalent based on the set of formatting rules specified by the comparison flags *flags*.

Comparison Flags

Line comparison flags, which affects the treatment of whitespace and character case.

Constant	Description
DIFF_IGNORE_WHITESPACE	Ignore white-space differences.
DIFF_IGNORE_CASE	Ignore character case, whereby characters using different case shall be treated as equivalent.

Constant	Description
DIFF_COMPRESS_WHITESPACE	Repeated whitespace characters are compressed into a single space and compared as such.
DIFF_SUPPRESS.LEADING	Leading whitespace is ignored.
DIFF_SUPPRESS.TRAILING	Trailing whitespace is ignored.
DIFF_SUPPRESS_LFCR	Line-feed and carriage-return characters are ignored.

Parameters

- flags Optional integer flags defining the rules to be applied during their comparison which maybe one or more of the predefined flags or'd together. If omitted or zero *diff_strings* behaves similar to `strcmp`.
 s1 First string to compare.
 s2 Second string against which to compare the first.

Returns

The *diff_strings()* primitive returns zero if the two string are equivalent otherwise non-zero.

Portability

A **GriefEdit** extension.

See Also

`<diff_buffers>`, `<diff_lines>`, `strcmp`, `strcasestr`

firstof

```
int firstof(string str,
            string chars,
            [int &result])
```

Leftmost character search.

Description

The *firstof()* primitive returns the offset to the first occurrence of any of the characters contained within *chars* in the string *str*.

If supplied, on success *result* shall be populated with the matching character otherwise is set if zero.

firstof() is similar to `index` yet allows multiple characters to be matched.

Parameters

- str String object to be searched.
 chars Character set to match against.
 result Optional result, populated with the matching character value.

Returns

The *firstof()* primitive returns the starting offset or 0 if none of the characters are found.

Portability

A **GriefEdit** extension.

See Also

`lastof`, `index`, `rindex`

format

```
string format(string format,
              ...)
```

Formatted printing.

Description

The *format()* primitive produces formatted output according to the format specification *format*. The trailing arguments ... are the integer, string or list expressions used to satisfy the % formatting options; refer to (See: `message`) for details on the supported format specifications.

The *format* primitive is similar to the `sprintf()` primitive with the exception the formatted result is returned directly as a string.

This function is one of a set of formatted output primitives each supporting a common feature set, see `message` for

a complete discussion on the supported format specification syntax.

Parameters

- `format` String that contains the text to be written. It can optionally contain an embedded <Format Specification> that are replaced by the values specified in subsequent additional arguments and formatted as requested.
- ... Optional format specific arguments, depending on the format string, the primitive may expect a sequence of additional arguments, each containing a value to be used to replace a format specifier in the format string.
There should be at least as many of these arguments as the number of values specified in the format specifiers. Additional arguments are ignored by the primitive.

Returns

The `format()` primitive returns a string containing the formatted results, as defined by the format specification and the values contained within the associated arguments.

Portability

See [message](#)

See Also

[sprintf](#)

index

```
int index(    string    str,
             int ch|string s    )
```

Search string for a leftmost sub-string or character.

Description

The `index()` primitive returns the offset to the first occurrence of the character `ch` or the string `s` in the string `str`.

Parameters

- `str` String object to be searched.
- `ch|s` Object to be matched against.

Returns

The `index()` primitive returns the starting offset or 0 if the character or string was not found.

Portability

The character needle form is a **GriefEdit** extension.

See Also

[rindex](#)

isalnum

```
int isalnum(string |int object,
           [int index])
```

Alphanumeric character predicate.

Description

The `isalnum()` primitive determines whether the specified object `object` belongs to the `alpha` or `numeric` character-classes.

This is [0 through 9], [A through Z] and [a through z] in the program's current locale; which is equivalent to `(isalpha() || isdigit())`.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional `index` allows an alternative character within the string to be tested, with the first character represented by offset `one`.

Parameters

- `object` Character or string object.
- [`index`] If a string, an optional character index within the string, starting at offset one being the first character. If the index denotes a character out-of-bounds, the function returns 0.

Returns

The `isalnum()` primitive returns non-zero if *object* is an alphanumeric character; otherwise it returns 0.

Portability

The *index* option is a **GriefEdit** extension.

Example

```
if (isalnum(read(1)))
    message("Next character is alnum.");
```

See Also

[ctype](#)

isalpha

```
int isalpha(string |int object,
           [int index])
```

Alpha character predicate.

Description

The `isalpha()` primitive determines whether the specified object *object* belongs to the *alpha* character-class.

This is [A through Z] and [a through z] in the program's current locale; which is equivalent to `(isupper() | islower())`.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. If the index denotes a character out-of-bounds, the function returns 0.

Returns

The `isalpha()` primitive returns non-zero if *object* is an alphanumeric character; otherwise it returns 0.

Example

```
if (isalpha(read(1)))
    message("Next character is a alpha character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

isascii

```
int isascii(string |int object,
           [int index])
```

ASCII character predicate.

Description

The `isascii()` primitive determines whether the specified object *object* belongs to the *ascii* character-class.

This is any character value in the range 0 through 0177 (0 through 0x7F), inclusive.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case

the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

- object* Character or string object.
- [*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isascii()* primitive returns non-zero if *object* is an alphanumeric character; otherwise it returns 0.

Example

```
if (isascii(read(1)))
    message("Next character is an ascii character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

isblank

```
int isblank(string |int object,
           [int index])
```

Blank character predicate.

Description

The *isblank()* primitive determines whether the specified object *object* belongs to the *blank* character-class.

This is a space () or tab (\t) character.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

- object* Character or string object.
- [*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isblank()* primitive returns non-zero if *object* is an alphanumeric character; otherwise it returns 0.

Example

```
if (isblank(read(1)))
    message("Next character is blank.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

iscntrl

```
int iscntrl(string |int object,
           [int index])
```

Control character predicate.

Description

The *iscntrl()* primitive determines whether the specified object *object* belongs to the *control* character-class.

The control-class is any character for which the *isprint()* subroutine returns a value of False (0) and any character that is designated a control character in the current locale. For the C locale, control characters are the ASCII delete character (0177 or 0x7F), or an ordinary control character

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *iscntrl()* primitive returns non-zero if *object* is a control character; otherwise it returns 0.

Example

```
if (iscntrl(read(1)))
    message("Next character is a control character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

iscsym

```
int iscsym(string |int object,
           [int index])
```

A symbol character predicate.

Description

The *iscsym()* primitive determines whether the specified object *object* belongs to the *c-symbol* classes, which represent a symbol in the C/C++, **GriefEdit** macro and similar languages.

This is [0 through 9], [A through Z], [a through z] and underscore (_) in the program's current locale. which is equivalent to (*isalpha()* || *isdigit()* || _).

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *iscsym()* primitive returns non-zero if *object* is a symbol character; otherwise it returns 0.

Example

```
if (iscsym(read(1)))
    message("Next character is symbol character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

ctype

isdigit

```
int isdigit(string |int object,
           [int index])
```

Numeric character predicate.

Description

The *isdigit()* primitive determines whether the specified object *object* belongs to the *numeric* character-class.

This is [0 through 9] in the program's current locale.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isdigit()* primitive returns non-zero if *object* is an numeric character; otherwise it returns 0.

Example

```
if (isdigit(read(1)))
    message("Next character is a numeric character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

ctype

isgold

```
int isgold(string |int object,
           [int index])
```

Alphanumeric character predicate.

Description

The *isgold()* primitive determines whether the specified object *object* belongs to either the function, keypad or special character-classes.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested. The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isgold()* primitive returns non-zero if *object* is a meta/gold key character; otherwise it returns 0.

Portability

A **GriefEdit** extension.

Example

```
if (isgold(read(1))
    message("Next character is gold.");
```

See Also[ctype](#)**isgraph**

```
int isgraph(string |int object,
           [int index])
```

Graphic character predicate.

Description

The *isgraph()* primitive determines whether the specified object *object* belongs to the *graphic* character-classes.

The a graphic character is equivalent to

```
(isprint() && not-space)
```

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isgraph()* primitive returns non-zero if *object* is an graphic character; otherwise it returns 0.

Example

```
if (isgraph(read(1)))
    message("Next character is a graphic character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also[ctype](#)**islower**

```
int islower(string |int object,
           [int index])
```

Lowercase character predicate.

Description

The *islower()* primitive determines whether the specified object *object* belongs to the *lower-case* character-classes.

This is [a through z] in the program's current locale.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

`object` Character or string object.

[`index`] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The `islower()` primitive returns non-zero if `object` is an lower-case character; otherwise it returns 0.

Example

```
if (islower(read(1)))
    message("Next character is a lower-case character.");
```

Portability

The `index` option is a **GriefEdit** extension.

See Also

[ctype](#)

ispaint

```
int isprint(string |int object,
           [int index])
```

A printable character predicate.

Description

The `isprint()` primitive determines whether the specified object `object` belongs to the *printable* character-class.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional `index` allows an alternative character within the string to be tested, with the first character represented by offset `one`.

Parameters

`object` Character or string object.

[`index`] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The `isprint()` primitive returns non-zero if `object` is an printable character; otherwise it returns 0.

Example

```
if (isprint(read(1)))
    message("Next character is printable.");
```

Portability

The `index` option is a **GriefEdit** extension.

See Also

[ctype](#)

ispunct

```
int ispunct(string |int object,
            [int index])
```

Punctuation character predicate.

Description

The `ispunct()` primitive determines whether the specified object `object` belongs to the *punctuation* character-class.

Punctuation characters are ones that are printable ((see [isprint](#))) character but neither alphanumeric (See: [isalnum](#)) nor a space (See: [isspace](#)).

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case

the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *ispunct()* primitive returns non-zero if *object* is an alphanumeric character; otherwise it returns 0.

Example

```
if (ispunct(read(1)))
    message("Next character is alnum.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

isspace

```
int isspace(string |int object,
            [int index])
```

Space character predicate.

Description

The *isspace()* primitive determines whether the specified object *object* belongs to the *space* character-class.

White-space characters are (space, form feed (\f), new-line (\n), carriage-return (\r), horizontal tab (\t) or vertical tab (\v).

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isspace()* primitive returns non-zero if *object* is an whitespace character; otherwise it returns 0.

Example

```
if (isspace(read(1)))
    message("Next character is a space character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

isupper

```
int isupper(string |int object,
            [int index])
```

Uppercase character predicate.

Description

The *isupper()* primitive determines whether the specified object *object* belongs to the *uppercase* character-class.

This is [A through Z] in the program's current locale.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isupper()* primitive returns non-zero if *object* is an uppercase character; otherwise it returns 0.

Example

```
if (isupper(read(1)))
    message("Next character is an uppercase character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

isword

```
int isword(string |int object,
           [int index])
```

Word character predicate.

Description

The *isword()* primitive determines whether the specified object *object* belongs to the *word* character-class.

This is [A through Z], [a through z], [0 through 9] and underscore (_) or dash (-) in the program's current locale; which is equivalent to (*isalpha()* || *isdigit()* || *_* || *-*).

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. if the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isword()* primitive returns non-zero if *object* is an word character; otherwise it returns 0.

Example

```
if (isword(read(1)))
    message("Next character is a word character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

isxdigit

```
int isxdigit(string |int object,
             [int index])
```

Hexadecimal character predicate.

Description

The *isxdigit()* primitive determines whether the specified object *object* belongs to the *hexadecimal* character-class.

This is [0 through 9], [A through f] and [a through f] in the program's current locale.

The specified object can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested.

The optional *index* allows an alternative character within the string to be tested, with the first character represented by offset *one*.

Parameters

object Character or string object.

[*index*] If a string, an optional character index within the string, starting at offset one being the first character. If the index denotes a character out-of-bounds, the function returns 0.

Returns

The *isxdigit()* primitive returns non-zero if *object* is a hexadecimal character; otherwise it returns 0.

Example

```
if (isxdigit(read(1)))
    message("Next character is a hexadecimal character.");
```

Portability

The *index* option is a **GriefEdit** extension.

See Also

[ctype](#)

itoa

```
string itoa(int value,
            [int base = 10])
```

Convert an integer into a string.

Description

The *itoa()* primitive converts an integer value to a string using the specified base and returns the result.

If base is 10 and value is negative, the resulting string is preceded with a minus sign (-). With any other base, value is always considered unsigned.

Returns

Upon successful completion, *itoa()* returns the converted value. If no conversion could be performed, *itoa()* returns an empty string.

Portability

A **GriefEdit** extension.

See Also

[atoi](#), [strtod](#), [strtof](#), [sscanf](#)

lastof

```
int lastof(string str,
           string chars,
           [int &result])
```

Rightmost character search.

Description

The *lastof()* primitive returns the offset to the last occurrence of any of the characters contained within *chars* in the string *str*.

If supplied, on success *result* shall be populated with the matching character otherwise is set if zero.

lastof() is similar to [rindex](#) yet allows multiple characters to be matched.

Parameters

str String object to be searched.
chars Character set to match against.
result Optional result, populated with the matching character value.

Returns

The *lastof()* primitive returns the starting offset or 0 if none of the characters are found.

Portability

A **GriefEdit** extension.

See Also

[index](#), [rindex](#)

lower

```
string|int lower(string str|int character)
```

Convert string or character to lowercase.

Description

The *lower()* primitive converts all alphabetic characters within the string object *str* or just the specified character *ch* to lowercase.

Returns

Returns the specified object with all alphabetic characters converted to lowercase.

If the argument is a string then a copy of the string is returned, otherwise the integer value of the converted character.

Portability

Character support is a **GriefEdit** extension.

See Also

ltrim

```
string ltrim(string str,
             [string chars = " \\\t\\\r\\\n"])
```

Chomp characters from the front of a string.

Description

The *ltrim()* primitive removes leading (or left) characters from the specified *string*.

The default is to remove all tabs, spaces and newline characters. If *chars* is specified, then all characters within the *trimchar* string are removed from the beginning of the string.

Returns

Returns a copy of string with all leading white space characters removed. (spaces, tabs and newlines by default).

Portability

n/a

See Also

[compress](#), [trim](#), [rtrim](#)

macro_list

```
list macro_list([string pattern = NULL])
```

Retrieve list of current macros.

Description

The *macro_list()* primitive retrieves a sorted list of all defined macros.

The optional *pattern* is a regular expression similar to that accepted by the command line shells (See: [file_match](#)), providing an inclusion filter.

Parameters

pattern Optional string value containing the macro-name pattern to be applied against each name, only returning the matching. A macro-name expression is a shell style regular expression (See: [file_match](#)) (e.g. * for wildcard, ? for wild-character, and [...] to select a range of characters).

Returns

The *macro_list()* primitive returns a list of strings each containing the name of a macro.

Portability

n/a

See Also

[command_list](#), [file_match](#)

read

```
string read([int number],
           [int &status])
```

Read characters from the buffer.

Description

The *read()* primitive reads characters up to the specified number of characters *length*, if omitted characters are read up to the end of the current buffer including the new-line terminator.

The current buffer position remains unchanged.

Parameters

- 0 - Partial, within a line.
- 1 - End of line.
- -1 - End of file.

number An optional number of characters to be read.

status Optional integer variable reference to be populated with the read completion status, representing the position of the cursor at the end of the read operation;

Returns

The *read()* primitive returns the string read.

Portability

n/a

See Also

[insert](#), [insertf](#)

rindex

```
int rindex(   string   str,
             int ch|string s  )
```

Search string for a rightmost sub-string or character.

Description

The *rindex()* primitive returns the offset to the last (in otherwords the right) occurrence of the character *ch* or the string *s* in the string *str*.

Parameters

str String object to be searched.
ch|s Object to be matched against.

Returns

The *rindex()* primitives returns the starting offset or 0 if the character or string was not found.

Portability

The character needle form is a **GriefEdit** extension.

See Also

[index](#)

rtrim

```
string rtrim(string str,
            string chars = "\\t \\r \\n")
```

Chomp characters from the end of a string.

Description

The *rtrim()* primitive removes trailing (or right) characters from the specified *string*.

The default is to remove all tabs, spaces and newline characters. If *chars* is specified, then all characters within the *trimchar* string are removed from the end of the string.

Returns

Returns a copy of *string* with all trailing white space characters removed. (spaces, tabs and newlines by default).

Portability

A **GriefEdit** extension; this is a replacement of the original *trim()* function.

See Also

[compress](#), [trim](#), [ltrim](#)

search_string

```
int search_string( string pattern,
                   string text,
                   [int &length],
                   [int re],
                   [int case]     )
```

Searches for a pattern in a string.

Description

The *search_string()* primitive searches the string *text* against the expression *pattern*. The treatment of the matched pattern may either be a regular-expression or constant string dependent on *re* and with case folding based on *case*.

Note:

The *search_string* primitive is provided primary for BRIEF compatibility. Since *re* and *case* can reference the current global search states, inconsistent and/or unexpected results may result between usage; as such it is advised that the state-less [re_search](#) primitive is used by new macros.

Parameters

- pattern* A string containing the pattern to be matched.
- text* A string containing the text to be search for the specified *pattern*.
- length* Optional integer variable reference. If this search is successful and specified is populated with the length of the matched text.
- re* Optional integer value. If *re* is specified and is zero, then *pattern* is treated as a literal string not as a regular expression; behaviour being the same as [index](#). Otherwise the string shall be treated as a regular expression using the current global syntax, see [re_syntax](#) and [search_back](#) for additional information.
- case* Optional integer value specifying whether case folding should occur. If *case* is specified and is zero, then the search is done with case insensitive. If omitted the current global setting is referenced.

Returns

The *search_string()* primitive returns the starting character in *string* where the match was found, or zero if the match failed.

Alternatively, if a `\|c` anchor was encountered within the pattern, it returns the length of the text from the position of the marker to the end of the matched text plus one.

Portability

n/a

See Also[re_search](#)**split**

```
list split( string      expr,
            string|integer delims,
                           [int numeric = FALSE],
                           [int noquoting = FALSE],
                           [int empties = FALSE],
                           [int limit = NULL]      )
```

Split a string into tokens.

Description

The *split()* primitive splits the string *expr* into a list of strings and returns the resulting list.

split provides simple field processing, compared to the [tokenize](#) primitive which offers greater functionality.

Parameters

expr	String to be parsed.
delims	Specifies either a string containing the characters used to split the token, alternative an integer character value of a single character.
numeric	If specified and is true(1), then all tokens which start with a digit are converted to a number, rather than a string. Alternative when given as two (2) then values are converted using <code>strtol()</code> , allowing support leading base specifications hexadecimal (0x), octal (0) and binary (0b).
nonquoting	Upon the <i>delim</i> parameter containing a double quote character then it is assumed that any entries inside double quotes should have any embedded characters preserved. When specified the <i>noquote</i> optional allows explicit control enabling and disabling of this feature.
empties	Upon the <i>delim</i> character being given as a integer value empty split column are not returned within the list. When specified the <i>empties</i> optional allows explicit control enabling (non-zero) and disabling (zero) to whether empties are returned.
limit	Limit the split to <i>limit</i> elements, returning remaining content in the last element; not implemented at this time.

Results

List of strings where each string is a *token* from the string parameter.

Portability

The options *empties* and *limit* are **GriefEdit** extensions.

See Also[tokenize](#), [sscanf](#), [index](#), [substr](#)**Examples**

Using | was a delimiter and empties being returned.

```
''      ==> ''
'a'    ==> 'a'
'|b'    ==> '|', 'b'
'||'   ==> '|', '|'
'a|b'  ==> 'a', 'b'
'a|b|' ==> 'a', 'b', '|'
'a|b|c' ==> 'a', 'b', 'c'
'a||b' ==> 'a', '|', 'b'
```

the same without empties

```
''      ==>
'a'    ==> 'a'
'|b'    ==> 'b'
'||'   ==>
'a|b'  ==> 'a', 'b'
'a|b|' ==> 'a', 'b'
'a||b' ==> 'a', 'b'
```

split_arguments

```
list split_arguments(string arguments)
```

Argument split.

Description

The *split_arguments()* primitive splits *argument* into list of words. As it parses the command line, *split_arguments* looks for command separators, white space (spaces and tabs). Normally, these special characters cannot be passed to a command as part of an argument. However, special characters in an argument by enclosing the entire argument in double quotes ["]. The ["] themselves will not be passed to the command as part of the argument. Within a double quoted string the ["] character and [\] character can be represented as [\""] and [\\].

Parameters

argument String containing the argument buffer.

Return

The *split_argument* returns a list of words encountered within the argument buffer.

Portability

A **GriefEdit** extension.

See Also

[getsubopt](#)

sprintf

```
int sprintf(string &buffer,
           string format,
           ...     )
```

Formatted printing to a string.

Description

The *sprintf()* primitive produces formatted output according to the format specification *format* into the given string *buffer*. The trailing arguments ... are the integer, string or list expressions used to satisfy the % formatting options; refer to (See: [message](#)) for details on the supported format specifications.

The format primitive is similar to the C *sprintf()* primitive, exporting the formatted result into the destination buffer.

This function is one of a set of formatted output primitives each supporting a common feature set, see [message](#) for a complete discussion on the supported format specification syntax.

Parameters

buffer String which shall be populated with the result.

format String that contains the text to be written. It can optionally contain an embedded <Format Specification> that are replaced by the values specified in subsequent additional arguments and formatted as requested.

... Optional format specific arguments, depending on the format string, the primitive may expect a sequence of additional arguments, each containing a value to be used to replace a format specifier in the format string.

There should be at least as many of these arguments as the number of values specified in the format specifiers. Additional arguments are ignored by the primitive.

Returns

The *sprintf()* primitive returns the number of characters stored within the result string *buffer*.

Portability

See [message](#)

See Also

[format](#), [error](#), [message](#)

sscanf

```
int sscanf(string str,
           string format,
           ...     )
```

Read formatted data from string.

Description

The `sscanf()` primitive reads data from the string buffer `str`. Data input are stored in the locations specified by argument according to the format string `format`.

The additional arguments should be objects of the type specified by their corresponding `format` specifier within the format string.

Parameters

- `str` Source buffer.
- `format` String that contains a format string, see below.
- `...` Additional arguments; depending on the format string, the function may expect a sequence of additional arguments, each containing a reference to a variable where the interpretation of the extracted characters is stored with the appropriate type. There should be at least as many of these arguments as the number of values stored by the format specifiers. Additional arguments are ignored by the function.

Return Value

The `sscanf()` primitive returns the number of input fields that were successfully converted. An EOF (-1) is returned if an error is encountered.

Portability

A **GriefEdit** extension.

Format Specification

The `format` may be composed of one or more whitespace characters, non whitespace characters, and format specifications.

The format string is read from left to right. Characters that are not part of the format specifications must match characters in the input stream. These characters are read from the input stream but are discarded and not stored. If a character in the input stream conflicts with the format string, `scanf` terminates. Any conflicting characters remain in the input stream.

- whitespace characters - blank (' '), tab ('\t'), or newline ('\n'), cause `scanf` to skip whitespace characters in the input stream. A single whitespace character in the format string matches 0 or more whitespace characters in the input stream.
- non whitespace characters - with the exception of the percent sign (%), cause `scanf` to read but not store a matching character from the input stream. The `scanf` function terminates if the next character in the input stream does not match the specified non-whitespace character.
- format specifications - begin with a percent sign (%) and cause `scanf` to read and convert characters from the input stream to the specified type values. The converted value is stored to an argument from the parameter list. Characters following a percent sign that are not recognized as a format specification are treated as ordinary characters. For example, %% matches a single percent sign in the input stream.

Format Specification

The first format specification encountered in the format string references the first argument after `format`. The `scanf` function converts input characters and stores the value using the format specification. The second format specification accesses the second argument after `format`, and so on. If there are more arguments than format specifications, the extra arguments are ignored. Results are unpredictable if there are not enough arguments for the format specifications.

Values in the input stream are called input fields and are delimited by whitespace characters. When converting input fields, `scanf` ends a conversion for an argument when a whitespace character or another unrecognized character is encountered.

Format specifications have the following format

```
% [*] [width] type
```

Each field in the format specification can be a single character or a number which specifies a particular format option.

The type field is where a single character specifies whether input characters are interpreted as a character, string, or number. This field can be any one of the characters in the following table.

scanf types

Character	Argument Type	Input Format
d	int &	Signed decimal number.
i	int &	Signed decimal, hexadecimal, or octal integer.
u	int &	Unsigned decimal number.
o	int &	Unsigned octal number.
x	int &	Unsigned hexadecimal number.
e	float &	Floating-point number.
f	float &	Floating-point number.
g	float &	Floating-point number.

Character	Argument Type	Input Format
c	int &	A single character.
s	string &	A string of characters terminated by whitespace.
\[string &	A string not to be delimited by space characters.

Type Specifiers

An asterisk (*) as the first character of a format specification causes the input field to be scanned but not stored. The asterisk suppresses assignment of the format specification.

The width field is a non-negative number that specifies the maximum number of characters read from the input stream. No more than width characters are read and converted for the corresponding argument. However, fewer than width characters may be read if a whitespace or other unrecognized character is encountered first.

Character Set

[indicates a string not to be delimited by space characters.

The conversion specification includes all subsequent characters in the format string up to and including the matching right square bracket (]).

The characters between the square brackets comprise the scanset, unless the character after the left square bracket is a circumflex (^), in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right square bracket.

If the conversion specification begins with "[]" or "[^]", the right square bracket is included in the scanlist and the next right square bracket is the matching right square bracket that ends the conversion specification; otherwise the first right square bracket is the one that ends the conversion specification.

If a hyphen character (-) is in the scanlist and is not the first character, nor the second where the first character is a circumflex (^), nor the last character, it indicates a range of characters to be matched. To include a hyphen, make it the last character before the final close bracket. For instance, '[^]0-9-]' means the set 'everything except close bracket, zero through nine, and hyphen'.

Character classes

Within a bracket expression, the name of a character class enclosed in [: and :] stands for the list of all characters (not all collating elements!) belonging to that class.

Types

Identifier	Description
alpha	A letter.
upper	An upper-case letter.
lower	A lower-case letter.
blank	A space or tab character.
digit	A decimal digit.
xdigit	A hexadecimal digit.
alnum	An alphanumeric (letter or digit).
print	An alphanumeric (same as alnum).
blank	A space or tab character.
space	A character producing white space in displayed text.
punct	A punctuation character.
graph	A character with a visible representation.
cntrl	A control character.
word	A "word" character (alphanumeric plus "_").

Type Specifiers

strcasemp

```
int strcasemp(string s1,
              string s2,
              [int length])
```

String case insensitive compare.

Description

The `strcmp()` primitive shall compare the string *s1* to the string *s2*, ignoring case.

Parameters

s1 First string.

`s2` Second string to compare against.

`length` Optional, when specified only the first `length` characters of both string shall be compared.

Return

`strcasestr()` shall return an integer greater than, equal to, or less than 0, if the string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2`, respectively; when ignoring case.

Portability

A **GriefEdit** extension.

See Also

`strcmp`, `==`, `<=>`

strcasestr

```
sub-string. int strcasestr(
    string haystack,
    string needle
)
```

Locate first occurrence of a case insensitive.

Description

The `strcasestr()` primitive finds the first occurrence of the case insensitive sub-string `needle` in the string `haystack`.

Parameters

`haystack` String object to be searched.
`needle` String to be matched.

Return

Index of first matching character starting at the index one, otherwise zero if no match.

Portability

A **GriefEdit** extension.

See Also

`strstr`, `strrstr`, `index`, `rindex`

strcmp

```
int strcmp(string s1,
          string s2,
          [ int length])
```

String compare.

Description

The `strcmp()` primitive shall compare the string `s1` to the string `s2` not ignoring case.

Parameters

`s1` First string.
`s2` Second string to compare against.
`length` Optional, when specified only the first `length` characters of both string shall be compared.

Return

`strcmp()` shall return an integer greater than, equal to, or less than 0, if the string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2`, respectively.

Portability

A **GriefEdit** extension.

See Also

`strcasestr`, `<=>`

strfilecmp

```
int strfilecmp(string file1,
               string file2,
               [int length])
```

Filename comparison.

Description

The *strfilecmp()* primitive lexicographically compares the filenames *name1* and *name2* and returns a value indicating their relationship, taking into account any filesystem implementation specifics.

If specified *length* defines the number of significant characters of each path which shall be compared, ignoring any characters after the length characters of each.

Notes

Under DOS, Windows and OS/2 this primitive is case insensitive and permits the intermixing of both back(\) and forward(/) slashes as directory delimiters.

Under Unix™ this primitive is the same as an equality (==) operation between two strings.

Returns

The return indicates the lexicographic relation of *name1* and *name2*.

```
<0 - name1 less than name2.  
0 - name1 identical to name2.  
>0 - name1 greater than name2.
```

Portability

A **GriefEdit** extenions.

See Also

[strcmp](#)

strftime

```
string strftime([string format = NULL],
                [int time = NULL] )
```

Format time and date.

Description

The *strftime()* primitive is an interface to the system library function *strftime*. Unless *time* is specified, the current time shall be formatted otherwise the stated time shall be formatted.

The *format* specification is a string and may contain special character sequences called conversion specifications, each of which is introduced by a % character and terminated by some other character known as a conversion specifier character. All other character sequences are ordinary character sequences.

Conversion specifications

Most implementations support the following conversion specifications.

- %a is replaced by the locale's abbreviated weekday name.
- %A is replaced by the locale's full weekday name.
- %b is replaced by the locale's abbreviated month name.
- %B is replaced by the locale's full month name.
- %c is replaced by the locale's appropriate date and time representation.
- %C is replaced by the century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99].
- %d is replaced by the day of the month as a decimal number [01,31].
- %D same as %m/%d/%y.
- %e is replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space.
- %h same as %b.
- %H is replaced by the hour (24-hour clock) as a decimal number [00,23].
- %I is replaced by the hour (12-hour clock) as a decimal number [01,12].
- %j is replaced by the day of the year as a decimal number [001,366].
- %m is replaced by the month as a decimal number [01,12].
- %M is replaced by the minute as a decimal number [00,59].

%n is replaced by a newline character.
 %p is replaced by the locale's equivalent of either a.m. or p.m.
 %r is replaced by the time in a.m. and p.m. notation; in the POSIX locale this is equivalent to %I:%M:%S %p.
 %R is replaced by the time in 24 hour notation (%H:%M).
 %S is replaced by the second as a decimal number [00,61].
 %t is replaced by a tab character.
 %T is replaced by the time (%H:%M:%S).
 %u is replaced by the weekday as a decimal number [1, 7], with 1 representing Monday.
 %U is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number [00,53].
 %V is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01, 53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1.
 %w is replaced by the weekday as a decimal number [0, 6], with 0 representing Sunday.
 %W is replaced by the week number of the year (Monday as the first day of the week) as a decimal number [00, 53]. All days in a new year preceding the first Monday are considered to be in week 0.
 %x is replaced by the locale's appropriate date representation.
 %X is replaced by the locale's appropriate time representation.
 %y is replaced by the year without century as a decimal number [00,99].
 %Y is replaced by the year with century as a decimal number.
 %Z is replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.
 %% is replaced by %.

Returns

The `strftime` function returns a string containing the formatted time and date.

Portability

A **GriefEdit** extension.

See Also

`time`, `cftime`, `date`, `stat`, `localtime`, `gmtime`

string_count

```
int string_count(    string      haystack,
                     int needle|string needles  )
```

Count occurrences of characters in a string.

Description

The `string_count()` primitive computes the number of occurrences of the character(s) within `needles` in the string `haystack`.

Parameters

haystack String to be searched.
 needle Elements to the counted, each character shall be accumulated.

Returns

Returns the number of times the characters in `needle` occur in the string parameter.

Portability

The integer needle form is a **GriefEdit** extension.

See Also

`substr`, `rindex`, `index`

strpbrk

```
int strpbrk(string str,
            string characters)
```

Search a string for any of a set of characters.

Description

The `strpbrk()` primitive returns the index of the first character in `str` which matches any character from the string

characters.

This function is like *index()* and *rindex()* yet these only matches a single character string rather than a complete sub-string.

Return

Index of first matching character starting at the index one, otherwise zero if no match.

Portability

A **GriefEdit** extension.

See Also

[firstof](#), [lastof](#)

strup

```
string strup(string str,
             [int length = 1])
```

Pop the leading character(s).

Description

The *strup()* primitive is equivalent to *substr(sym, 1, length)* with the additional functionality that the returned character is removed from the specified string *str*.

Portability

length is a **GriefEdit** extension.

Return

String value containing the first character(s) of the original value of *sym*.

See Also

[substr](#), [characterat](#)

strrstr

```
int strrstr(string haystack,
            string needle    )
```

Locate last occurrence of a sub-string.

Description

The *strrstr()* primitive finds the last occurrence of the sub-string *needle* in the string *haystack*.

Parameters

haystack String object to be searched.

needle String to be matched.

Return

Index of last matching character starting at the index one, otherwise zero if no match.

Portability

A **GriefEdit** extension.

See Also

[strstr](#), [index](#), [rindex](#)

strstr

```
int strstr(string haystack,
           string needle    )
```

Locate first occurrence of a sub-string.

Description

The *strstr()* primitive finds the first occurrence of the sub-string *needle* in the string *haystack*.

Parameters

haystack String object to be searched.
needle String to be matched.

Return

Index of first matching character starting at the index one, otherwise zero if no match.

Portability

A **GriefEdit** extension.

See Also

[strcasestr](#), [strstr](#), [index](#), [rindex](#)

strtod

```
int strtod(string str,
           [int &endoffset])
```

String to double.

Description

The *strtod()* primitive converts the initial portion of the string *str* to a floating point double representation; it is an interface to the standard library function of the same name.

Returns

Upon successful completion, *strtod()* returns the converted value, if any, and (if supplied) an index (base of 1) to the first unprocessed character within the string is stored in the integer object *endoffset*.

If no conversion could be performed, *strtod()* returns 0 and *errno* may be set to **EINVAL**. The subject sequence contains no characters and index of 0 is stored in *endoffset*.

If the correct value is outside the range of representable values, *strtod()* returns **HUGE_MAX** or **HUGE_MIN** and *errno* is set to **ERANGE**.

Portability

A **GriefEdit** extension.

See Also

[atoi](#), [strtod](#), [sscanf](#)

strtod

```
int strtod(string str,
           [int &endoffset])
```

String to float.

Description

The *strtod()* primitive converts the initial portion of the string *str* to a floating point representation; it is an interface to the standard library function of the same name.

Returns

Upon successful completion, *strtod()* returns the converted value, if any, and (if supplied) an index (base of 1) to the first unprocessed character within the string is stored in the integer object *endoffset*.

If no conversion could be performed, *strtod()* returns 0 and *errno* may be set to **EINVAL**. The subject sequence contains no characters and index of 0 is stored in *endoffset*.

If the correct value is outside the range of representable values, *strtod()* returns **FLOAT_MAX** or **FLOAT_MIN** and *errno* is set to **ERANGE**.

Portability

A **GriefEdit** extension.

See Also

[atoi](#), [strtod](#), [sscanf](#)

strtol

```
int strtol(string str,
           [int &endoffset],
           [int base]      )
```

Convert a string into its numeric value.

Description

The *strtol()* primitive converts the initial portion of the string *str* to a type integer representation; it is an interface to the standard library function of the same name.

If the value of *base* is 0 (or if not supplied), the expected form of the subject sequence is that of a decimal constant, octal constant or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix 0 optionally followed by a sequence of the digits '0' to '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters a (or A) to f (or F) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a + or - sign. The letters from a (or A) to z (or Z) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it is used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

Returns

Upon successful completion, *strtol()* returns the converted value, if any, and (if supplied) an index (base of 1) to the first unprocessed character within the string is stored in the integer object *endoffset*.

If no conversion could be performed, *strtol()* returns 0 and *errno* may be set to **EINVAL**. The subject sequence contains no characters and index of 0 is stored in *endoffset*.

If the correct value is outside the range of representable values, *strtol()* returns **LONG_MAX** or **LONG_MIN** and *errno* is set to **ERANGE**.

Portability

A **GriefEdit** extension.

See Also

[atoi](#), [itoa](#), [strtod](#), [strtof](#), [sscanf](#)

strverscmp

```
int strverscmp(string s1,
               string s2 )
```

Version string compare.

Description

strverscmp(3)/versionsort(3) style version comparison function.

Often one has files jan1, jan2, ..., jan9, jan10, ... and it feels wrong when ls orders them jan1, jan10, ..., jan2, ..., jan9. In order to rectify this, GNU introduced the -v option to ls(1), which is implemented using *versionsort(3)*, which again uses *strverscmp*.

Thus, the task of *strverscmp* is to compare two strings and find the "right" order, while *strcmp* only finds the lexicographic order.

Return

The *strverscmp()* primitive returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be earlier than, equal to, or later than *s2*.

Portability

A **GriefEdit** extension.

See Also

[strcmp](#), [strcasecmp](#), ==

substr

```
string substr(string str,
             [int offset],
             [int length] )
```

Extract a sub-string.

Description

The *substr()* primitive extracts parts of a string, beginning at the character at the specified position, and returns the specified number of characters.

The *substr()* primitive does not change the original string.

Parameters

`offset` The position where to start the extraction. First character is at index 1.
`length` Optional, the number of characters to extract.

Returns

Returns the sub-string of string which starts at start, and goes on for length characters, or the end of the string if length is omitted.

See Also

`index`, `rindex`

tokenize

```
list tokenize(string expr,
             string delims,
             int     flags,
             [string whitespace = "\t\n\r"])
```

Tokenize a string into token elements.

Description

The *tokenize()* primitive tokenizes the string *expr* into a list of strings and returns the list in list context.

tokenize() provides greater field processing than the simpler `split` primitive and should be used by new macros.

Parameters

`expr` String to be tokenize.
`delims` is a string consisting of one or more characters which indicate the delimiter characters.
`flags` is an integer containing a set of flags which indicate how the input string is to be tokenized. Flags consist of one or more the *tokenize flags* detailed below OR'ed together.
`whitespace` Optional set of characters to be treated as whitespace.

Tokenize flags**General**

- **TOK_COLLAPSE_MULTIPLE** - Splits the string *expr* into a list of strings and returns the list in list context. Collapses occurrences of the repeated delimiter characters treating them as single delimiter, in other words empty elements with the delimited text shall not be returned.

Numeric field conversion

- **TOK_NUMERIC** - Fields which begin with a digit are converted into their decimal numeric value and returned as integer element rather than a string.
- **TOK_NUMERIC_STRTOL** - Numeric fields are converted using `strtol` allowing support leading base specifications hexadecimal (0x), octal (0) and binary (0b).
- **TOK_NUMERIC_STRICT** - Strict conversion of numeric fields where by any invalid values, for example trailing non-numeric characters, result in the the field being returned as a string element and not a integer element.

Parsing options

- **TOK_WHITESPACE** - Allow leading and trailing whitespace around quoted and numeric element.
- **TOK_BACKSLASHES** - Allow backslashes to escape the meaning of any delimiter characters and both single and double.
- **TOK_ESCAPE** - Enable backslash escape sequence processing.
- **TOK_ESCAPEALL** - Control the behaviour of **TOK_ESCAPE** to escape all characters preceded with a backslashes, otherwise by default unknown escape sequences are ignored.

Quote options

- **TOK_DOUBLE_QUOTES** - Enables double quote support where all characters enclosed within a pair of

matching quotes are treated as a single element including any embedded delimiters.

- **TOK_DOUBLE_QUOTES** - Same as **TOK_DOUBLE_QUOTES** but enables single quote support.
- **TOK_QUOTE_STRINGS** - When single or double quoted support is enabled allow the element to be enclosed within an extra pair of quotes, for example

```
""hello world""
```

- **TOK_PRESERVE_QUOTES** - When an element is enclosed in quotes and the quote character is specified in *delims* then the returned element shall also be enclosed within the encountered quotes.

Field Processing Options

- **TOK_TRIM.LEADING** - Remove any leading whitespace from non-quoted string elements. Whitespace is defined as any space, tab or newline character unless they exist within the set of specified delimiters.
- **TOK_TRIM.TRAILING** - Remove any trailing whitespace from string elements.
- **TOK_TRIM** - Remove any leading and trailing whitespace characters.
- **TOK_TRIM.QUOTED** - Apply trim logic to quoted strings.

Return

The *tokensize()* primitive returns a list of the words and/or numeric values as encountered within the string *str*.

Portability

Many of the features are **GriefEdit** specific; *CRISP™* has a similar primitive yet as the two were developed independently features differ.

See Also

[split](#), [sscanf](#), [index](#), [substr](#)

trim

```
string trim(string str,
           [string chars = " \\\t\\\r\\\n"])
```

Chomp characters from a string.

Description

The *trim()* primitive removes leading and trailing characters from the specified *string*.

The default is to remove all tabs, spaces and newline characters. If *chars* is specified, then all characters within the *trimchar* string are removed from the beginning and end of the string.

Parameters

str String object to be trimmed.
chars Optional string defining the set of characters to be removed.

Returns

Returns a copy of string with all leading and trailing white space characters removed. (spaces, tabs and newlines by default).

Portability

The *chars* and removal of trailing characters in addition to leading is a **GriefEdit** extension (See: [rtrim](#)).

See Also

[compress](#), [ltrim](#), [rtrim](#)

upper

```
string/int upper(string str|int character)
```

Convert string or character to uppercase.

Description

The *upper()* primitive converts all alphabetic characters within the string object *str* or just the specified character *ch* to uppercase.

Returns

Returns the specified object with all alphabetic characters converted to uppercase.

If the argument is a string then a copy of the string is returned, otherwise the integer value of the converted

character.

Portability

Character support is a **GriefEdit** extension.

See Also

[lower](#)

ctype

Character classes

Description

The **GriefEdit** *ctype* (character class) functionality is used to designate character-coded integer values into one or more character types.

Standard character classes are;

- alnum - An alphanumeric (letter or digit).
- alpha - A letter.
- ascii - An ASCII character (ie $0 > x < 127$).
- blank - A space or tab character.
- cntrl - A control character.
- csym - A symbol.
- digit - A decimal digit.
- graph - A character with a visible representation.
- lower - A lower-case letter.
- print - An alphanumeric (same as alnum).
- punct - A punctuation character.
- space - A character producing white space in displayed text.
- upper - An uppercase letter.
- xdigit - A hexadecimal digit.

Support

The following **GriefEdit** interfaces have direct support for character-classes:

Bracket expressions

Within **sscanf** and search <regexp> expressions, the name of a character class enclosed in [: and :] stands for the list of all characters (not all collating elements!) belonging to that class.

Macros

The set of the **isaxxxx(object, [index])** primitive exist allowing tests within macros. Each of these subroutines returns a nonzero value if the specified value is contained within the given class, otherwise zero.

The specified argument *object* can be an integer (whose ASCII code represents a single character), or a string, in which case the first character of the string is tested. The optional index allows an alternative character within the string to be tested, starting at offset one being the first character.

Note:

The **isaxxx()** primitives should only be used on character data that can be represented by a single byte value (0 through 255). Attempting to use the *ctype* subroutines on multi-byte locale data may give inconsistent results.

See Also

[sscanf](#), [<regexp>](#), [isalnum](#), [isalpha](#), [isascii](#), [iscntrl](#), [iscsym](#), [isdigit](#), [isgraph](#), [islower](#), [isprint](#), [ispunct](#), [isspace](#), [isupper](#), [isxdigit](#)

[\\$Id: \\$](#)

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Syntax Highlighting Primitives

Summary

Syntax Highlighting Primitives

Macros

attach_syntax	Attach a syntax to a buffer.
create_syntax	Syntax table creation.
define_keywords	Add keywords to a syntax dictionary.
detach_syntax	Detach a syntax from a buffer.
get_color_pair	Retrieve the specific color.
hilite_create	Create a hilite resource.
hilite_delete	Delete a hilite resource.
hilite_destroy	Destroy hilite resources.
inq_hilite	Retrieve a hilite definition.
inq_syntax	Retrieve the syntax identifier.
set_color_pair	Set a specific color.
set_syntax_flags	Set syntax flags.
syntax_build	Build a syntax hiliting engine.
syntax_column_ruler	Column syntax coloriser.
syntax_rule	Define a syntax hilite rule.
syntax_token	Define a syntax token.

Macros

attach_syntax

```
int attach_syntax(int|string syntable)
```

Attach a syntax to a buffer.

Description

The *attach_syntax()* primitive associates the current buffer with the syntax table specified by the name *table*.

Until another syntax table is associated with the buffer, the syntax table *syntable* shall be used in all operations that require a syntax; these include parenthesis matching and spell-checks.

Parameters

syntable Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Returns

The *attach_syntax()* primitive returns the syntax identifier associated with the attached syntax table, otherwise -1 if table did not exist.

On invalid table reference errors the following diagnostics message(s) shall be echoed on the command prompt.

```
syntax: table 'xxx' undefined.
```

```
syntax: no current table.
```

Portability

A **GriefEdit** extension.

See Also

[create_syntax](#), [detach_syntax](#), [inq_syntax](#)

create_syntax

```
int create_syntax(string table)
```

Syntax table creation.

Description

The `create_syntax` primitive creates a new syntax table with the name specified by `table`. If the table already exists, the existing table shall be reinitialised.

Parameters

`table` Unique syntaxtable name.

Returns

The `create_syntax` primitive returns the syntax identifier associated with the syntax table, otherwise -1 if the syntax could not be created.

Portability

A **GriefEdit** extension.

See Also

`attach_syntax`, `detach_syntax`, `inq_syntax`

define_keywords

```
void define_keywords(      [int|string] keywords,
                           string words|list  words,
                           [int length],
                           [int flags],
                           [int|string syntable])
```

Add keywords to a syntax dictionary.

Description

The `define_keywords()` primitive adds a set of keywords to the specified dictionary, which shall be color syntax highlighted in the associated color with the table specified by `table`.

Description

`keywords` Keyword table identifier, see table below.

`words` List of words, otherwise a string that is the concatenation of keywords each being of absolute `length` characters, optionally comma separated (See: <length>).

`length` Length of the keywords. If positive the keyword string is assumed to contain non delimited words of all of same length. Otherwise if given as a negative value, the keyword string is assumed to contain comma separated values of variable length words.

`flags` Optional control flags.

`SYNF_IGNORECASE` Ignore case.

`SYNK_NATCHCASE` Match case.

`SYNF_PATTERN` Pattern match (glob style).

`syntable`Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Keyword Tables

Constant	Name	Attribute
<code>SYNK_PRIMARY</code>	<code>primary</code>	<code>ATTR_KEYWORD</code>
<code>SYNK_FUNCTION</code>	<code>function</code>	<code>ATTR_KEYWORD_FUNCTION</code>
<code>SYNK_EXTENSION</code>	<code>extension</code>	<code>ATTR_KEYWORD_EXTENSION</code>
<code>SYNK_TYPE</code>	<code>type</code>	<code>ATTR_KEYWORD_TYPE</code>
<code>SYNK_STORAGECLASS</code>	<code>storageclass</code>	<code>ATTR_KEYWORD_STORAGECLASS</code>
<code>SYNK_DEFINITION</code>	<code>definition</code>	<code>ATTR_KEYWORD_DEFINTION</code>
<code>SYNK_CONDITIONAL</code>	<code>conditional</code>	<code>ATTR_KEYWORD_CONDITIONAL</code>
<code>SYNK_REPEAT</code>	<code>repeat</code>	<code>ATTR_KEYWORD_REPEAT</code>
<code>SYNK_EXCEPTION</code>	<code>exception</code>	<code>ATTR_KEYWORD_EXCEPTION</code>
<code>SYNK_DEBUG</code>	<code>debug</code>	<code>ATTR_KEYWORD_DEBUG</code>
<code>SYNK_LABEL</code>	<code>label</code>	<code>ATTR_KEYWORD_LABE</code>
<code>SYNK_STRUCTURE</code>	<code>structure</code>	<code>ATTR_KEYWORD_STRUCTURE</code>
<code>SYNK_TYPEDEF</code>	<code>typedef</code>	<code>ATTR_KEYWORD_TYPEDEF</code>
<code>SYNK_CONSTANT</code>	<code>constant</code>	<code>ATTR_CONSTANT</code>
<code>SYNK_OPERATOR</code>	<code>operator</code>	<code>ATTR_OPERATOR</code>
<code>SYNK_BOOLEAN</code>	<code>boolean</code>	<code>ATTR_BOOLEAN</code>

Constant	Name	Attribute
SYNK_PREPROCESSOR	preprocessor	ATTR_PREPROCESSOR_KEYWORD
SYNK_PREPROCESSOR_INCLUDE	ppinclude	ATTR_PREPROCESSOR_INCLUDE
SYNK_PREPROCESSOR_DEFINE	ppdefine	ATTR_PREPROCESSOR_DEFINE
SYNK_PREPROCESSOR_CONDITIONAL	ppconditional	ATTR_PREPROCESSOR_CONDITIONAL
SYNK_TODO	todo	ATTR_TODO
SYNK_MARKUP	markup	ATTR_COMMENT_STANDOUT

Returns

nothing

PortabilityA **GriefEdit** extension.**See Also**[create_syntax](#)**detach_syntax**

```
void detach_syntax()
```

Detach a syntax from a buffer.

DescriptionThe *detach_syntax()* primitive removes the associated syntax definition from the current buffer.**Parameters**

none

ReturnsThe *detach_syntax()* primitive returns the syntax identifier which was associated with the buffer, otherwise -1 if no syntax was associated.**Portability**A **GriefEdit** extension.**See Also**[create_syntax](#), [attach_syntax](#), [inq_syntax](#)**get_color_pair**

```
void get_color_pair(string name|int ident,
                    [int|string fg],
                    [int|string bg],
                    [int|string sf] )
```

Retrieve the specific color.

DescriptionThe *get_color_pair()* primitive retrieves a specific attribute that GRIEF utilities on a color display. Attributes may be specified as integers or strings, with strings being case-insensitive see [set_color](#) for more details.The specified attribute color values shall be assigned to the specified arguments *foreground*, *backbound* and *style*.**Parameters**

- ident Attribute identifier.
- fg Optional variable reference to be populated with the foreground color. If an integer reference the numeric colour value is assigned otherwise if a string reference the associated name.
- bg Optional variable reference to be populated with the background color. If an integer reference the numeric colour value is assigned otherwise if a string reference the associated name.
- sf Optional variable reference to be populated with the style and flags. If an integer reference the numeric value is assigned otherwise when a string reference the human readable decoding version is assigned.

Returns

nothing

PortabilityA **GriefEdit** extension.

See Also

[set_color_pair](#), [set_color](#), [get_color](#)

hilite_create

```
int hilite_create( [int bufnam],
                   [int type],
                   [int timeout],
                   [int sline],
                   [int scol],
                   [int eline],
                   [int ecol],
                   [string | int attr],
                   [int ident] )
```

Create a hilite resource.

Description

The *hilite_create()* primitive creates a buffer hilite resource with the referenced buffer *bufnum* under the classification *type*; the given classification groups hilite resources allowing bulk management and removal.

Similar to the buffer anchors yet they can not be edited and there maybe as many as desired hilite's are available for use by macros to mark elements within documents, for example search results.

The created resource decorators the region for the duration *timeout* between the stated starting position *sline*, *eline* upto the ending position *eline*, *ecol* using the display attribute *attr*.

Parameter

<i>bufnum</i>	Optional buffer number, if omitted the current buffer shall be referenced.
<i>type</i>	Optional type, default 0; user assignable label.
<i>timeout</i>	Specifies a timeout in seconds. If specified then the hilite shall be automatically deleted upon the timeout expiring. A timeout of -1 implies on next buffer change.
<i>sline</i> , <i>scol</i>	Start of the hilite region.
<i>eline</i> , <i>ecol</i>	End for the hilite region.
<i>attr</i>	Associated attribute.
<i>ident</i>	User assigned identifier.

Returns

The *hilite_create()* primitive returns the unique hilite identifier, otherwise -1 on error.

Portability

A **GriefEdit** extension, yet it was noted similar functionality has been introduced to *CRISP™* in parallel; compatibility as yet confirmed.

See Also

[hilite_destroy](#), [inq_hilite](#)

hilite_delete

```
int hilite_delete( [int bufnum],
                   int hilite )
```

Delete a hilite resource.

Description

The *hilite_delete()* primitive removes the stated hilite's *hilite* from the associated with the buffer *bufnum*.

Parameter

<i>bufnum</i>	Optional buffer number, if omitted the current buffer shall be referenced.
<i>hilite</i>	Unique hilite identifier which are returned during the hilite's corresponding creation by hilite_create .

Returns

The *hilite_delete()* primitive returns 1 if the hilite existed and was removed successfully, 0 if the hilite did not exist, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[hilite_create](#), [hilite_destroy](#), [inq_hilite](#)

hilite_destroy

```
int hilite_destroy([int bufnum],
                  [int type] )
```

Destroy hilite resources.

Description

The *hilite_destroy()* primitive either removes hilite's of the specified *type* otherwise if omitted all hilite's associated with the buffer *bufnum*.

Parameter

bufnum Optional buffer number, if omitted the current buffer shall be referenced.
type Optional hilite type, if omitted all buffer hilite's are released.

Returns

The *hilite_destroy()* primitive returns the number of hilite's removed, 0 if none were found, otherwise -1 on error.

Portability

A **GriefEdit** extension, yet it was noted similar functionality has been introduced to *CRISP™* in parallel; compatibility as yet confirmed.

See Also

[hilite_create](#), [inq_hilite](#)

inq_hilite

```
int inq_hilite( [int bufnum],
                 [int line],
                 [int column],
                 [int &attribute],
                 [int &ident] )
```

Retrieve a hilite definition.

Description

The *inq_hilite()* primitive retrieves details about the hilite resource at the specified position.

Parameters

bufnum Optional buffer number, if omitted the current buffer shall be referenced.
line Optional integer line number within the referenced buffer, if omitted the current line number is used.
column Optional integer column number within the referenced buffer, if omitted the current column number is used.
attribute Optional integer variable, if specified shall be populated with the hilite assigned attribute.
indent Optional integer variable, if specified shall be populated with the hilite user assigned user identifier.

Returns

The *inq_hilite()* primitive returns the type of the active hilite and populates *attribute* and *indent*, otherwise -1 and the arguments shall remain unmodified..

Portability

A **GriefEdit** extension, yet it was noted similar functionality has been introduced to *CRISP™* in parallel; compatibility as yet confirmed.

See Also

[hilite_create](#), [hilite_destroy](#)

inq_syntax

```
int inq_syntax([int &flags],
              [int|string syntable])
```

Retrieve the syntax identifier.

Description

The *inq_syntax()* primitive retrieves the syntax identifier associated with the specified syntax.

Parameters

`flags` Option integer reference, to be populated with the active flags of the referenced syntax-table.
`syntable` Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Returns

The `inq_syntax()` primitive returns the syntax-table identifier, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[create_syntax](#), [set_syntax_flags](#)

set_color_pair

```
void set_color_pair( string|int ident,
                     [int|string fg],
                     [int|string bg],
                     [int|string sf] )
```

Set a specific color.

Description

The `set_color_pair()` primitive sets the pair of foreground and background colors associated with the color attribute `indent`.

The specified attribute shall be assigned the given foreground color `fg`, with an optional background `bg` and style `sf`. If the foreground is omitted the user shall be prompted.

Parameters

`ident` Attribute identifier either using its their manifest integer constants or string aliases, with names being case-insensitive; see [set_color](#) for details.
`fg` Optional foreground color. If omitted, then the foreground and (if required) background are prompted.
`bg` Optional background color.
`sf` Optional style and flags.

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[set_color](#), [get_color_pair](#), [get_color](#)

set_syntax_flags

```
int set_syntax_flags(int flags,
                     [int|string syntable])
```

Set syntax flags.

Description

The `set_syntax_flags()` primitive sets the active flags for the specified syntax table.

Parameters

`flags` Integer syntax flags, one or more of the following flags OR'ed together control the attributes of the reference syntax table.
`syntable` Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Flags

Flag	Description
<code>SYNF_CASEINSENSITIVE</code>	Case insensitive language tokens.
<code>SYNF_FORTRAN</code>	FORTRAN style language.
<code>SYNF_STRING_ONELINE</code>	String definitions dont continue over line breaks.
<code>SYNF_LITERAL_NOQUOTES</code>	Literals dont quote.
<code>SYNF_COMMENTS.LEADINGWS</code>	Dont hilite leading white-space.
<code>SYNF_COMMENTS.TRAILINGWS</code>	Dont hilite trailing white-space.

Flag	Description
SYNF_COMMENTS_QUOTE	Allow comment character to be quoted.
SYNF_COMMENTS_CSTYLE	C-style comments.
SYNF_PREPROCESSOR_WS	Dont hilite leading white-space.
SYNF_LINECONT_WS	Allow white-space after cont token.
SYNF_HILITE_WS	Hilite white-space.
SYNF_HILITE_LINECONT	Hilite line continuations.
SYNF_HILITE_PREPROCESSOR	Hilite preprocessor directives.
SYNF_SPELL_WORD	Enable word spell check.
SYNF_SPELL_COMMENT	Enable comment spell check.

Returns

The `set_syntax_flags()` primitive returns the value of the resulting flags, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[create_syntax](#), [inq_syntax](#)

syntax_build

```
void syntax_build( [int timestamp],
                   [string cache],
                   [int|string syntable])
```

Build a syntax hilitng engine.

Description

The `syntax_build()` primitive constructs the underlying Deterministic Finite Automata (DFA) from the current set of defined rule via the [syntax_rule](#) primitive.

Parameters

<code>timestamp</code>	Optional numeric time reference, utilised to time-stamp the cache (See: time); should be modified upon each change to the DFA scheme.
<code>cache</code>	Optional name of the cache file image.
<code>syntable</code>	Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Returns

`nothing`

Portability

A **GriefEdit** extension.

See Also

[create_syntax](#), [attach_syntax](#), [detach_syntax](#), [inq_syntax](#), [syntax_rule](#)

syntax_column_ruler

```
int syntax_column_ruler( list ruler,
                         [string attribute],
                         [int|string syntable])
```

Column syntax coloriser.

Description

The `syntax_column_ruler()` primitive sets the column originated syntax coloriser.

Parameters

<code>ruler</code>	Ruler specification, represented by a set of increasing integer columns plus an optional string containing an attribute name (See: set_color). If a columns trailing attribute is omitted then the <code>default_attr</code> argument is applied. A NULL ruler clears the current ruler.
<code>default_attr</code>	Optional default attribute specification, if omitted "hilite" is assumed.
<code>syntable</code>	Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Returns

The `syntax_column_ruler()` primitive returns the length of the resulting ruler, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

`create_syntax`, `attach_syntax`, `detach_syntax`, `inq_syntax`

syntax_rule

```
void syntax_rule( string pattern,
                  string attribute,
                  [int|string syntable])
```

Define a syntax hilite rule.

Description

The `syntax_rule()` primitive pushes a Deterministic Finite Automata (DFA) rule into the enhanced highlighting definition for the syntax table specified by `syntable`.

The rule is described by the regular expression contained within the string `rule`, and the associated `attribute` is then applied against any matched constructs.

These rules work alongside the basic syntax elements declared by `syntax_token` against the same syntax table.

For example, the rules to highlight floating point numbers could be encoding as;

```
syntax_rule("[0-9]+\\.[0-9]*([Ee][+-]?[0-9]*)?[fFLL]?[iIjJ]?", "float");
syntax_rule("[0-9]+[Ee][+-]?[0-9]*[fFLL]?[iIjJ]?", "float");
```

Note:

Consult the numerous bundled language mode definitions for working examples.

Parameters

`pattern` Rule regular expression.
`attribute` Attribute specification.
`syntable` Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Attribute Specification

The attribute specification takes the following form. None or comma separated options plus the associated colour attribute (See: `set_color`).

```
[<option>[="...."] [, <option> ...] :] <attribute>
```

Options

- word - possible word.
- keyword - possible keyword.
- tags - possible symbol within tagdb.
- directive - possible preprocessor directive.
- preproc/pp - enter preprocessor mode.
- quick - quick expression evaluation. marks the regular expression for minimal closure, by default evaluation matches against the longest possible rule, quick short-circuits expression execution upon being matched, reducing the greedy nature of DFA regular expressions.
- spell - apply spell checks.
- todo - apply TODO checks.
- markup - apply markup checks.
- silent - silent regarding issues, for example non-existent children.
- name=<name> - rule name.
- group=<grpname> - group name, implied sub-rule.
- color=<spec> - color specification, implies the creation of the attribute if it does not exist.
- contains=<rule> - contains one or more rules.

- contained - is contained within another rule.

Examples

Comment block, with both spelling and TODO token processing enabled.

```
"spell,todo:comment"
```

Normal text, yet token may be either a keyword or directive.

```
"keyword,directive:normal"
```

Rule Regular Expression

A regular expression is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs. For a brief introduction, see .NET Framework Regular Expressions.

Each section in this quick reference lists a particular category of characters, operators, and constructs that you can use to define regular expressions:

Character Escapes

The backslash character (\) in a regular expression indicates that the character that follows it either is a special character (as shown in the following table), or should be interpreted literally.

Escape	Description
\t	Tab.
\n	Newline.
\r	Return.
\f	Formfeed.
\a	Alarm (bell, beep, etc).
\e	Escape (\027).
\\	This escapes the meaning of a special.

Character Classes

A character class matches any one of a set of characters. Character classes include the language elements listed in the following table.

Escape	Description
[...]	Matches any one character contained within the character sequence.
.	Match any single character except newline.
\d	Same as [0-9].
\x	Same as [a-zA-F0-9].
\s	Same as [\t\f].
\w	Same as [a-zA-Z_0-9].

Character Sequences

The conversion specification includes all subsequent characters in the format string up to and including the matching right square bracket ()].

The characters between the square brackets (the scanlist) comprise the scanset, unless the character after the left square bracket is a circumflex (^), in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right square bracket.

If the conversion specification begins with "[" or "[^]", the right square bracket is included in the scanlist and the next right square bracket is the matching right square bracket that ends the conversion specification; otherwise the first right square bracket is the one that ends the conversion specification.

If a hyphen character (-) is in the scanlist and is not the first character, nor the second where the first character is a circumflex (^), nor the last character, it indicates a range of characters to be matched. To include a hyphen, make it the last character before the final close bracket. For instance, '[^]0-9-]' means the set 'everything except close bracket, zero through nine, and hyphen'.

Within a bracket expression, the name of a character class enclosed in [: and :] stands for the list of all characters (not all collating elements!) belonging to that class.

alnum	An alphanumeric (letter or digit).
alpha	A letter.

blank	A space, tab or form-feed character.
cntrl	A control character.
digit	A decimal digit.
graph	A character with a visible representation.
lower	A lower-case letter.
print	An alphanumeric (same as alnum).
punct	A punctuation character.
space	A character producing white space in displayed text.
upper	An upper-case letter.
word	"word" character (alphanumeric plus "_").
xdigit	A hexadecimal digit.

Anchors

Anchors, or atomic zero-width assertions, cause a match to succeed or fail depending on the current position in the string, but they do not cause the engine to advance through the string or consume characters. The metacharacters listed in the following table are anchors.

Anchor	Description
^	If this is the first character of the regular expression, it matches the beginning of the line.
\$	If this is the last character of the regular expression, it matches the end of the line.
\c	Anchor start of the matched text to the proceeding token.

Quantifiers

A quantifier specifies how many instances of the previous element must be present in the input string for a match to occur.

Anchor	Description
*	Match the preceding character or range of characters 0 or more times.
+	Match the preceding character or range of characters 1 or more times.
?	Match the preceding character or range of characters 0 or 1 times.

Specials

Grouping constructs delineate subexpressions of a regular expression and typically capture substrings of an input string.

Anchor	Description
	This symbol is used to indicate where to separate two sub regular expressions for a logical OR operation.
(..)	Group boundaries.
\Q..\E	A section enclosed in these symbols is taken literally. Inside these sections, meta characters and special symbols have no meaning. If a \E needs to appear in one of these sections, the \\ must be escaped with \\.

Returns

nothing

Portability

A **GriefEdit** extension.

See Also

[create_syntax](#), [attach_syntax](#), [detach_syntax](#), [inq_syntax](#), [syntax_build](#)

syntax_token

```
void syntax_token( int type,
                   [<type1> param1],
                   [<type2> param2],
                   [int|string syntable])
```

Define a syntax token.

Description

The `syntax_token()` primitive adds and/or modifies a syntax tokeniser element of the table specified by the first parameter `table`. The actual type and number of parameters vary according to the second parameter `type`.

Parameters

<code>type</code>	Table attribute.
<code>param1</code>	First parameter.
<code>param2</code>	Optional second parameter.
<code>syntable</code>	Optional syntax-table name or identifier, if omitted the current syntax table shall be referenced.

Table Attributes

The following **SYNT** table attribute are available;

<code>SYNT_COMMENT</code>	<code><COMMENT>, <open-string> [, <close>-string>]</code> Comment syntax definition, defining either a block comment or an end-of-line comment. Block comments are specified as token pair, being an <code><open></code> and non new-line <code><close></code> strings, with end-of-line comments being a single <code><open></code> token.
<code>SYNT_CSTYLE</code>	<code><CSTYLE>, <character> <character-set></code>
<code>SYNT_PREPROCESSOR</code>	<code><PRE-PROCESSOR>, <character-set></code>
<code>SYNT_STRING</code>	<code><STRING>, <character-set></code>
<code>SYNT_LITERAL</code>	<code><LITERAL>, <character-set></code>
<code>SYNT_LINECONT</code>	<code><LINECONT>, <character></code>
<code>SYNT_LINEJOIN</code>	<code><LINEJOIN>, <character></code>
<code>SYNT_QUOTE</code>	<code><QUOTE>, <character-set></code>
<code>SYNT_CHARACTER</code>	<code><CHARACTER>, <character-set></code>
<code>SYNT_BRACKET</code>	<code><BRACKET>, <open> [, <close>]</code>
<code>SYNT_HTML</code>	<code><HTML>, <open>, <close></code>
<code>SYNT_TAG</code>	<code><TAG>, <type>, <word,word...></code>
<code>SYNT_WORD</code>	<code><WORD>, <character-set></code> Defines the character-set which represent a single word.
<code>SYNT_KEYWORD</code>	<code><KEYWORD>, <character-set></code> Defines the character-set which represent a single keyword.
<code>SYNT_NUMERIC</code>	<code><NUMERIC>, <primary-set> [, <secondary-set>]</code>
<code>SYNT_OPERATOR</code>	<code><OPERATOR>, <character></code>
<code>SYNT_DELIMITER</code>	<code><DELIMITER>, <character-set></code>
<code>SYNT_FORTRAN</code>	<code><FORTRAN>, <character-set>, <[left-margin], code [,comment-margin]></code>

Returns

`nothing`

Portability

A **GriefEdit** extension.

See Also

[create_syntax](#)

`$Id: $`

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Variable Declaration Primitives

Summary

Variable Declaration Primitives

Macros

<code>__lexicalblock</code>	Lexical scope.
<code>arg_list</code>	Argument list.
<code>array</code>	Declare a array symbol.
<code>bool</code>	Declare a boolean symbol.
<code>const</code>	Define a variable as being constant.
<code>declare</code>	Declare a polymorphic symbol.
<code>double</code>	Declare a double float symbol.
<code>extern</code>	Declare an external variable.
<code>float</code>	Declare a float symbol.
<code>get_parm</code>	Retrieve the value of a macro parameter.
<code> getopt</code>	Get options.
<code>getsubopt</code>	Parse suboption arguments from a string.
<code>global</code>	Declare a global variable.
<code>inq_symbol</code>	Determine if the symbol exists.
<code>int</code>	Declare an integer symbol.
<code>is_array</code>	Determine whether an array type.
<code>is_float</code>	Determine whether a float type.
<code>is_integer</code>	Determine whether an integer type.
<code>is_list</code>	Determine whether a list type.
<code>is_null</code>	Determine whether a NULL type.
<code>is_string</code>	Determine whether a string type.
<code>is_type</code>	Determine whether an explicit type.
<code>make_local_variable</code>	Make a buffer local variable.
<code>put_parm</code>	Assign an argument value.
<code>ref_parm</code>	Create a reference parameter.
<code>register</code>	Define a variable as being a register.
<code>static</code>	Define a function or module scope.
<code>string</code>	Declare a string symbol.
<code>typeof</code>	Determine the symbol type.

Macros

lexicalblock

```
__lexicalblock(.... block ...);
```

Lexical scope.

Description

The `__lexicalblock` statement is used to implement lexical block scoping within function/macro bodies.

By default local variables are all function scoped unless lexical block scoping is enabled via the `#pragma lexical_scoping`.

```
#pragma scope(push, lexical)
void function() {
    int i;           // first variable, function scope
    {
        int i;       // second variable, lexically block scope
    }
}
#pragma scope(pop)
```

Note:

The `__lexicalblock()` primitive is **internal** to the macro language. It is generated by the GRIEF macro compiler when nested variable declarations are encountered within block statements. It is not

intended for direct use.

Returns

nothing

See Also

[Scope](#)

arg_list

```
list arg_list( [int eval = FALSE],
               [int start = 0],
               [int end = -1] )
```

Argument list.

Description

The `arg_list()` primitive retrieves a list of the values representing the arguments which were passed to the current macro; allows for arbitrary argument processing.

Parameters

- `eval` Optional boolean flag, if **true** each argument shall be evaluated (e.g. variables are referenced) with the result being retrieved, otherwise the raw value (e.g. variable name) shall be retrieved.
- `start` Optional integer starting the index of the first argument to be including within the list. If omitted defaults to 1, being the first argument.
- `end` Optional integer starting the index of the last argument to be including within the list. If omitted defaults to -1, being the last argument.

Returns

The `arg_list()` primitive returns the arguments passed to the current macro as a list.

Examples

```
void
func()
{
    message("%s", arg_list());
}
```

In the example above, the list of arguments will be shown. But in a call like

```
int val = 99;
func(val);
```

In this case, the message will show "val" and not 99. To get the values replaced, use

```
message("%s", arg_list(1));
Return value
```

List of arguments passed to calling macro.

Portability

The `start` and `end` parameters are **GriefEdit** extensions.

See Also

[put_parm](#), [get_parm](#)

array

```
array sym1, sym2 ...;
```

Declare a array symbol.

Description

The `array` statement is reserved for future use.

An array is a variable array of fixed sized arbitrary values. Unlike a list, an array supports constant-time element access and updates.

Returns

nothing

Portability

n/a

See Also

[Types](#), [int](#), [string](#), [list](#), [float](#), [double](#)

bool

```
bool sym1, sym2 ...;
```

Declare a boolean symbol.

Description

The `bool` statement as an alias for `int`.

Returns

nothing

Portability

n/a

See Also

[Types](#), [int](#), [string](#), [list](#), [float](#), [double](#), [array](#)

const

```
const <type> sym1, sym2 ...;
```

Define a variable as being constant.

Description

The `const` qualifier explicitly declares a locally scoped data object as something that cannot be changed. Its an immutable variable can only be set during initialization.

You cannot use `const` data objects in expressions requiring a modifiable lvalue. If you try to give it a new value, it will return you an error.

For example, a `const` data object cannot appear on the lefthand side of an assignment statement.

Returns

nothing

Portability

An experimental GRIEF extension; functionality may change.

See Also

[Types](#), [global](#), [extern](#), [static](#), [register](#)

declare

```
declare sym1, sym2 ...;
```

Declare a polymorphic symbol.

Description

The `declare` statement declares a polymorphic data type.

Returns

nothing

Portability

n/a

See Also

[Types](#), [int](#), [string](#), [list](#), [float](#), [double](#), [array](#)

double

```
double sym1, sym2 ...;
```

Declare a double float symbol.

Description

The *double* statement is an alias for the [float](#) type.

Note:

Unlike C and C++, both float and *double* are internally represented using a 64-bit *double* precision floating-point

Returns

nothing

Portability

n/a

See Also

[Types](#), [int](#), [string](#), [list](#), [float](#), [array](#)

extern

```
extern <type> sym1, sym2, ...;
```

Declare an external variable.

Description

The *extern* storage class specifier extends the visibility of variables and functions, allowing objects and functions to be accessed across several source files.

An *extern* variable, function definition, or declaration makes the described variable or function usable by the succeeding part of the current source file.

This declaration does not replace the definition. The declaration is used to describe the variable that is externally defined. Essentially the *extern* keyword creates a place holder in the symbol table to avoid undefined symbol reference errors.

Note:

You should note that we are using the words definition and declaration carefully when we refer to external variables in this section. Definition refers to the place where the variable is created or assigned storage; declaration refers to places where the nature of the variable is stated but no storage is allocated.

Variables

A variable must be defined once in one of the modules of the program; this sets aside storage. If there is no definition, a runtime error shall result since the storage of the variable would not have been created,

There may be none or more variable declarations. These can be declared *extern* in many modules, including the module where it was defined, and even many times in the same module. All the declarations must match, which is normally by the use of a common header file shared between all source files.

Any *extern* declaration of the same identifier found within a block refers to that same object. If no other declaration for the identifier exists at file scope, the identifier has external linkage.

Unlike C external linkage of variables occurs during macro execution, see [Scope Rules](#). If a declaration for an identifier already exists in one of the visible namespaces they reference to the same image.

When searching for a variable definition, GRIEF searches the symbol tables in the following order:

- static variable definition in the current function.
- buffer local variable.
- local variables of a current block.
- nested stack frames to the outermost function call *dynamic scope*.

- global variable.

Within the following example the variable *x* reference by the function *foo* is resolved against the global image of *x*, whereas *b* shall be resolved against the image within the caller *bar*.

```
static int x = 0;

void
foo()
{
    extern int b, x;

    b = 0;           // resolved by 'i' within bar()
    x = 0;           // global 'x'
}

void
bar()
{
    int b;
    foo();
}
```

Functions

The *extern* statement applied to a function prototype does nothing; as *extern* is the default linkage. A function prototype is always a declaration and never a definition.

All functions across loaded macros which refer to the same external identifier refer to the same object, so care must be taken that the type and extent specified in the definition are compatible with those specified by each function which references the data. This is generally achieved by the use of a common header file shared between all source files (e.g. "grief.h").

It is an error to include a declaration for the same function with the storage class specifier *static* before the declaration with no storage class specifier because of the incompatible declarations. Including the *extern* storage class specifier on the original declaration is valid and the function has internal linkage.

Builtin macros do not have explicit prototypes as the Macro compiler has internal knowledge of all visible primitives.

Returns

nothing

Portability

n/a

See Also

[int](#), [string](#), [list](#), [float](#), [declare](#), [static](#), [global](#)

float

```
float sym1, sym2 ...;
```

Declare a float symbol.

Description

The *float* statement declares a simple type that stores 64-bit floating-point values that stores values in the approximate range;

```
1.7E308 to 1.7E+308
```

Note:

Unlike C and C++, both *float* and *double* are internally representing using a 64-bit double precision floating-point

Returns

nothing

Portability

n/a

See Also

[Types](#), [int](#), [string](#), [list](#), [double](#), [array](#)

get_parm

```
int get_parm(      [int argument],
                  declare &symbol,
                  [string prompt],
                  [int length = MAXPROMPT],
                  [declare default],
                  [int one = FALSE]      )
```

Retrieve the value of a macro parameter.

Description

The `get_parm()` primitive shall retrieve the value of the specified macro parameter *argument* optionally prompting the user if the referenced parameter was not given during the macro execution. When prompted the question within *prompt* is presented using the optional *default* which the user can then edit.

This function can also be used to always prompt the user for a reply by invoking with the argument parameter being omitted and specified as **NULL**.

Generally the user must complete the input using an *enter* unless the prompt is in *single character mode*, whereby the first key is taken as the input. The mode is either implied by the *length* parameter or an explicit *one* parameter, see below.

Navigation/actions Keys

The following keys bindings are active during parameter prompts.

Key	Function
Right, Left	Move cursor the back/forward one character.
Ctrl+Right, Ctrl+Left	Move cursor the start/end of the current word.
Home, End	Move to first/last character within the edit field.
Alt+I, Ctrl-O	Toggle insert/overstrike mode.
DEL	Delete character under the cursor.
Backspace, Ctrl+H	Delete character prior to the cursor.
Alt+K	Delete from cursor to the end of line.
Insert	Paste from scape.
Backspace, Ctrl+H	Delete character prior to the cursor.
ESC	Abort current edit, restoring original content.
Alt-D	Delete current line.
Alt-K	Delete from cursor to the end of line.
Alt+Q, Ctrl+Q	Quote next character.
Enter	Process change.
ALT+H	Help.
Ctrl+A (*)	Move cursor to beginning of line.
Ctrl+D (*)	Delete character under cursor.
Ctrl+K (*)	Delete from cursor to the end of line.
Ctrl+X, Ctrl+U (*)	Delete current line.
Ctrl+V, Ctrl+Y (*)	Paste from clipboard.

(*) Emacs style key mappings.

Note:

Arguments passed to macros are passed as call by name, that is every time a `get_parm` is issued against a particular parameter, that parameter is re-evaluated.

This lazy evaluation has a number of implications.

- The order of parameter evaluation is dependent on the `get_parm` execution order within the called macro, not the arguments position.
- Each parameter may be evaluated several times.
- Parameters may never be evaluated, which is again dependent on the logic placed around `get_parm` usage.

This feature can be very useful sometimes, and at other times it can cause anomalous side-effects ((see [Lazy Evaluation](#))).

Parameters

<code>argument</code>	An integer stating the associated macro argument index to be retrieved, starting at an offset of zero for the first parameter.
<code>symbol</code>	Specifies the symbol reference into which the resulting parameter shall be stored.
<code>prompt</code>	Option string which specifies the prompt which is represented to the user. If the prompt is omitted, the user is not prompted.
<code>length</code>	Optional integer parameter that specifies the upper limit of the string length that is to be retrieved. When given as <code>1</code> then single character mode is implied unless overridden using the <code>one</code> parameter. If omitted the upper length is assumed to be <code>MAXPROMPT</code> .
<code>default</code>	An optional value, whose type should match the type of <code>symbol</code> , if specified contains the value which shall initially be placed on the command line if a prompt is required.
<code>one</code>	Optional integer flag, if specified as non-zero than the user shall be prompted for a single character. Generally the user must complete the input using an <code>enter</code> , whereas in single character mode the first key is taken as the input. In addition, single character mode disables the execution of the <code>_prompt_begin</code> and <code>_prompt_end</code> callbacks plus any associated prompt history. If omitted the character mode shall be implied from the specified <code>length</code> ; a length of <code>1</code> being <code>true</code> otherwise <code>false</code> .

Returns

The `get_parm()` primitive returns greater than zero on success, otherwise zero if either the user aborted or an error was encountered.

Return	Description
0	Abort, invalid argument or conversion error.
1	Success.
2	Default was assigned (extension).

Portability

Unlike BRIEF the default parameter is always the fifth, whereas with BRIEF the default value is either the fourth or fifth dependent on whether an integer `length` is stated since the `default` was only permitted to be a string.

The `one` option is a **GriefEdit** extension.

See Also

`inq_prompt`, `_prompt_begin`, `_prompt_end`

getopt

```
int getopt(
    string value,
    [[string shortopts], list longopts, string|list args, [string caller]]
)
```

Get options.

Description

The `getopt()` primitive is a command-line parser similar to the system library function of the same name. The `getopt()` function provides a superset of the functionality of `getopt`, accepting options in two forms, words (long-options) and characters (short-options).

The `getopt()` primitive is designed to be executed within a loop, with the first invocation supplying the available options `shortopts` and/or `longopts`, the arguments to be parsed `args` and the application/macro name `caller`. Subsequent calls within the same execution loop only then request the next available option without additional arguments; the general form of `getopt` usage is as follows.

```
string value;
int ch;

if ((ch = getopt(value, shortopts, longopts, args)) > 0) {
    do {
        } while ((ch = getopt(value)) > 0);
```

Short Options

The short option string `shortopts` may contain the following elements

- individual characters
- characters followed by a colon to indicate an option argument is to follow.

For example, an option string "x" recognizes an option '-x', and an option string "x:" recognizes an option and

argument ` -x argument'.

Long Options

The long option list *longopts* may contain a list of strings each defining a long option, of the form:

```
"[tag] [, [#]value] [[:=] [scinfd]]"
```

tag	Option tag or name.
value	Index value returned upon an option match. The value is either a character or a numeric denoted by a leading #. If omitted the index shall be taken as the next within the sequence either following the previous index or using the opening index of zero(0).
operator	Defines whether the value is optional (=) or required (:). If omitted, it is assumed no value is required.
type	Optional value type against which the value shall be validated, stated as either a single character or its equivalent full name.
a[nything]	Anything.
s[tring]	String.
c[haracter]	Character value.
i[nteger] [+]	Decimal integer value plus optional positive only modifier.
n[umeric] [+]	Numeric (oct, dec and hex).
f[loat]	Floating point including integer.
d[ouble]	Double precision floating point including integer.
b[olean]	Boolean, 0/1, y[e]s/n[o], on/off, true/false; result being a string containing 0 or 1.

Returns

Option value or index, starting at zero, otherwise one of the following negative error codes. If the case of error codes -2 or less, value shall contain an error condition describing the condition.

- 1 End of the options (EOF).
- 2 Unknown option.
- 3 Ambiguous option.
- 4 Argument required.
- 5 Unexpected argument.
- 10 Invalid value, for example "expected a numeric value".
- 99 Invalid option specification.

Portability

A **GriefEdit** extension.

Example

```

list longoptions = {
    "help,h",           // note duplicates result in first match
    "verbose,v",
    "verbose2,#2",
    "integer"          // extended format
};
int ch;

if ((ch = getopt(value, "hv", longoptions, get_parm(1))) > 0) {
    do {
        switch (ch) {
        case 'h':           // -h (short) or --help (long) option
            break;

        case 'v':           // -v (short) or --verbose (long)
            break;

        case 2:             // --verbose2
            break;

        case 3:             // --integer=<value>
            i = atoi(value);
            break;

        case '?':           // error or unknown option
        case ':':           // or argument expected
            if (length(value)) {
                error("myfunction: %s", value);
            }
            break;

        default:
            break;
        }
    } while ((ch = getopt(value)) > 0);
}

```

See Also[arg_list](#), [split_arguments](#)**getsubopt**

```

int getsubopt( string value,
               [list options],
               [string|list args],
               [string delim],
               [string quotes]   )

```

Parse suboption arguments from a string.

Description

The *getsubopt()* primitive shall parse suboption arguments in a flag argument. Such options often result from the use of [getopt](#).

Parameters

<i>options</i>	Option declaration list of one or more strings of the following form "tag[, index][[:=][type]]", see below for more details.
<i>value</i>	Tag value, otherwise set to an empty string.
<i>args</i>	Argument buffer to be processed. Note, on success the buffer shall be modified with the leading matched option and trailing value removed.
<i>delim</i>	Optional delimiter, if omitted defaults to a comma(,).

Return

Option index, otherwise one of the following negative error codes. If the case of error codes -2 or less, value shall contain an error condition describing the condition,

- 1 End of the options (EOF).
- 2 Unknown option.
- 4 Argument required.
- 5 Unexpected argument.

- 6 Invalid value, for example "expected a numeric value".
- 10 Invalid option specification.

Portability

A **GriefEdit** extension.

Example

```
list suboptions = {
    "help,h",
    "verbose,v",
    "verbose2,#2",
    "integer:i"
};

if ((ch = getsubopt(value, suboptions, get_parm(1))) >= 0) {
    do {
        switch (ch) {
        case 'h':
            break;
        case 'v':
            break;
        case 2:
            break;
        case 3:
            break;
        default:
            break;
        }
    } while ((ch = getsubopt(value)) >= 0);
}
if (ch < 0) {
    error(value);
}
```

global

global sym1, sym2, ...

Declare a global variable.

Description

The *global()* primitive promotes a local declaration and making symbol global to all macros.

Global variables maintain their value across macro invocation and occupy permanent storage (See: [Scope](#)), whereas local variables are destroyed upon the macro they are contained within is terminated.

A variable must have been specified in a previous *int*, *string*, *list*, *float* or *declare* statement before it can be made into a global.

The *global* is a managed primitive and shall be automatically invoked on *global* variable declarations as follows:

```
// global declarations

int global_int1 = 1234;
static int global_int2;
const static int global_int3 = 125;

string global_string2 = "Hello world";
static string global_string2;

float global_float1 = "Hello world";
static float global_float2;

list global_list1;
static list global_list2;

void
mymacro()
{}
```

Each macro object containing *global* declarations shall contain an internal *_init* macro, which is utilised by the Macro

Compiler to initialise file level variables.

Note:

Both `global` and `_init` are considered as internal primitives, reserved for exclusive use by the GRIEF Macro Compiler and may change without notice.

Management of variable scoping and argument binding shall be handled automatically by the compiler.

Returns

nothing

Portability

n/a

See Also

[Types](#), [extern](#), [static](#), [const](#), [register](#)

inq_symbol

```
int inq_symbol(string symbol)
```

Determine if the symbol exists.

Description

The `inq_symbol()` primitive determines whether the variable `Iname` exists at the current scope (See: [Scope](#)). One main use of this primitive is to determine if a specific local buffer symbol has been defined.

Parameters

`symbol` Name of the symbol.

Example

An example usage

```
// required to resolve symbol at compile time
extern string my_buffer_var;

string
_set_buffer_var(string val)
{
    if (inq_symbol("my_buffer_var")) {
        my_buffer_var = mode;           // buffer-scope
    } else {
        // otherwise we must create
        string my_buffer_var = mode;
        make_local_variable( my_buffer_var );
    }
}

string
_get_buffer_var()
{
    if (inq_symbol("my_buffer_var")) {
        return my_buffer_var;
    }
    return "";
}
```

Returns

The `inq_symbol()` primitive returns a positive value indicates that the symbol exists, otherwise 0 if not found within the current scope.

Portability

A [GriefEdit](#) extension.

See Also

[make_local_variable](#)

int

```
int sym1, sym2 ...;
```

Declare an integer symbol.

Description

The *int* statement declares an integral type that stores values in the range;

-2,147,483,648 to 2,147,483,647

You can declare and initialize a variable of the type *int* like this example:

```
'int' i = 1234;
```

Returns

nothing

Portability

n/a

See Also

[Types](#), [string](#), [list](#), [float](#), [double](#), [array](#)

is_array

```
int is_array(declare &symbol)
```

Determine whether an array type.

Description

The *is_array()* primitive determines the type of a polymorphic expression and tests whether the specified *symbol* has of a array type.

Parameters

symbol Symbol reference.

Returns

true if a array type, otherwise **false**.

Portability

n/a

See Also

[array](#), [typeof](#)

is_float

```
int is_float(declare &symbol)
```

Determine whether a float type.

Description

The *is_float()* primitive determines the type of a polymorphic expression and tests whether the specified *symbol* has of a floating-point type.

Parameters

symbol Symbol reference.

Returns

true if a float type, otherwise **false**.

Portability

A **GriefEdit** extension.

See Also[float](#), [typeof](#)**is_integer**

```
int is_integer(declare &symbol)
```

Determine whether an integer type.

Description

The `is_integer()` primitive determines the type of a polymorphic expression and tests whether the specified `symbol` has of an integer type.

Parameters

`symbol` Symbol reference.

Returns

true if an integer type, otherwise **false**.

Portability

n/a

See Also[int](#), [typeof](#)**is_list**

```
int is_list(declare &symbol)
```

Determine whether a list type.

Description

The `is_list()` primitive determines the type of a polymorphic expression and tests whether the specified `symbol` has of a list type.

Parameters

`symbol` Symbol reference.

Returns

true if a list type, otherwise **false**.

Portability

n/a

See Also[list](#), [typeof](#)**is_null**

```
int is_null(declare &symbol)
```

Determine whether a NULL type.

Description

The `is_null()` primitive determines the type of a polymorphic expression and tests whether the specified `symbol` has of a NULL type.

Parameters

`symbol` Symbol reference.

Returns

true if a NULL type, otherwise **false**.

Portability

n/a

See Also

`list, typeof`

is_string

```
int is_string(declare &symbol)
```

Determine whether a string type.

Description

The `is_string()` primitive determines the type of a polymorphic expression and tests whether the specified `symbol` has of a string type.

Parameters

`symbol` Symbol reference.

Returns

true if a string type, otherwise **false**.

Portability

n/a

See Also

`string, typeof`

```
int is_type(declare    &symbol,
            int|string  type   )
```

Determine whether an explicit type.

Description

The `is_type()` primitive determines the type of a polymorphic expression and tests whether the specified `symbol` is of the type `type`.

Parameters

`symbol` Symbol reference.

`type` Type identifier or name as follows.

Type	Name
F_INT	integer
F_STR	string
F_FLOAT	float, double
F_LIST	list
F_ARRAY	array
F_NULL	NULL
F_HALT	undef

Returns

true if the stated type, otherwise **false**.

Portability

n/a

See Also

`array, typeof`

```
void make_local_variable(declare &sym,
                        ... )
```

Make a buffer local variable.

Description

The `make_local_variable()` primitive associates the specified variable with the current buffer, becoming a buffer local variable..

Unlike scoped variables that are destroyed when the block within which they are defines terminates, buffer local variables maintain their value across macro invocation and occupy permanent storage until the buffer is deleted (See: <scope>).

Parameters

`sym` Symbol reference.
`...` Optional additional symbol references.

Returns

The `make_local_variable()` primitive returns nothing directly.

On error conditions the following diagnostics message shall be echoed on the command prompt, with `xxx` representing the symbol name.

missing symbol.

'`xxx`' not found at current scope.

cannot promote reference '`xxx`'.

system variable '`xxx`'.

Portability

n/a

See Also

[inq_symbol](#), <types>

put_parm

```
int put_parm( int      argidx,
              declare val,
              [int optional = TRUE])
```

Assign an argument value.

Description

The `put_parm()` primitive assigns a value `val` to a parameter positioned at `argidx` that was passed to the current macro.

Parameters

`argidx` Integer argument index of the parameter passed to the current macro which is to be assigned a value; parameter indexes start at zero.
`val` Value to be assigned to the parameter; the value type should match the type of the parameter.
`optional` Optional boolean value, if **true** missing parameters shall not generate an error, otherwise if **false** or omitted an error will be echoed.

Returns

The `put_parm()` primitive returns 1 or greater on sucesss, otherwise 0 or less on error.

On the detection of error conditions the following diagnostics messages shall be echoed on the command prompt where `x` is the associated argument number;

put_parm: argument index '`x`' out of range

```
put_parm: argument 'x' incorrect type
```

```
put_parm: argument 'x' type conversion error
```

Example

Assign the value *100* to the first parameter.

```
if (put_parm(0, 100)) {
    message("assigned");
}
```

Portability

n/a

See Also

[get_parm](#)

ref_parm

```
void ref_parm( int     argument,
               string local_symbol,
               [int optional = FALSE])
```

Create a reference parameter.

Description

The *ref_parm()* primitive creates a local reference to one of the current macro arguments, this primitive is the underlying implementation of macro reference arguments.

```
int mymacro(int &iparm, string &sparm)
```

is equivalent to the following

```
int mymacro()
{
    ref_parm(0, "iparm");
    ref_parm(1, "sparm");
```

Note:

This interface should be considered as an internal primitive, reserved for exclusive use by the GRIEF Macro Compiler and may change without notice. Management of variable scoping and argument binding shall be handled automatically by the compiler.

Parameters

argument	Argument index.
local_symbol	Name of the local symbol used as the local alias to the referenced argument.
optional	Optional integer flag determining whether the argument is optional, if omitted is assumed FALSE .

Returns

nothing

Portability

A **GriefEdit** extension

See Also

[get_parm](#), [put_parm](#)

register

```
register idx1, sym1, idx2, sym2 ...;
```

Define a variable as being a register.

Description

The *register* qualifier explicitly declares a locally scoped data object as something that should be cached against the given index.

Registers are function scoped, being unique to each macro and are visible only within the function.

Returns

nothing

Portability

An experimental GRIEF extension; functionality may change.

See Also

[Types](#), [global](#), [extern](#), [static](#), [const](#)

static

```
static var1, var2, ...;
```

Define a function or module scope.

Description

The *static* statement can be used to change the storage scope of a variable or function. It is one of the major mechanisms to enforce information hiding. *static* denotes that a function or data element is only known within the scope of the current module. This provides a form of object-hiding and can avoid name clashes with other macros (See: [Scope](#)).

In addition, if you use the *static* statement with a variable that is local to a function, it allows the last value of the variable to be preserved between successive calls to that function, including during recursions.

Variables

A *static* variable is initialized only once. Globals are performed within the body of the *_init* function, whereas a function *static* variable that has an initializer is initialized the first time it comes into existence.

Function::*

A *static* function is hidden from usage outside their own macro file (or module), this can present a problem with functionality which involves the usage of callbacks (e.g. *assign_to_key*). In this case, the :: (scope resolution) operator is used to qualify hidden names so that they can still be used.

Example

```
void
main()
{
    assign_to_key("<Alt-A>", "my_alt_a");
}

static void
my_alt_a()
{
    //function body
}
```

The *static* declaration of *my_alt_a()* referenced by the *assign_to_key()* within *main()* won't be visible when the "Alt-A" key is processed as it shall be out of scope. The usage of "::my_alt_a" forces the *my_alt_a* reference to become fully qualified at the time of the key assignment. The following examples have the equivalent result:

Implicit current module, where if a null module name is specified (e.g. "::function") then the symbol shall be bound to current module.

```
assign_to_key("<Alt-A>", "::my_alt_a");
```

or, explicit current module, where a named namespaces is specified (e.g. "module::function"):

```
module("my_module");
assign_to_key("<Alt-A>", "my_module::my_alt_a");
```

or

```
module("my_module");
assign_to_key("<Alt-A>", inq_module() + "::my_alt_a");
```

Returns

nothing

Portability

Module *static* declarations are a GRIEF extension.

See Also

[Types](#), [global](#), [extern](#), [const](#), [register](#)

string

```
string sym1, sym2 ...;
```

Declare a string symbol.

Description

The *string* statement declares a containers which may contain zero or more characters.

Returns

nothing

Portability

n/a

See Also

[Types](#), [int](#), [list](#), [float](#), [double](#), [array](#)

typeof

```
string typeof(declare &symbol)
```

Determine the symbol type.

Description

The *typeof()* primitive determines the type of a polymorphic expression returning a string describing the underlying type.

Returns

The *typeof()* primitive returns one of following strings dependent on the type of the specified symbol.

- "integer" - An integer type.
- "string" - A string type.
- "float" - A floating-point type.
- "list" - A List.
- "array" - An array (reserved for future use).
- "NULL" - NULL.
- "undef" - Undefined or omitted.
- "unknown-type" - Unknown type.

Portability

n/a

See Also

[is_integer](#), [is_string](#), [is_float](#), [is_list](#), [is_null](#)

\$Id: \$

To send feedback on this topic **email:** griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Window Primitives

Summary

Window Primitives

Macros

change_window	Selects a new window.
change_window_pos	Modify window coordinates/size.
close_window	Close specified the window.
create_edge	Create an edge, splitting the window.
create_menu_window	Create the menu window.
create_tiled_window	Creates a tiled window.
create_window	Create a popup window.
delete_edge	Delete an edge, combining a split window.
delete_window	Delete a window.
distance_to_indent	Calculate distance to next indent.
distance_to_tab	Calculate distance to next tab.
inq_char_map	Retrieve the character-map.
inq_ctrl_state	Retrieve the state of a window control.
inq_mode	Returns the overstrike mode.
inq_top_left	Retrieve window view port coordinates.
inq_views	Determine window count.
inq_window	Retrieve the current window.
inq_window_buf	Retrieve the associated buffer.
inq_window_flags	Retrieve window flags.
inq_window_info	Retrieve the current window information.
inq_window_infox	Retrieve information about a window.
insert	Insert string into current buffer.
insert_buffer	Insert format text into a buffer.
insert_mode	Set the buffer insert/overstrike mode.
insertf	Insert a formatted string.
move_edge	Modify a window.
next_window	Obtain the next window identifier.
right	Move position right one character.
self_insert	Insert a character as if it was typed.
set_buffer_cmap	Set a buffers character-map.
set_ctrl_state	Set the state of a window control.
set_feature	Config an editor feature.
set_font	Set the current window fonts.
set_top_left	Manages window view port coordinates.
set_window	Set the active window.
set_window_cmap	Set a windows character-map.
set_window_flags	Set window flags.
set_wm_name	Set the window and/or icon name.
translate_pos	Convert window coordinates.
up	Move position up one line.

Macros

change_window

```
void change_window([int direction],  
                  [string message])
```

Selects a new window.

Description

The *change_window()* primitive selects an adjoining window as the current located in the specified *direction* identified as follows,

- 0 Above/up.
- 1 Right.
- 2 Below/down.
- 3 Left.

If *direction* is omitted the user is prompted; the user indicates the change direction by use of the arrow keys.

Change direction [<^v>]

If the selected edge has no other window associated, then the user is informed as follows:

No window available

Parameters

- `direction` Optional integer direction stating the edge on which the change operation occurs resizing the associated window (as above), if omitted the user is prompted.
- `message` Optional message string to be used as the prompt, if omitted the default of "Change direction" is used.

Returns

The `change_window()` primitive returns 1 on success, 0 if the edge does exist, otherwise -1 if the user aborted.

Portability

n/a

See Also

[set_window](#), [next_window](#)

change_window_pos

```
int change_window_pos( [int topx],
                      [int topy],
                      [int width],
                      [int height],
                      [int winnum] )
```

Modify window coordinates/size.

Description

The `change_window_pos()` primitive modifies the coordinates and/or the size of the specified window, if omitted the current window.

Note, care must be taken not position a window outside the physical window otherwise unexpected results including application crashes may result.

Parameters

- `topx` Optional integer, if stated sets the x coordinate of the top left corner of the window, otherwise the current coordinate is taken.
- `topy` Optional integer, if stated sets the y coordinate of the top left corner of the window, otherwise the current coordinate is taken.
- `width` Optional integer, if stated sets the new advised width in columns; the resulting width shall not be permitted to be smaller than the required width to display the current window title and/or message content.
- `height` Optional integer, if stated sets the new advised height in lines.
- `winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

Returns non-zero if the windows coordinates were modified, otherwise zero if an error occurred.

To determine the resulting window coordinates and size the [inq_window_info](#) primitive should be used, since boundary logic may have resized the window in order to obey the limits of the physical display.

Portability

The `winnum` is a **GriefEdit** extension.

See Also

[inq_window_info](#), [create_window](#)

close_window

```
void close_window([int winnum])
```

Close specified the window.

Description

The `close_window()` primitive is reserved for future BRIEF compatibility.

The `close_window()` primitive closes the specified window *winum*. The adjoining windows are enlarged to take up the space along the *best fit* border.

The *best fit* border is the one that has a set of windows exactly matching the border of the window being closed. If there is more than one, the left is eliminated first, then right, bottom and top. The top left window becomes current.

Parameters

`winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

`none`

Portability

Not implemented.

See Also

[delete_window](#)

create_edge

```
int create_edge([int direction])
```

Create an edge, splitting the window.

Description

The `create_edge()` primitive creates a new edge, splitting the current window in half resulting in a new window.

The window local in the specified *direction* is split identified as follows,

- 0 Above/up.
- 1 Right.
- 2 Below/down.
- 3 Left.

If *direction* is omitted the user is prompted for the split direction; the user indicates the split direction by use of the arrow keys.

Select new side [<^v>]

The selected window edge should have suitable screen space to permit the split operation otherwise the request is ignored and the user is informed as follows:

Window would be too small.

Parameters

`direction` Optional integer direction stating the edge on which the split operation occurs creating the new window (as above), if omitted the user is prompted.

Returns

The `create_edge()` primitive returns 1 on success, 0 if the window was too small to split, otherwise -1 if the user aborted.

Portability

n/a

See Also

[delete_edge](#), [move_edge](#)

create_menu_window

```
int create_menu_window([int create])
```

Create the menu window.

Description

The *create_menu_window()* primitive retrieves and optionally creates the menu if not already created. The menu resource is a singleton being the top line of display.

Parameters

create Optional integer flag, if specified as non-zero and the menu resource has as yet to be created, it shall be built.

Returns

Returns the unique identifier of the menu window resource, otherwise -1 if the menu has yet to be created.

Portability

A **GriefEdit** extension.

See Also

[create_window](#), [create_tiled_window](#)

create_tiled_window

```
int create_tiled_window(int lx,
                      int by,
                      int rx,
                      int ty,
                      [int bufnum])
```

Creates a tiled window.

Description

The *create_tiled_window()* primitive creates a tiled window. The stated coordinates (lx, by, rx and ty) represent the total window arena irrespective of the **borders** status, with (0, 0) being the top left hand corner of the screen.

Care should be taken not to overlay tiled windows and that all of the visible display has been assigned to window.

Generally tiled windows are created by the end user splitting the editor windows. This primitive is primarily utilised during state restoration to recover the state of the previous edit session.

The created window shall not be visible until the display has been enabled using [display_windows](#).

Parameters

lx Coordinate of the left edge.
by Coordinate of the bottom edge.
rx Coordinate of the right edge.
ty Coordinate of the top edge.
bufnum Optional buffer identifier to be attached to the newly created window, see [attach_buffer](#).

Returns

Returns the unique identifier of the new window.

Portability

n/a

See Also

[create_window](#), [display_windows](#), [inq_screen_size](#)

create_window

```
int create_window( int lx,
                  int by,
                  int rx,
                  int ty,
                  [string message])
```

Create a popup window.

Description

The *create_window()* primitive creates a popup window resource which shall become the current window. The

window should be suitably sized based on the current screen size, which can be determined using [inq_screen_size](#).

Popup windows stack upon any underlying tiled or other popup which are located in the same position, the order may be controlled using [set_window_priority](#),

On completion at buffer needs to be associated with the newly created window using [attach_buffer](#).

Parameters

- `lx` Coordinate of the left edge.
- `by` Coordinate of the bottom edge.
- `rx` Coordinate of the right edge.
- `ty` Coordinate of the top edge.
- `message` Optional string containing the message which shall be displayed on the bottom frame.

Returns

Returns the unique identifier of the new window.

Portability

Unlike BRIEF the number of windows which may be active at any one time is only limit by system resources.

See Also

[attach_buffer](#), [create_edge](#), [create_tiled_window](#), [create_menu_window](#), [delete_window](#), [inq_screen_size](#)

delete_edge

```
int delete_edge([int direction])
```

Delete an edge, combining a split window.

Description

The `delete_edge()` primitive deletes an edge, combined two windows sharing an adjoining edge into a single window.

The windows located in the specified *direction* are joined as follows,

- 0 Above/up.
- 1 Right.
- 2 Below/down.
- 3 Left.

If *direction* is omitted the user is prompted for the split direction; the user indicates the split direction by use of the arrow keys.

Select edge to delete [<^v>]

Once a window is deleted with `delete_edge`, its window identifier become invalid.

Parameters

- `direction` Optional integer direction stating the edge on which the join operation occurs deleting the associated windows (as above), if omitted the user is prompted.

Returns

The `delete_edge()` primitive returns 1 on success, 0 if the edge does exist, otherwise -1 if the user aborted.

Portability

n/a

See Also

[create_edge](#), [move_edge](#)

delete_window

```
void delete_window([int winum])
```

Delete a window.

Description

The `delete_window()` primitive deletes the specified window.

Parameters

`winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

`none`

Portability

Unlike BRIEF any window may be deleted.

See Also

[create_window](#), [delete_edge](#)

distance_to_indent

```
int distance_to_indent([int column])
```

Calculate distance to next indent.

Description

The `distance_to_indent()` primitive calculates the distance in characters to the next active indentation from either the specified `column` otherwise if omitted the current cursor position.

The active indentation information is sourced from the first available specification in order from the following:

- Ruler specification (See: [set_ruler](#)).
- Buffer indentation (See: [set_indent](#)).
- Tab specification (See: [tabs](#)).

Parameters

`column` Optional column if omitted the current buffer position is referenced.

Returns

Returns the number of columns/characters between the referenced column and the next indentation. If the referenced column is on a tab stop, then the number of characters to the next tab stop shall be returned.

Portability

A **GriefEdit** extension.

See Also

[distance_to_tab](#), [set_ruler](#), [set_indent](#), [tabs](#)

distance_to_tab

```
int distance_to_tab([int column])
```

Calculate distance to next tab.

Description

The `distance_to_tab()` primitive calculates the distance in characters to the next tab from either the specified `column` otherwise if omitted the current cursor position.

Parameters

`column` Optional column if omitted the current buffer position is referenced.

Returns

Returns the number of columns/characters between the referenced column and the next tab stop. If the referenced column is on a tab stop, then the number of characters to the next tab stop shall be returned.

Portability

n/a

See Also

[tabs](#), [distance_to_indent](#)

inq_char_map

```
int inq_char_map([int winnum],
                 [string &name])
```

Retrieve the character-map.

Description

The *inq_char_map()* primitive retrieves the current character-map identifier of the underlying buffer, otherwise if none is associated with the specified window. If the window identifier is omitted, the current window shall be queried.

Parameters

`winnum` Optional window identifier, if omitted the current window shall be referenced.
`name` Optional string referenced, if specified shall be populated with the assigned character-map name.

Returns

The *inq_char_map()* primitive returns the associated character-mapid otherwise -1 if one is not assigned.

Portability

The *name* parameter is a GRIEF extension.

See Also

[create_char_map](#), [set_window_cmap](#), [set_buffer_cmap](#)

inq_ctrl_state

```
int inq_ctrl_state(int ctrl,
                   [int winnum])
```

Retrieve the state of a window control.

Description

The *inq_ctrl_state()* primitive retrieves the state of a window control of the specific window, if omitted the current window.

Parameters

`ctrl` Control identifier; see [set_ctrl_state](#) for details.
`winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

The return value depends on the specified control.

- CLOSE_BTN, ZOOM_BTN, VERT_SCROLL, and HORZ_SCROLL
 - 0 Disabled.
 - 1 Enabled.
 - 1 Hidden globally.
 - 2 Hidden explicitly for this window.
 - 3 Hidden globally and explicitly.
- VERT_THUMB and HORZ_THUMB,
 - 1 Disabled.
 - n The position (percentage) of the thumb on the scroll bar, with a value 0 thru to 100.

Portability

n/a

See Also

[set_ctrl_state](#)

inq_mode

```
int inq_mode([int bufnum],
            [int &localised])
```

Returns the overstrike mode.

Description

The *inq_mode()* primitive retrieves the current insert/overstrike (also known as overtype) mode.

Parameters

- bufnum Optional buffer number, if omitted the current buffer shall be referenced.
 localised Optional integer reference, if specified is populated with the localisation status. If **true** then the mode is specific to the referenced buffer, otherwise the global mode is active.

Returns

The *inq_mode()* primitive retrieves the non-zero if in insert mode, otherwise zero if in overstrike mode.

Portability

The localised status is a **GriefEdit** extension.

See Also

[insert_mode](#), [insert](#)

inq_top_left

```
int inq_top_left( [int &top],
                  [int &indent],
                  [int winnum],
                  [int &line],
                  [int &col],
                  [int &bufnum] )
```

Retrieve window view port coordinates.

Description

The *inq_top_left()* primitive retrieves the view port coordinates of the specified window into their associated buffer, if omitted the current window is referenced.

The variables *line* and *column* retrieves the buffer coordinates displayed at the top left corner of the window. *csrline* and *csrcolumn* retrieve the buffers cursor position.

Parameters

- line Optional integer, if specified is populated with the line at the top of the window.
 column Optional integer, retrieves the column at the top left corner of the window.
 winnum Optional window identifier, if omitted the current window shall be referenced.
 csrline Optional integer, if specified is populated with the buffer cursor line position.
 csrcolumn Optional integer, retrieves the buffer cursor column position.
 bufnum Optional integer, if specified is populated with the associated buffer identifier.

Returns

The associated window identifier, otherwise -1 on error.

Portability

n/a

See Also

[set_top_left](#), [inq_position](#)

inq_views

```
int inq_views([int bufnum])
```

Determine window count.

Description

The *inq_views()* primitive determines the number of windows that are viewing the specified buffer *bufnum*.

Parameters

- bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The *inq_views()* primitive returns the number of windows attached to the specified buffer, otherwise 0 on error.

Portability

n/a

See Also

[set_window](#)**inq_window**

```
int inq_window()
```

Retrieve the current window.

Description

The *inq_window()* primitive retrieves the window identifier of the current window.

Parameters

none

Returns

The *inq_window()* primitive returns the current window identifier otherwise -1 if there is no window. This identifier can be used to save and restore the active window.

Portability

n/a

See Also

[change_window](#), [create_window](#)

inq_window_buf

```
int inq_window_buf([int winnum])
```

Retrieve the associated buffer.

Description

The *inq_window_buf()* primitive retrieves the associated buffer identifier of the specified window, if omitted the current window.

Parameters

winnum Optional window identifier, if omitted the current window shall be referenced.

Returns

The associated buffer identifier, otherwise -1 on error.

Portability

n/a

See Also

[attach_buffer](#), [inq_window](#)

inq_window_flags

```
int inq_window_flags([int winnum],
                     [string flags])
```

Retrieve window flags.

Description

The *inq_window_flags()* primitive retrieves the window flags of the specified window *winnum* otherwise if omitted the current window.

Window Flags

Available window flags.

Constant	Name	Description
WF_HIDDEN	hidden	Hide the window from view, used to hide nested popup's/boss mode etc.
WF_NO_SHADOW	no_shadow	Turn off the popup window shadows.
WF_NO_BORDER	no_border	Turn off borders, regardless of the borders() setting.
WF_SYSTEM	system	Window is a system window (e.g. menu).
WF_SHOWANCHOR	showanchor	Show anchor regardless of selection status.

Constant	Name	Description
WF_SELECTED	selected	Highlight the title regardless of selection status.
WF_LAZYUPDATE	lazyupdate	Delay any updates until next refresh().
WF_LINE_NUMBERS	line_numbers	Line numbers.
WF_LINE_STATUS	line_status	Line status.
WF_EOF_DISPLAY	eof_display	Show <EOF> marker.
WF_TILDE_DISPLAY	tilde_display	Show ~ marker as EOF marker.
WF_HIMODIFIED	himodified	Highlight modified lines.
WF_HIADDITIONAL	hiadditional	Highlight additional lines.
WF_HICHANGES	hichanges	Highlight in-line changes.
WF_EOL_HILITE	eol_hilite	Limit highlight to EOL.
WF_EOL_CURSOR	eol_cursor	Limit cursor to EOL.

Parameters

- winnum Optional window identifier, if omitted the current window shall be referenced.
 flags Optional comma separated list of window flag names, if given the value of the specific flags are returned, otherwise the full flags is returned.

Returns

Returns the associated window flags.

Portability

The feature set exposed differs from CRISP™. It is therefore advised that the symbolic constants are using within #ifdef constructs.

The *flag* argument is a **GriefEdit** extension.

See Also

[set_window_flags](#)

inq_window_info

```
int inq_window_info( [int &winnum],
                     [int &bufnum],
                     [int &lx],
                     [int &by],
                     [int &rx],
                     [int &ty],
                     [string &title = NULL],
                     [string &message = NULL])
```

Retrieve the current window information.

Description

The *inq_window_info()* primitive retrieves information associated with the current window.

Parameters

- winnum An integer variable which shall be populated with the window identifier, otherwise -1 if no buffer is attached.
 bufnum An integer variable which shall be populated with the buffer identifier of the buffer attached to the specified window, otherwise -1 if no buffer is attached.
 lx An integer variable which shall be populated with the left x coordinate of the specified window.
 by An integer variable which shall be populated with the bottom x coordinate of the specified window.
 rx An integer variable which shall be populated with the right x coordinate of the specified window.
 ty An integer variable which shall be populated with the top y coordinate of the specified window.
 title A string variable, which shall be assigned the specified window title value.
 message A string variable, which shall be assigned the specified window message value.

Returns

The *inq_window_info()* primitive returns zero if the specified window is tiled, one if the window is an popup/overlapping window and two if the menu window.

Portability

This primitive differs from the BRIEF implementation, in that it only returns information associated with the current window and update the first argument to reflect the windows identifier, also see [inq_window_infox](#).

The *title* and *message* parameters are extensions.

See Also

[inq_window](#), [create_window](#), [create_tiled_window](#), [create_menu_window](#)

inq_window_info

```
int inq_window_info( [int winnum],
                     [int &bufnum],
                     [int &lx],
                     [int &by],
                     [int &rx],
                     [int &ty],
                     [string &title = NULL],
                     [string &message = NULL])
```

Retrieve information about a window.

Description

The *inq_window_info()* primitive retrieves information associated with the specified window *winnum* or the current window if no window is specified.

Parameters

<i>winnum</i>	Optional integer window identifier, if omitted the current window is referenced.
<i>bufnum</i>	An integer variable which shall be populated with the buffer identifier of the buffer attached to the specified window, otherwise -1 if no buffer is attached.
<i>lx</i>	An integer variable which shall be populated with the left x coordinate of the specified window.
<i>by</i>	An integer variable which shall be populated with the bottom x coordinate of the specified window.
<i>rx</i>	An integer variable which shall be populated with the right x coordinate of the specified window.
<i>ty</i>	An integer variable which shall be populated with the top y coordinate of the specified window.
<i>title</i>	A string variable, which shall be assigned the specified window title value.
<i>message</i>	A string variable, which shall be assigned the specified window message value.

Returns

The *inq_window_info()* primitive returns zero if the specified window is tiled, one if the window is an popup/overlapping window and two if the menu window.

Portability

This primitive mirrors the original BRIEF interface presented by *inq_window_info*, permitted either the current or an explicit window to be referenced.

The *title* and *message* parameters are extensions.

See Also

[inq_window](#), [create_window](#), [create_tiled_window](#), [create_menu_window](#)

insert

```
int insert(string|int val,
          [int num = 1])
```

Insert string into current buffer.

Description

The *insert_process()* primitive inserts the specified string or integer character value *val* into the current buffer. The value shall be inserted *num* times, which if omitted defaults to 1.

Parameters

<i>val</i>	String or integer character value to be inserted.
<i>num</i>	Option integer number stating the repeat count, if specified then the string is inserted the given number of times. If omitted, it defaults to 1.

Returns

The *insert()* primitive returns the number of characters inserted.

Portability

The standard function has a void declaration and returns nothing.

See Also

[insertf](#), [insert_buffer](#), [insert_process](#)

insert_buffer

```
int insert_buffer(int bufnum,
                 string format,
                 ... )
```

Insert format text into a buffer.

Description

The *insert_buffer()* primitive behaves like the C `printf()` function. It inserts the string *format* into the specified buffer and applies the `printf` style formatting commands as specified in *format*, and using the argument list.

When more than one argument is specified then *format* is treated as a `printf`-style format specification. (ie. % sequences are interpreted, otherwise % characters are inserted literally).

Parameters

bufnum Buffer number.
format String that contains the text to be written. It can optionally contain an embedded <Format Specification> that are replaced by the values specified in subsequent additional arguments and formatted as requested.
... Optional format specific arguments, depending on the format string, the primitive may expect a sequence of additional arguments, each containing a value to be used to replace a format specifier in the format string.
There should be at least as many of these arguments as the number of values specified in the format specifiers. Additional arguments are ignored by the primitive.

Returns

The *insert_buffer()* primitive returns the number of characters written to the referenced buffer, otherwise -1 on error.

Portability

The CRISPEdit™ version has a void declaration and returns nothing.

See Also

[insert](#), [insertf](#)

insert_mode

```
int insert_mode([int value],
               [int bufnum])
```

Set the buffer insert/overstrike mode.

Description

The *insert_mode()* primitive sets the insert/over-strike mode to *value* otherwise toggles if omitted.

The applied mode shall either be localised to the specified buffer *bufname*, otherwise if omitted the global (default) mode that applies to all buffers within a localised setting.

Parameters

value Optional integer stating the insert mode being zero for over-strike and non-zero for insert. For localised modes -1 clears the active localisation, restoring use of the global (default) mode. If omitted the current mode is toggled.
bufnum Optional buffer number when stated the buffer specific insert mode shall be modified, if omitted the global insert mode is modified.

Returns

The *insert_mode()* primitive returns the previous insert mode.

Portability

n/a

See Also

[inq_mode](#)

insertf

```
int insertf(string format,
           ... )
```

Insert a formatted string.

Description

The *insertf()* primitive behaves like the C `printf()` function. It inserts the string *format* into the current buffer and applies the `printf` style formatting commands as specified in *fmt*, and using the argument list.

When more than one argument is specified then *expr* is treated as a `printf`-style format specification. (ie. % sequences are interpreted, otherwise % characters are inserted literally).

Parameters

format String that contains the text to be written. It can optionally contain an embedded <Format Specification> that are replaced by the values specified in subsequent additional arguments and formatted as requested.
... Optional format specific arguments, depending on the format string, the primitive may expect a sequence of additional arguments, each containing a value to be used to replace a format specifier in the format string.
 There should be at least as many of these arguments as the number of values specified in the format specifiers. Additional arguments are ignored by the primitive.

Returns

The *insertf()* primitive returns the number of characters written to the referenced buffer, otherwise -1 on error.

Portability

A **GriefEdit** extension.

See Also

[insert](#), [insert_buffer](#)

move_edge

```
int move_edge([int direction],  
             [int amount]      )
```

Modify a window.

Description

The *move_edge()* primitive modifies the edges of tiled window, whereas for popup's allow the user to increase or decrease the size, and also move the window around the screen.

Example

Moves the lower edge of the current window up four lines.

```
move_edge(2, -4);
```

Parameters

direction Optional integer direction stating the edge on which the move operation occurs resizing the associated windows (as above), if omitted the user is prompted.
amount Optional integer expression stating the number or characters or lines to move the window. The number is relative to the top left of the screen, so positive numbers move away from the origin (0, 0) and negative numbers towards the origin. If not specified, the user will be prompted to move the edge with the arrow keys.

Returns

The *move_edge()* primitive returns non-zero if cursor moved, otherwise 0 if cursor did not move.

Portability

n/a

See Also

[create_edge](#), [delete_edge](#)

next_window

```
int next_window(int winnum)
```

Obtain the next window identifier.

Description

The `next_window()` primitive retrieves the window identifier of the next window from the internal window list relative to the specified window, if omitted the current window shall be referenced.

Parameters

`winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

The `next_window()` primitive returns the window identifier of the next tiled window from the window list.

If there is only a single window, then the same identifier as the specified shall be returned.

Example

Iterate though all windows

```
int curwin, win;

curwin = inq_window();
if ((win = curwin) != -1) {
    do {
        // ... process
    } while ((win = next_window(win)) != curwin);
}
```

Portability

n/a

See Also

`inq_window`, `set_window`

right

```
int right([int columns = 1],
          [int wrap = TRUE] )
```

Move position right one character.

Description

The `right()` primitive moves the cursor right one column retaining the current line.

Parameters

`columns` Optional number of columns to move the cursor; negative in which case the cursor movement is reversed behaving like `left`.

`wrap` Optional boolean value controlling whether the cursor wraps when positioned at the beginning of line. If **FALSE** line wrapping shall be disabled, see `left`.

Returns

The `right()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

Unlike BRIEF, if the cursor is moved past the beginning of the current line, then the cursor wraps around to the end of the previous line.

`wrap` is a **GriefEdit** extension.

See Also

`left`, `up`, `down`

self_insert

```
void self_insert([int character])
```

Insert a character as if it was typed.

Description

The `self_insert()` primitive insert a character into the current buffer. If `character` is specified, then the character

whose ASCII value is *character* is inserted into the current buffer instead of the last character typed.

The majority of characters are directly inserted yet the following infer special processing.

tab Cursor is moved the next tab stop, and space backfilled if at the end of the line.

newlines Cursor is repositioned on the next line.

This primitive is normally used in conjunction with [assign_to_key](#) to unassign an ASCII key by making it a normal, typeable character.

Parameters

character Optional integer character code to be inserted. If omitted, the value of the last key typed is inserted into the buffer.

Returns

nothing

Portability

n/a

See Also

[insert](#), [insertf](#), [assign_to_key](#), [keyboard_typeables](#)

set_buffer_cmap

```
int set_buffer_cmap([int mapid|string name],
                    [int bufnum])
```

Set a buffers character-map.

Description

The *set_buffer_cmap()* primitive attachs the specified character-map to a given buffer. A single character-map can be attached to any number of buffers.

Note that this association shall have precedence over the windows view of a buffer [set_window_cmap](#).

Parameters

mapid, name Character-map reference, being either an integer map identifier or the associated map name as a string. If omitted the default character-map shall be attached.

bufnum Optional buffer number, if omitted the current buffer shall be referenced.

Returns

The *set_buffer_cmap()* primitive returns the identifier of the resolved character-map otherwise -1 if the specified character map does not exist.

Portability

n/a

See Also

[create_char_map](#), [set_window_cmap](#), [inq_char_map](#)

set_ctrl_state

```
void set_ctrl_state(int ctrl,
                    int state,
                    [int winnum])
```

Set the state of a window control.

Description

The *set_ctrl_state()* primitive sets the state of a window control of the specific window, if omitted the current window.

Parameters

ctrl Control identifier.

WCTRLO_CLOSE_BTN Close button.

WCTRLO_ZOOM_BTN Zoom button.

WCTRLO_VERT_SCROLL Vertical scroll.

WCTRLO_HORZ_SCROLL Horizontal scroll.

WCTRLO_VERT_THUMB Vertical thumb.

WCTRLO_HORZ_THUMB Horizontal thumb.

state An integer specifying the desired state.

WCTRLS_ENABLE Enable the control all windows.

WCTRLS_DISABLE Disable the control all windows.

WCTRLS_HIDE Used to temporarily hide object for either the specified window or all windows, if window is omitted.

WCTRLS_SHOW Restore the show status of a hidden control. HIDE/SHOW calls nests, hence for a hidden object to be displayed the number of HIDE operations must be matched by the same number of SHOW operations.

WCTRLS_ZOOMED Display the zoomed button.

winnum Optional window identifier, if omitted the current window shall be referenced.

Returns

nothing

Portability

n/a

See Also

[inq_ctrl_state](#)

set_feature

```
int set_feature([int|string feature],
               [string value] )
```

Config an editor feature.

Description

The `set_feature()` primitive sets the status of the specific feature *feature*.

Warning:

The `set_feature()` primitive is an experimental interface and may change without notice.

Parameters

`feature` Name of the feature.

`value` Configuration value.

Return

The `set_feature()` primitive returns non-zero on success, otherwise zero on error.

Macro Portability; A **GriefEdit** extension.

See Also

[inq_feature](#)

set_font

```
int set_font([string normalfont],
            [string italicfont] )
```

Set the current window fonts.

Description

The `set_font()` primitive configures the active normal and/or italic font of the current running **GriefEdit** image.

Note:

Only available when running under a suitable windowing system, otherwise this primitive is a no-op.

Parameters

`normalfont` Optional string containing the normal text font.

`italicfont` Optional italic font.

Returns

The `set_font()` primitive returns zero or greater on success, otherwise -1 on error.

Portability

GriefEdit extended.

See Also

`inq_font`, `set_wm_name`

set_top_left

```
int set_top_left( [int line],
                  [int column],
                  [int winnum],
                  [int csrlne],
                  [int csrcolumn],
                  [int bufnum] )
```

Manages window view port coordinates.

Description

The `set_top_left()` primitive manages the view port coordinates, setting up the window and buffer positional relationship.

The window effected can be referenced by one of three means. Directly, `winnum` if specified states the window identifier used. Indirectly, if `winnum` is omitted the window referenced shall be the one attached to the specified buffer identifier `bufnum`. If neither are specified then the current window is assumed.

The arguments `line` and `column` set the buffer coordinates displayed at the top left corner of the window.

`csrlne` and `csrcolumn` set the buffers cursor position.

Parameters

<code>line</code>	Optional integer, specifies the line within the buffer which should be at the top of the window.
<code>column</code>	Optional integer, specifies the column within the buffer which should be at the top of the window
<code>winnum</code>	Optional window identifier, if omitted the current window shall be referenced.
<code>csrlne</code>	Optional integer states the cursor position, if stated specifies the line within the buffer on which the cursor shall be positioned.
<code>csrcolumn</code>	Optional integer states the cursor position, if stated specifies the column within the buffer on which the cursor shall be positioned.
<code>bufnum</code>	Optional buffer identifier can be used to define a window indirectly, if <code>winnum</code> is omitted the window referenced shall be the one attached to the specified buffer identifier.

Returns

The effected window identifier, otherwise -1 on error.

Portability

n/a

See Also

`inq_top_left`

set_window

```
int set_window(int winnum)
```

Set the active window.

Description

The `set_window()` primitive set the current window to the specified window identifier.

Parameters

`winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

Returns non-zero if the window was changed, otherwise zero if specified window was already the current and no change occurred.

Portability

Unlike BRIEF the current buffer is not affected, which changed the buffer to the one associated with the specified

window.

See Also

[inq_window](#), [next_window](#), [change_window](#)

set_window_cmap

```
int set_window_cmap([int mapid|string name],
                    [int winnum]           )
```

Set a windows character-map.

Description

The `set_window_cmap()` primitive attaches the specified character-map to a given window. A single character-map can be attached to any number of windows.

By default two system predefined character-map's are available known by the names "normal" and "binary", these are in addition to the number managed by the view's package.

Note that any buffer level association `set_buffer_cmap` shall have precedence over the windows view of a buffer.

Parameters

`mapid, name` Character-map reference, being either an integer map identifier or the associated map name as a string. If omitted the default character-map shall be attached.
`winnum` Optional window identifier, if omitted the current window shall be referenced.

Returns

The `set_window_cmap()` primitive returns the identifier of the resolved character-map otherwise -1 if the specified character map does not exist.

Portability

n/a

See Also

[create_char_map](#), [set_buffer_cmap](#), [inq_char_map](#)

set_window_flags

```
void set_window_flags( [int winnum],
                      [string set|int or_mask],
                      [string clear|int and_mask])
```

Set window flags.

Description

The `set_window_flags()` primitive modifies the window flags of the specified window `winnum` otherwise if omitted the current window.

If specified one or more flags shall be cleared using the `and_mask`, in addition one or more flags are set using the `or_mask`.

Note that `and_mask` (clear) is applied prior to the application of the `or_mask` (set).

Window Flags

Available window flags.

Constant	Name	Description
WF_HIDDEN	hidden	Hide the window from view, used to hide nested popup's/boss mode etc.
WF_NO_SHADOW	no_shadow	Turn off the popup window shadows'
WF_NO_BORDER	no_border	Turn off borders, regardless of the borders() setting.
WF_SYSTEM	system	Window is a system window (e.g. menu).
WF_SHOWANCHOR	showanchor	Show anchor regardless of selection status.
WF_SELECTED	selected	Highlight the title regardless of selection status.
WF_LAZYUPDATE	lazyupdate	Delay any updates until next refresh().
WF_LINE_NUMBERS	line_numbers	Line numbers.
WF_LINE_STATUS	line_status	Line status.
WF_EOF_DISPLAY	eof_display	Show <EOF> marker.
WF_TILDE_DISPLAY	tilde_display	Show ~ marker as EOF marker.
WF_HIMODIFIED	himodified	Highlight modified lines.

Constant	Name	Description
WF_HIADDITIONAL	hiadditional	Highlight additional lines.
WF_HICHANGES	hichanges	Highlight in-line changes.
WF_EOL_HILITE	eol_hilite	Limit highlight to EOL.
WF_EOL_CURSOR	eol_cursor	Limit cursor to EOL.

Parameters

- winnum Optional window identifier, if omitted the current window shall be referenced.
 set_mask Optional mask of flags to set. May either be an integer of AND'ed together flag constants, or alternatively a string of comma separated flag names.
 clear_mask Optional mask of flags to clear. May either be an integer of AND'ed together flag constants, or alternatively a string of comma separated flag names.

Returns

nothing

Portability

The feature set exposed differs from CRISP™. It is therefore advised that the symbolic constants are using within a #ifdef construct.

String flag forms are **GriefEdit** extensions.

See Also

[inq_window_flags](#)

set_wm_name

```
void set_wm_name([string wname],
                 [string iname] )
```

Set the window and/or icon name.

Description

The *set_wm_name()* primitive configures the window and/or the minimised icon name of the current running **GriefEdit** image.

Note:

Only available when running under a suitable windowing system, otherwise this primitive is a no-op.

Parameters

- wname Optional string containing the window name.
 iname Optional icon name.

Returns

The *set_wm_name()* primitive returns zero or greater on success, otherwise -1 on error.

Portability

GriefEdit extended.

See Also

[set_font](#)

translate_pos

```
int translate_pos( int x,
                   int y,
                   [int &winnum],
                   [int &line],
                   [int &col] )
```

Convert window coordinates.

Description

The *translate_pos()* primitive translates the physical screen position (x, y) into a window and logical line/col position.

Parameters

- `x, y` Screen coordinates to be translate.
- `winnum` Optional an integer variable which shall be populated with the window identifier within which the (x, y) position falls, otherwise -1 if no window was mapped.
- `line, col` Optional integer variables which shall be populated with the translated window coordinates, otherwise -1 if no window was mapped.

Returns

The `translate_pos()` primitive returns where the mouse cursor is located.

Value	Definition
<code>MOBJ_NOWHERE</code>	Not in any window.
<code>MOBJ_LEFT_EDGE</code>	Left bar of window.
<code>MOBJ_RIGHT_EDGE</code>	Right bar of window.
<code>MOBJ_TOP_EDGE</code>	Top line of window.
<code>MOBJ_BOTTOM_EDGE</code>	Bottom line of window.
<code>MOBJ_INSIDE</code>	Mouse inside window.
<code>MOBJ_TITLE</code>	On title.
<code>MOBJ_VSCROLL</code>	Vertical scroll area.
<code>MOBJ_VTHUMB</code>	Vertical scroll area.
<code>MOBJ_HSCROLL</code>	Horz scroll area.
<code>MOBJ_HTHUMB</code>	Horz scroll area.
<code>MOBJ_ZOOM</code>	Zoom button,
<code>MOBJ_CLOSE</code>	Close.
<code>MOBJ_SYSMENU</code>	System Menu.

Portability

n/a

See Also

[get_mouse_pos](#), [process_mouse](#)

up

```
int up([int lines = 1])
```

Move position up one line.

Description

The `up()` primitive moves the cursor up one line to the same column on the previous line.

Parameters

- `lines` Optional number of lines to move the cursor; may be negative in which case the cursor moves forward behaving like `down`.

Returns

The `up()` primitive returns non-zero on success denoting that the cursor moved, otherwise zero if the cursor remained unchanged.

Portability

n/a

See Also

[down](#), [left](#), [right](#)

\$Id: \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Third Party Packages

GRIEF development would not be possible without the work of others.

The following packages are normally only required for Windows builds as all are generally available on other targets, either as standard or optional installations.

Summary

Third Party Packages

GRIEF development would not be possible without the work of others.

Common Modules	A number of source modules have been sourced from current BSD distributions, which are under the <i>BSD-License</i> .
Crisp	CRISP-Custom Reduced Instruction Set Programmers Editor.
ND+ - Natural Docs Plus	ND+ is an open-source documentation generator for multiple programming languages.
ucpp	GRIEF utilises <i>ucpp</i> as its preprocessor, integrated into the Macro Compiler as an external stand-alone program as the initial stage of macro compilation.
makedepend	Modified version of makedepend 1.0.5
mandoc	The mandoc UNIX manpage compiler toolset.
extags	Exuberant Ctags.
flex	Version 2.5.10 released 2002-7-24
Oniguruma	Oniguruma is a regular expressions library.
TRE	TRE, The free and portable approximate regex matching library.
libregex	Henry Spencer's libregex
libteken	libteken: terminal emulator library
libmagic	file-5.29, sourced from OpenBSD.
libcharudet	Mozilla Universal Charset Detector.
libguess	High-speed character set detection.
hunspell	hunspell 1.3.2 and 1.7.0
libarchive	libarchive 3.3.3 and 3.6.1
iconv	NetBSD intl/iconv implementation from the Citrus Project
libz	Zlib Data compress library.
libbz2	bzip2 is a freely available, patent free, high-quality data compressor.
liblzma	XZ Utils is free general-purpose data compression software with a high compression ratio.

Common Modules

A number of source modules have been sourced from current BSD distributions, which are under the *BSD-License*.

The specific modules are located thru-out the code base.

Copyright (c) 1991, 1993
The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Crisp

CRISP-Custom Reduced Instruction Set Programmers Editor.

Available on a number of archive sites.

Source: <http://www.filewatcher.com> (crisp2.2e-src.tar.gz)

```
/pub/Linux/apps/editors
```

```
crisp2.2e-bin.tar.gz    clone of the DOS programmer's editor BRIEF [bin]
crisp2.2e-src.tar.gz    clone of the DOS programmer's editor BRIEF [src]
```

Crisp Copyright

The code in this file is part of the CRISP package which is (C) P Fox.

This code may be freely used in any product but the copyright remains that of the author.

This copyright notice is present to avoid a conflict of interest and to ensure that CRISP can continue to be a part of the public domain.

ND+ - Natural Docs Plus

ND+ is an open-source documentation generator for multiple programming languages.

The html version of the **GRIEF** documentation is generated using ND+.

Source: <http://sourceforge.net/projects/ndplus/>

ucpp

GRIEF utilises *ucpp* as its preprocessor, integrated into the Macro Compiler as an external stand-alone program as the initial stage of macro compilation.

A C preprocessor is a part of a C compiler responsible for macro replacement, conditional compilation and inclusion of header files. It is often found as a stand-alone program on Unix systems.

ucpp is such a preprocessor; it is designed to be quick and light, but anyway fully compliant to the ISO standard 9899:1999, also known as C99.

ucpp can be compiled as a stand-alone program, or linked to some other code; in the latter case, *ucpp* will output tokens, one at a time, on demand, as an integrated lexer.

Operational Modes

lexer mode *ucpp* is linked to some other code and outputs a stream of tokens (each call to the *lex()* function will yield one token)

on-lexer *ucpp* preprocesses text and outputs the resulting text to a file descriptor; if linked to some other code, the *cpp()* function must be called repeatedly, otherwise *ucpp* is a stand-alone binary. *ucpp* was written by Thomas Pornin.

Source: <http://code.google.com/p/ucpp>

(c) Thomas Pornin 2002

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
4. The name of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

makedepend

Modified version of makedepend 1.0.5

makedepend was developed as part of MIT's Project Athena. It was used extensively in building X11 and ancillary packages, but has since become superseded by the dependency generation facilities of various compilers, and is now used primarily as a worst-case fallback, e.g. by depcomp and GNU Automake.

The master development code repository can be found at.

- <git://anongit.freedesktop.org-/git-/xorg-/util-/makedepend>
- <http://cgit.freedesktop.org/xorg/util/makedepend>

Copyright (c) 1993, 1994, 1998 The Open Group

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE OPEN GROUP BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of The Open Group shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from The Open Group.

mandoc

The **mandoc** UNIX manpage compiler toolset.

Modified version of mandoc 1.13.4 (July 10, 2016) using sqlite3 (version 3.16.2). Minor modifications to allow win32 builds.

mandoc is a suite of tools compiling mdoc, the roff macro language of choice for BSD manual pages, and man, the predominant historical language for UNIX manuals. It is small, ISO C, ISC-licensed, and quite fast.

The main component of the toolset is the mandoc utility program, based on the libmandoc validating compiler, to format output for UNIX terminals (with support for wide-character locales), XHTML, HTML, PostScript, and PDF.

mandoc has predominantly been developed on OpenBSD and is both an OpenBSD and a BSD.lv project. We strive to support all interested free operating systems, in particular DragonFly, NetBSD, FreeBSD, Minix 3, and GNU/Linux, as well as all systems running the pkgsrc portable package build system.

Source: <http://mdocml.bsd.lv/>

mandoc Copyright

```
Copyright (c) 2011, 2012, 2013 - 2016 Ingo Schwarze <schwarze@openbsd.org>
Copyright (c) 2008, 2009, 2010, 2011 Kristaps Dzonsons <kristaps@bsd.lv>
```

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHORS DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

extags

Exuberant Ctags.

Author, Darren Hiebert <dhiebert at users.sourceforge.net>

Exuberant Ctags is a multilanguage reimplementation of the much-underused ctags(1) program and is intended to be the mother of all ctags programs. It generates indexes of source code definitions which are used by a number of editors and tools. The motivation which drove the development of Exuberant Ctags was the need for a ctags program which supported generation of tags for all possible C language constructs (which no other ctags offers), and because most were easily fooled by a number of preprocessor constructs.

No components of Exuberant Ctags are included within GRIEF, other than the *public domain* readtags.c module, which is utilised to read tags file generated by Exuberant.

Exuberant Ctags is separately licensed under the GNU GPL; see the source for details.

Source: <http://ctags.sourceforge.net>

flex

Version 2.5.10 released 2002-7-24

Flex carries the copyright used for BSD software, slightly modified because it originated at the Lawrence Berkeley (not Livermore!) Laboratory, which operates under a contract with the Department of Energy:

Copyright (c) 2001 by W. L. Estes <wlestes@uncg.edu>

Copyright (c) 1990, 1997 The Regents of the University of California.
All rights reserved.

This code is derived from software contributed to Berkeley by
Vern Paxson.

The United States Government has rights in this work pursuant
to contract no. DE-AC03-76SF00098 between the United States
Department of Energy and the University of California.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

Neither the name of the University nor the names of its contributors
may be used to endorse or promote products derived from this software
without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.

This basically says "do whatever you please with this software except
remove this notice or take advantage of the University's (or the flex
authors') name".

Note that the "flex.skl" scanner skeleton carries no copyright notice.
You are free to do whatever you please with scanners generated using flex;
for them, you are not even bound by the above copyright.

(code)

Topic: dmalloc

This is a version (aka dmalloc) of malloc/free/realloc written by
Doug Lea and released to the public domain.

Source: <http://g.oswego.edu/dl/html/malloc.html>

Dmalloc Copyright:

(code)

The person who associated a work with this deed has dedicated the work to
the public domain by waiving all of his or her rights to the work
worldwide under copyright law, including all related and neighboring
rights, to the extent allowed by law.

Oniguruma

Oniguruma is a regular expressions library.

Oniguruma by K. Kosako is a BSD licensed regular expression library that supports a variety of character encodings.

The Ruby programming language, since version 1.9, as well as PHP's multi-byte string module (since PHP5), use
Oniguruma as their regular expression engine. It is also used in products such as Tera Term, TextMate, Sublime Text
and SubEthaEdit.

Oniguruma is Japanese for "Devil's Chariot".

Source: <http://www.geocities.jp/kosako3/oniguruma>

TODO!

Migrate to *Oniguruma-mod*

Onigmo is a regular expressions library forked from Oniguruma. Some of new features introduced in Perl 5.10+ can be used.

Source: <https://github.com/k-takata/Onigmo>

Oniguruma Copyright

Copyright (c) 2002-2007 K.Kosako (sndgk393@ybb.ne.jp)
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

TRE

TRE, The free and portable approximate regex matching library.

TRE is a lightweight, robust, and efficient POSIX compliant regexp matching library with some exciting features such as approximate (fuzzy) matching.

The matching algorithm used in TRE uses linear worst-case time in the length of the text being searched, and quadratic worst-case time in the length of the used regular expression. In other words, the time complexity of the algorithm is $O(M^2N)$, where M is the length of the regular expression and N is the length of the text. The used space is also quadratic on the length of the regex, but does not depend on the searched string. This quadratic behaviour occurs only on pathological cases which are probably very rare in practice.

Source: <http://laurikari.net/tre/>

Features

TRE is not just yet another regexp matcher. TRE has some features which are not there in most free POSIX compatible implementations. Most of these features are not present in non-free implementations either, for that matter.

- Approximate matching.
- Strict standard conformance.
- Predictable matching speed.
- Portable.
- Free.

TRE Copyright

This is the license, copyright notice, and disclaimer for TRE, a regex matching package (library and tools) with support for approximate matching.

Copyright (c) 2001-2009 Ville Laurikari <vl@iki.fi>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

libregex

Henry Spencer's libregex

Source: <http://www.arglist.com/regex>

Libregex Copyright

Copyright 1992, 1993, 1994 Henry Spencer. All rights reserved.
This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it, subject to the following restrictions:

1. The author is not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
4. This notice may not be removed or altered.

libteken

libteken: terminal emulator library

It is currently used by FreeBSD's console driver. libteken is a terminal emulator, which implements a fair amount of escape sequences used by VT100, xterm and cons25.

Traditionally the FreeBSD console driver (syscons) uses the cons25 terminal type. This terminal type is basically a very compact subset of later VT-devices or graphical terminals like xterm. There are two problems with this approach.

- Many other operating systems do not ship cons25 termcap entries.
- Many embedded devices with serial or telnet interfaces only support VT100-style escape sequences.

Source: <http://80386.nl/projects/>

Libteken Copyright

Copyright (c) 2008-2009 Ed Schouten <ed@@FreeBSD.org>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

libmagic

file-5.29, sourced from OpenBSD.

This is Release 5.x of Ian Darwin's (copyright but distributable) file(1) command, an implementation of the Unix File(1) command.

It knows the *magic number* of several thousands of file types.

This version is the standard "file" command for Linux, *BSD, and other systems.

Libmagic Copyright

Copyright (c) Ian F. Darwin 1986, 1987, 1989, 1990, 1991, 1992, 1994, 1995.
Software written by Ian F. Darwin and others;
maintained 1994- Christos Zoulas.

This software is not subject to any export provision of the United States Department of Commerce, and may be exported to any country or planet.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice immediately at the beginning of the file, without modification, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

libcharudet

Mozilla Universal Charset Detector.

This library provides a highly accurate set of heuristics that attempt to determine the character set used to encode some input text. This is extremely useful when your program has to handle an input file which is supplied without any encoding metadata.

Recognised character set.

- UTF-8.
- UTF-16 (BE and LE).
- UTF-32 (BE and LE).
- windows-1252 (mostly equivalent to iso8859-1).
- windows-1251 and ISO-8859-5 (cyrillic).
- windows-1253 and ISO-8859-7 (greek).
- windows-1255 (logical hebrew. Includes ISO-8859-8-I and most of x-mac-hebrew).
- ISO-8859-8 (visual hebrew).
- Big-5.
- gb18030 (superset of gb2312).
- HZ-GB-2312.
- Shift-JIS.
- EUC-KR, EUC-JP, EUC-TW.
- ISO-2022-JP, ISO-2022-KR, ISO-2022-CN.
- KOI8-R.
- x-mac-cyrillic.
- IBM855 and IBM866.
- X-ISO-10646-UCS-4-3412 and X-ISO-10646-UCS-4-2413 (unusual BOM).
- ASCII.

The original code and documentation of the universalchardet library are available at,

- <http://www-archive.mozilla.org/~projects-/intl-/chardet.html>
- <http://lxr.mozilla.org/seamonkey/source/extension/universalchardet/>

This port exposes a C interface and dependency-free interface to the Mozilla C++ UCSD library.

Libcharudet Copyright

The library is subject to the Mozilla Public License Version 1.1 (the "License").

Alternatively, it may be used under the terms of either the GNU General Public License Version 2 or later (the "GPL"), or the GNU Lesser General Public License Version 2.1 or later (the "LGPL").

Usage within GRIEF is under the LGPL 2.1.

libguess

High-speed character set detection.

libguess-1.1 employs discrete-finite automata to deduce the character set of the input buffer. The advantage of this is that all character sets can be checked in parallel, and quickly. Right now, libguess passes a byte to each DFA on the same pass, meaning that the winning character set can be deduced as efficiently as possible.

Libguess Copyright

Copyright (c) 2000-2003 Shiro Kawai
Copyright (c) 2005-2010 Yoshiki Yazawa
Copyright (c) 2007-2010 William Pitcock

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the authors nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

hunspell

hunspell 1.3.2 and 1.7.0

Hunspell is the spell checker of LibreOffice, OpenOffice.org, Mozilla Firefox 3 & Thunderbird, Google Chrome, and it is also used by proprietary software packages, like Mac OS X, InDesign, memoQ, Opera and SDL Trados.

Source: <http://hunspell.sourceforge.net>

Main features

- Extended support for language peculiarities; Unicode character encoding, compounding and complex morphology.
- Improved suggestion using n-gram similarity, rule and dictionary based pronunciation data.
- Morphological analysis, stemming and generation.
- Hunspell is based on MySpell and works also with MySpell dictionaries.
- C++ library under GPL/LGPL/MPL tri-license.
- Interfaces and ports: AndroidHunspellService (for Android, based on the Chromium fork of Hunspell), Enchant (Generic spelling library from the Abiword project), XSpell (Mac OS X port, but Hunspell is part of the OS X from version 10.6 (Snow Leopard), and now it is enough to place the Hunspell dictionary files into ~/Library/Spelling or /Library/Spelling for spell checking), Delphi, Java (JNA, JNI), Perl, .NET, Python, Ruby ([1], [2], [3]), UNO, RichEdit.

hunspell Copyright

Copyright (C) 2002-2017 NÃ©meth LÃ¡szlÃ³³

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

Hunspell is based on MySpell which is Copyright (C) 2002 Kevin Hendricks.

Alternatively, the contents of this file may be used under the terms of either the GNU General Public License Version 2 or later (the "GPL"), or the GNU Lesser General Public License Version 2.1 or later (the "LGPL"), in which case the provisions of the GPL or the LGPL are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of either the GPL or the LGPL, and not to allow others to use your version of this file under the terms of the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the GPL or the LGPL. If you do not delete the provisions above, a recipient may use your version of this file under the terms of any one of the MPL, the GPL or the LGPL.

GRIEF usage is under the LGPL 2.1, dynamically linked.

libarchive

libarchive 3.3.3 and 3.6.1

C library and command-line tools for reading and writing tar, cpio, zip, ISO, and other archive formats.

Source: <http://www.libarchive.org>

Library features

- Support for a variety of archive and compression formats.
- Robust automatic format detection, including archive/compression combinations such as tar.gz.
- Zero-copy internal architecture for high performance.
- Streaming architecture eliminates all limits on size of archive, limits on entry sizes depend on particular formats.
- Carefully factored code to minimize bloat when programs are statically linked.
- Growing test suite to verify correctness of new ports.
- Works on most POSIX-like systems (including FreeBSD, Linux, Solaris, etc.)
- Supports Windows, including Cygwin, MinGW, and Visual Studio.
- Support libz, libbz2 and liblzma compression.

Libarchive Copyright

Copyright (c) 2003-2018 <author(s)>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer in this position and unchanged.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR(S) ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

iconv

NetBSD intl/iconv implementation from the Citrus Project

libcitrus, libintl, libiconv and iconv.

Iconv Copyright

Copyright (c) 2003 Citrus Project,
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

libz

Zlib Data compress library.

A Massively Spiffy Yet Delicately Unobtrusive Compression Library (Also Free, Not to Mention Unencumbered by Patents)

zlib 1.2.7 is a general purpose data compression library. All the code is thread safe. The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files <http://tools.ietf.org/html/rfc1950> (zlib format), rfc1951 (deflate format) and rfc1952 (gzip format).

Source: <http://www.zlib.net/>

Acknowledgement

The deflate format used by zlib was defined by Phil Katz. The deflate and zlib specifications were written by L. Peter Deutsch. Thanks to all the people who reported problems and suggested various improvements in zlib; they

are too numerous to cite here.

Libz Copyright

(C) 1995-2012 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly Mark Adler
jloup@gzip.org madler@alumni.caltech.edu

If you use the zlib library in a product, we would appreciate *not* receiving lengthy legal documents to sign. The sources are provided for free but without warranty of any kind. The library has been entirely written by Jean-loup Gailly and Mark Adler; it does not include third-party code.

If you redistribute modified sources, we would appreciate that you include in the file ChangeLog history information documenting your changes. Please read the FAQ for more information on the distribution of modified source versions.

libbzip2

bzip2 is a freely available, patent free, high-quality data compressor.

The bzip2 file compression program was developed by Julian Seward and launched on the 18th of July in 1996. It has remained an open source program, available to all for free, for over twenty two years now. The last stable release was seven years ago. The version 1.0.6 was released on the 20th of September in 2010. bzip2 compression program is based on BurrowsWheeler algorithm. The program can compress files but cannot archive them. Julian Seward is still in charge of maintaining the program. The compression application works on all major operating systems and is available as a BSD-like license. The program uses .bz2 as its filename extension, application/x-bzip2 as the media type on internet and public.archive.bzip2 as the uniform type identifier.

It typically compresses files to within 10% to 15% of the best available techniques (the PPM family of statistical compressors), whilst being around twice as fast at compression and six times faster at decompression.

Source: <http://www.bzip.org/>

Libbzip2 Copyright

This program, "bzip2", the associated library "libbzip2", and all documentation, are copyright (C) 1996-2010 Julian R Seward. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
3. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
4. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Julian Seward, jseward@bzip.org
bzip2/libbzip2 version 1.0.8 of Jan 2024

liblzma

XZ Utils is free general-purpose data compression software with a high compression ratio. XZ Utils were written for POSIX-like systems, but also work on some not-so-POSIX systems. XZ Utils are the successor to LZMA Utils.

The core of the XZ Utils compression code is based on LZMA SDK, but it has been modified quite a lot to be suitable for XZ Utils. The primary compression algorithm is currently LZMA2, which is used inside the .xz container format. With typical files, XZ Utils create 30 % smaller output than gzip and 15 % smaller output than bzip2.

XZ Utils consist of several components

- liblzma is a compression library with an API similar to that of zlib.
- xz is a command line tool with syntax similar to that of gzip.
- xzdec is a decompression-only tool smaller than the full-featured xz tool.
- A set of shell scripts (xzgrep, xzdiff, etc.) have been adapted from gzip to ease viewing, grepping, and comparing compressed files.
- Emulation of command line tools of LZMA Utils eases transition from LZMA Utils to XZ Utils.

While liblzma has a zlib-like API, liblzma doesn't include any file I/O functions. A separate I/O library is planned, which would abstract handling of .gz, .bz2, and .xz files with an easy to use API.

Source: <https://tukaani.org/xz/>

XZ Utils Copyright

Different licenses apply to different files in this package. Here is a rough summary of which licenses apply to which parts of this package (but check the individual files to be sure!):

- liblzma is in the public domain.
- xz, xzdec, and lzmadec command line tools are in the public domain unless GNU getopt_long had to be compiled and linked in from the lib directory. The getopt_long code is under GNU GPLv2.1+.
- The scripts to grep, diff, and view compressed files have been adapted from gzip. These scripts and their documentation are under GNU GPLv2+.
- All the documentation in the doc directory and most of the XZ Utils specific documentation files in other directories are in the public domain.
- Translated messages are in the public domain.
- The build system contains public domain files, and files that are under GNU GPLv2+ or GNU GPLv3+. None of these files end up in the binaries being built.
- Test files and test code in the tests directory, and debugging utilities in the debug directory are in the public domain.
- The extra directory may contain public domain files, and files that are under various free software licenses.

You can do whatever you want with the files that have been put into the public domain. If you find public domain legally problematic, take the previous sentence as a license grant. If you still find the lack of copyright legally problematic, you have too many lawyers.

As usual, this software is provided "as is", without any warranty.

If you copy significant amounts of public domain code from XZ Utils into your project, acknowledging this somewhere in your software is polite (especially if it is proprietary, non-free software), but naturally it is not legally required. Here is an example of a good notice to put into "about box" or into documentation:

This software includes code from XZ Utils <<http://tukaani.org/xz/>>.

The following license texts are included in the following files:

- COPYING.LGPLv2.1: GNU Lesser General Public License version 2.1
- COPYING.GPLv2: GNU General Public License version 2
- COPYING.GPLv3: GNU General Public License version 3

Note that the toolchain (compiler, linker etc.) may add some code pieces that are copyrighted. Thus, it is possible that e.g. liblzma binary wouldn't actually be in the public domain in its entirety even though it contains no copyrighted code from the XZ Utils source package.

If you have questions, don't hesitate to ask the author(s) for more information.

\$Id: contrib.txt,v 1.9 2024/04/18 14:56:48 cvsuser Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Appendix A - Error Codes

Manifest system error codes

errno

The following standard POSIX *errno* are defined as constants. In addition this error code list is extended to cover Win32 systems codes. All constants are defined regardless of the current host system, with unsupported constants being assigned -1.

Constant	Description
E2BIG	Argument list too long.
EACCES	Permission denied.
EADDRINUSE	Address in use.
EADDRNOTAVAIL	Address not available.
EAFNOSUPPORT	Address family not supported.
EAGAIN	Resource unavailable; try again.
EALREADY	Connection already in progress.
EBADF	Bad file descriptor.
EBADMSG	Bad message.
EBUSY	Device or resource busy.
ECANCELED	Operation cancelled.
ECHILD	No child processes.
ECONNABORTED	Connection aborted.
ECONNREFUSED	Connection refused.
ECONNRESET	Connection reset.
EDEADLK	Resource deadlock would occur.
EDESTADDRREQ	Destination address required.
EDOM	Argument out of domain of function.
EDQUOT	Reserved.
EEXIST	File exists.
EFAULT	Bad address.
EFBIG	File too large.
EHOSTUNREACH	Host is unreachable.
EIDRM	Identifier removed.
EILSEQ	Illegal byte sequence.
EINPROGRESS	Operation in progress.
EINTR	Interrupted function.
EINVAL	Invalid argument.
EIO	I/O error.
EISCONN	Socket is connected.
EISDIR	Is a directory.
ELOOP	Too many levels of symbolic links.
EMFILE	Too many open files.
EMLINK	Too many links.
EMSGSIZE	Message too large.
EMULTIHOP	Reserved.
ENAMETOOLONG	Filename too long.
ENETDOWN	Network is down.
ENETRESET	Connection aborted by network.
ENETUNREACH	Network unreachable.
ENFILE	Too many files open in system.
ENOBUFS	No buffer space available.
ENODATA	No message is available on the STREAM.
ENODEV	No such device.
ENOENT	No such file or directory.
ENOEXEC	Executable file format error.
ENOLCK	No locks available.
ENOLINK	Reserved.
ENOMEM	Not enough space.
ENOMSG	No message of the desired type.
ENOPROTOOPT	Protocol not available.

Constant	Description
ENOSPC	No space left on device.
ENOSR	No STREAM resources.
ENOSTR	Not a STREAM.
ENOSYS	Function not supported.
ENOTCONN	The socket is not connected.
ENOTDIR	Not a directory.
ENOTEMPTY	Directory not empty.
ENOTSOCK	Not a socket.
ENOTSUP	Not supported.
ENOTTY	Inappropriate I/O control operation.
ENXIO	No such device or address.
EOPNOTSUPP	Operation not supported on socket.
EOVERFLOW	Value too large to be stored in data type.
EPERM	Operation not permitted.
EPIPE	Broken pipe.
EPROTO	Protocol error.
EPROTONOSUPPORT	Protocol not supported.
EPROTOTYPE	Protocol wrong type for socket.
ERANGE	Result too large.
EROFS	Read-only file system.
ESPIPE	Invalid seek.
ESRCH	No such process.
ESTALE	Reserved.
ETIME	Stream ioctl() timeout.
ETIMEDOUT	Connection timed out.
ETXTBSY	Text file busy.
EWOULDBLOCK	Operation would block (can equal EAGAIN).
EXDEV	Cross-device link.

Error Codes

\$Id: appendixa.txt,v 1.3 2014/10/31 01:09:02 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Appendix B - Backus Naur Form

Language Backus Naur Form (BNF)

Note that the Grief compiler is based upon a modified C11 grammar, which shall generally error when functionality not implementation is referenced.

Despite this fact, in a number of cases it has been seen that source may successfully compile when including syntax which are not explicitly supported resulting in unexpected execution.

Please consult the *Macro Compatibility* Section for specific details and if able report details of your specific case.

Syntax

```

translation_unit      : external_decl
                      | translation_unit external_decl
                      ;
external_decl        : function_definition
                      | decl
                      ;
function_definition : decl_specs declarator decl_list compound_stat
                      | declarator decl_list compound_stat
                      | decl_specs declarator compound_stat
                      | declarator compound_stat
                      ;
decl                : decl_specs init_declarator_list ';'
                      | decl_specs ';'
                      ;
decl_list           : decl
                      | decl_list decl
                      ;
decl_specs          : storage_class_spec decl_specs
                      | storage_class_spec
                      | type_spec decl_specs
                      | type_spec
                      | type_qualifier decl_specs
                      | type_qualifier
                      ;
storage_class_spec : 'auto' | 'register' | 'static' | 'extern' | 'typedef'
                      ;
type_spec           : 'void' | 'char' | 'short' | 'int' | 'long' | 'float'
                      | 'double' | 'signed' | 'unsigned'
                      | struct_or_union_spec
                      | enum_spec
                      | typedef_name
                      ;
type_qualifier      : 'const' | 'volatile'
                      ;
struct_or_union_spec : struct_or_union id '{' struct_decl_list '}'
                      | struct_or_union     '{' struct_decl_list '}'
                      | struct_or_union id
                      ;
struct_or_union     : 'struct' | 'union'
                      ;
struct_decl_list    : struct_decl
                      | struct_decl_list struct_decl
                      ;
init_declator_list  : init_declarator
                      | init_declator_list ',' init_declarator
                      ;
init_declarator     : declarator
                      | declarator '=' initializer
                      ;

```

```

struct_decl : spec_qualifier_list struct_declarator_list ';' ;
spec_qualifier_list : type_spec spec_qualifier_list
| type_spec
| type_qualifier spec_qualifier_list
| type_qualifier
;
struct_declarator_list : struct_declarator
| struct_declarator_list ',' struct_declarator
;
struct_declarator : declarator
| declarator ':' const_exp
|           ':' const_exp
;
enum_spec : 'enum' id '{' enumerator_list '}'
| 'enum'      '{' enumerator_list '}'
| 'enum' id
;
enumerator_list : enumerator
| enumerator_list ',' enumerator
;
enumerator : id
| id '=' const_exp
;
declarator : pointer direct_declarator
| direct_declarator
;
direct_declarator : id
| '(' declarator ')'
| direct_declarator '[' const_exp ']'
| direct_declarator '['      ']'
| direct_declarator '(' param_type_list ')'
| direct_declarator '(' id_list ')'
| direct_declarator '('      ')'
;
pointer : '*' type_qualifier_list
| '*'
| '*' type_qualifier_list pointer
| '*' pointer
;
type_qualifier_list : type_qualifier
| type_qualifier_list type_qualifier
;
param_type_list : param_list
| param_list ',' '...'
;
param_list : param_decl
| param_list ',' param_decl
;
param_decl : decl_specs declarator
| decl_specs abstract_declarator
| decl_specs
;
id_list : id
| id_list ',' id
;
initializer : assignment_exp
| '{' initializer_list '}'
| '{' initializer_list ',' '}'
;
initializer_list : initializer
| initializer_list ',' initializer
;
type_name : spec_qualifier_list abstract_declarator
| spec_qualifier_list
;

```

```

; : - - - - -
abstract_declarator : pointer
| pointer direct_abstract_declarator
| direct_abstract_declarator
;

direct_abstract_declarator : '(' abstract_declarator ')'
| direct_abstract_declarator '[' const_exp ']'
| direct_abstract_declarator '[' const_exp ']'
| direct_abstract_declarator '[' ']'
| direct_abstract_declarator '[' ']'
| direct_abstract_declarator '(' param_type_list ')'
| direct_abstract_declarator '(' param_type_list ')'
| direct_abstract_declarator '(' ')'
| direct_abstract_declarator '(' ')'
;

typedef_name : id
;

stat : labeled_stat
| exp_stat
| compound_stat
| selection_stat
| iteration_stat
| jump_stat
;

labeled_stat : id '::' stat
| 'case' const_exp '::' stat
| 'default' '::' stat
;

exp_stat : exp ';'
| ';'
;

compound_stat : '{}' decl_list stat_list '{}'
| '{}' stat_list '{}'
| '{}' decl_list '{}'
| '{} {}'
;

stat_list : stat
| stat_list stat
;

selection_stat : 'if' '(' exp ')' stat
| 'if' '(' exp ')' stat 'else' stat
| 'switch' '(' exp ')' stat
;

iteration_stat : 'while' '(' exp ')' stat
| 'do' stat 'while' '(' exp ')' ';' stat
| 'for' '(' exp ';' exp ';' exp ')' stat
| 'for' '(' exp ';' exp ';' ')' stat
| 'for' '(' exp ';' ';' exp ')' stat
| 'for' '(' ';' exp ';' exp ')' stat
| 'for' '(' ';' ';' exp ';' ')' stat
| 'for' '(' ';' ';' exp ')' stat
| 'for' '(' ';' ';' ')' stat
;

jump_stat : 'continue' ';'
| 'break' ';'
| 'return' exp ';'
| 'return' ';'
;

exp : assignment_exp
| exp ',' assignment_exp
;

assignment_exp : conditional_exp
| unary_exp assignment_operator assignment_exp
;

assignment_operator : '=' | '*=' | '/=' | '%=' | '+=' | '-=' |
| '<=' | '>=' | '&=' | '^=' | '|=' | '|<=' |
| '|>=' |
;
```

```

-->
;

conditional_exp      : logical_or_exp
| logical_or_exp '?' exp ':' conditional_exp
;

const_exp            : conditional_exp
;

logical_or_exp       : logical_and_exp
| logical_or_exp '||' logical_and_exp
;

logical_and_exp      : inclusive_or_exp
| logical_and_exp '&&' inclusive_or_exp
;

inclusive_or_exp     : exclusive_or_exp
| inclusive_or_exp '||' exclusive_or_exp
;

exclusive_or_exp      : and_exp
| exclusive_or_exp '^' and_exp
;

and_exp               : equality_exp
| and_exp '&' equality_exp
;

equality_exp          : relational_exp
| equality_exp '==' relational_exp
| equality_exp '!=' relational_exp
;

relational_exp        : shift_expression
| relational_exp '<' shift_expression
| relational_exp '>' shift_expression
| relational_exp '<=' shift_expression
| relational_exp '>=' shift_expression
;

shift_expression       : additive_exp
| shift_expression '<<' additive_exp
| shift_expression '>>' additive_exp
;

additive_exp          : mult_exp
| additive_exp '+' mult_exp
| additive_exp '-' mult_exp
;

mult_exp              : cast_exp
| mult_exp '*' cast_exp
| mult_exp '/' cast_exp
| mult_exp '%' cast_exp
;

cast_exp              : unary_exp
| '(' type_name ')' cast_exp
;

unary_exp             : postfix_exp
| '++' unary_exp
| '--' unary_exp
| unary_operator cast_exp
| 'sizeof' unary_exp
| 'sizeof' '(' type_name ')'
;

unary_operator         : '&' | '*' | '+' | '-' | '~' | '!'
;

postfix_exp           : primary_exp
| postfix_exp '[' exp ']'
| postfix_exp '(' argument_exp_list ')'
| postfix_exp '(' ')'
| postfix_exp '.' id
| postfix_exp '->' id
| postfix_exp '++'
| postfix_exp '--'
;

```

```
primary_exp      : id
                  | const
                  | string
                  | '(' exp ')'
                  ;

argument_exp_list : assignment_exp
                  | argument_exp_list ',' assignment_exp
                  ;

const            : int_const
                  | char_const
                  | float_const
                  | enumeration_const
                  ;
```

\$Id: appendixb.txt,v 1.3 2014/10/31 01:09:04 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Appendix C - ASCII Chart

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as *a* or *@* or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose.

The following chart contains all 128 ASCII decimal (dec), octal (oct), hexadecimal (hex) and character (ch) codes and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat dated.

Decimal	Octal	Hexadecimal	Description
0	0	00	NUL (null)
1	1	01	SOH (start of header)
2	2	02	STX (start of text)
3	3	03	ETX (end of text)
4	4	04	EOT (end of transmission)
5	5	05	ENQ (enquiry)
6	6	06	ACK (acknowledge)
7	7	07	BEL (bell)
8	10	08	BS (backspace)
9	11	09	HT (horizontal tab)
10	12	0a	LF (line feed/new line)
11	13	0b	VT (vertical tab)
12	14	0c	FF (form feed/new page)
13	15	0d	CR (carriage return)
14	16	0e	SO (shift out)
15	17	0f	SI (shift in)
16	20	10	DLE (data link escape)
17	21	11	DC1 (device control 1)
18	22	12	DC2 (device control 2)
19	23	13	DC3 (device control 3)
20	24	14	DC4 (device control 4)
21	25	15	NAK (negative acknowledge)
22	26	16	SYN (synchronous idle)
23	27	17	ETB (end of transmission block)
24	30	18	CAN (cancel)
25	31	19	EM (end of medium)
26	32	1a	SUB (substitute)
27	33	1b	ESC (escape)
28	34	1c	FS (file separator)
29	35	1d	GS (group separator)
30	36	1e	RS (record separator)
31	37	1f	US (unit separator)
32	40	20	(space)
33	41	21	!
34	42	22	\
35	43	23	#
36	44	24	\$
37	45	25	%
38	46	26	&
39	47	27	'
40	50	28	(
41	51	29)
42	52	2a	*
43	53	2b	+
44	54	2c	,
45	55	2d	-
46	56	2e	.
47	57	2f	/
48	60	30	0
49	61	31	1
50	62	32	2
51	63	33	3
52	64	34	4

Decimal	Octal	Hexidecimal	Description
53	65	35	5
54	66	36	6
55	67	37	7
56	70	38	8
57	71	39	9
58	72	3a	:
59	73	3b	;
60	74	3c	<
61	75	3d	=
62	76	3e	>
63	77	3f	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4a	J
75	113	4b	K
76	114	4c	L
77	115	4d	M
78	116	4e	N
79	117	4f	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5a	Z
91	133	5b	[
92	134	5c	\
93	135	5d]
94	136	5e	^
95	137	5f	-
96	140	60	'
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6a	j
107	153	6b	k
108	154	6c	l
109	155	6d	m
110	156	6e	n
111	157	6f	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s

Decimal	Octal	Hexidecimal	Description
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7a	z
123	173	7b	{
124	174	7c	
125	175	7d	}
126	176	7e	~
127	177	7f	DEL (delete)

ASCII Chart

\$Id: appendixc.txt,v 1.3 2014/10/31 01:09:04 ayoung Exp \$

To send feedback on this topic email: griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

Appendix D - BRIEF Macros

The following is the list of available macros for Brief compatibility, formally known as CBRIEF following the later 3.x releases.

Macro	Description
abort	Unconditionally abort BRIEF.
above	Tests greater than for unsigned ints.
above_eq	Tests greater than or equal for unsigned ints.
assign_to_key	Assign command to key or key sequence.
atoi	Convert character to ascii or string to integer.
attach_buffer	Attach a buffer to the current window.
autoload	Notify which file macros or globals are in.
backspace	Move left and delete char.
beep	Make a noise.
beginning_of_line	Move cursor to first column of current line.
below	Tests less than for unsigned ints.
below_eq	Tests less than or equal for unsigned ints.
borders	Toggle window borders.
call_registered_macro	Invokes all registered macros of a given type.
cd	Change current directory.
change_window	Make a different window active.
close_window	Close a window.
color	Set BRIEF screen colors.
compress	Compress consecutive whitespace characters.
copy	Copy marked area to scrap.
copy_keyboard	Copy all or part of a keyboard map.
create_buffer	Create a buffer.
create_edge	Create a new window.
create_tiled_window	Creates a tiled window.
create_window	Create an overlapping window.
cut	Cut the marked area to scrap.
date	Get the system date and time.
del	Delete file.
delete_block	Delete the marked block.
delete_buffer	Remove a buffer.
delete_char	Delete character at cursor location.
delete_edge	Deletes a tiled window edge, closing a window.
delete_line	Deletes current line.
delete_macro	Delete all macros in a macro file from memory.
delete_to_eol	Deletes from cursor to end of line.
delete_window	Delete an overlapping window.
display_windows	Displays tiled windows.
distance_to_tab	Give the number of characters to next tab stop.
dos	Execute a DOS command.
down	Move cursor down one row.
drop_anchor	Begin a marked area.
drop_bookmark	Make a bookmark at the current location.
edit_file	Edit a file.
end_of_buffer	Move to the end of the buffer.
end_of_line	Move to the last character on the current line.
end_of_window	Move to the last line in the window.
error	Show error message on prompt line.
execute_macro	Run a macro.
exist	Check to see if a file exists.
exit	Exit to DOS or return to BRIEF.
file_pattern	Set up a pattern for file searching.
find_file	Search for file names matching file_pattern.
first_time	Detects the first time a macro is executed.
get_parm	Get a passed parameter, prompting if needed.
getwd	Get the working directory name.
goto_bookmark	Move to bookmark, or just get the location.

Macro	Description
goto_line	Move to a particular line.
goto_old_line	Move to line before buffer modification.
index	Locate first occurrence of one string in another.
inq_assignment	Get key assignment for function.
inq_borders	Get borders setting.
inq_brief_level	Get number of copies of BRIEF in memory.
inq_buffer	Get current buffer id.
inq_called	Get name of calling macro.
inq_btn2_action	Get the action attached to mouse button 2.
inq_cmd_line	Get text currently on prompt line.
inq_command	Get name of last command invoked from keyboard.
inq_ctrl_state	Get the state of window controls.
inq_environment	Get a value from the DOS environment.
inq_idle_default	Get the default amount of idle time.
inq_idle_time	Get number of seconds since user pressed a key.
inq_kbd_char	Get status of keyboard input buffer.
inq_kbd_flags	Get BIOS keyboard flags.
inq_keyboard	Get current keyboard id.
inq_keystroke_macro	Is keystroke macro record/playback active?.
inq_line_length	Get max line length.
inq_local_keyboard	Get local keyboard id for current buffer.
inq_macro	Find out if a macro exists.
inq_mark_size	Get number of characters in marked area.
inq_marked	Get the boundaries and type of a marked area.
inq_message	Get current message displayed on the prompt line.
inq_mode	Get typing mode: insert or overstrike.
inq_modified	Get modified status for buffer.
inq_mouse_action	Get the name of the current mouse event handler.
inq_msg_level	Get current message level.
inq_names	Get file and buffer name.
inq_position	Get current position.
inq_scrap	Get scrap buffer id and mark type.
inq_screen_size	Get screen size.
inq_system	Find out if a buffer is a system buffer.
inq_top_left	Get buffer positioning information.
inq_views	Get number of windows containing a buffer.
inq_window	Get window id.
inq_window_color	Get window color (borderless only).
inq_window_info	Get buffer id, size and type for a given window id.
inq_window_size	Get height, width, and horiz scroll amount.
insert	Insert a formatted string into current buffer.
insert_mode	Toggle insert/overstrike mode.
int_to_key	Convert an integer to mnemonic key string.
key_to_int	Convert mnemonic key string to an integer.
keyboard_flush	Clear all waiting keyboard input.
keyboard_pop	Pop a keyboard from the keyboard stack.
keyboard_push	Push a keyboard onto the keyboard stack.
keyboard_typeables	Assign self_insert to all typeable keys.
left	Move cursor left.
load_macro	Load a macro file.
lower	Convert a string to all lower case characters.
ltrim	Remove leading white space from string.
mark	Toggle mark state.
message	Display a status message on the prompt line.
move_abs	Move to an absolute location in the buffer.
move_edge	Move a tiled window boundary.
move_rel	Move to a relative location in the buffer.
next_buffer	Get buffer id of next buffer.
next_char	Move cursor to next character.
next_window	Get window id of next tiled window.
nothing	Do nothing; bound to unassigned keys.
output_file	Set name of output file for current buffer.
page_down	Move down a page.
page_up	Move up a page.

Macro	Description
paste	Insert scrap buffer at cursor location.
pause_on_error	Toggle pausing execution on every error.
playback	Playback remembered keystrokes.
prev_char	Move to previous character.
print	Print the marked area.
printf	Print message to DOS standard output (debug).
process	Process keystrokes until exit.
push_back	Push back a keystroke for processing elsewhere.
put_parm	Set the value of a passed parameter.
raise_anchor	Remove a marked area.
read	Get characters starting at current location.
read_char	Get the character from the current location.
read_file	Insert a copy of a file into current buffer.
redo	Redo the last undo command.
refresh	Redraw the screen.
register_macro	Add a new registered macro.
remember	Start recording keystrokes.
restore_position	Go back to most recent save_position location.
returns	Just set return value.
right	Move cursor right.
rindex	Locate last occurrence of one string in another.
save_keystroke_macro	Save keystroke macro to file.
save_position	Save a position for use by restore_position.
search_back	Search backward for a pattern.
search_case	Change the case-sensitivity flag.
search_fwd	Search forward for a pattern.
search_string	Search a string for another string.
self_insert	Insert key value (assigned to typeable keys).
set_backup	Controls creation of backup files.
set_btn2_action	Sets the action for mouse button 2.
set_buffer	Sets the current buffer.
set_calling_name	Sets the name of calling function.
set_ctrl_state	Sets the state of window controls.
set_mouse_action	Sets the name of the mouse event handler.
set_mouse_type	Sets the type of mouse.
set_msg_level	Sets the message level.
set_scrap_info	Sets the scrap information.
set_top_left	Position buffer in a window.
set_window	Sets the current window.
sprintf	Formatted print into a string.
strlen	Get length of a string.
substr	Extract a sub-string from a string.
swap_anchor	Exchange cursor and anchor locations.
tabs	Set tab stops.
time	Get current system time.
top_of_buffer	Move to top of buffer.
top_of_window	Move to top of window.
transfer	Direct buffer-to-buffer text transfer.
translate	Translate pattern to a replacement string.
trim	Remove trailing whitespace from string.
undo	Undo the last command.
unregister_macro	Remove a registered macro.
up	Move cursor up one line.
upper	Convert string to upper case characters.
use_local_keyboard	Attaches a local keyboard to current buffer.
use_tab_char	Controls whether tabs or spaces are used.
version	Get version number of BRIEF.
window_color	Set background color for borderless window.
write_block	Write marked block to a file.

To send feedback on this topic **email:** griefedit@gmail.com

Copyright © Adam Young All Rights Reserved.

INDEX

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

\$#!

appendixd.txt
prim_arith.txt
prim_buffer.txt
prim_callback.txt
prim_debug.txt
prim_dialog.txt
prim_env.txt
prim_file.txt
prim_kbd.txt
prim_macro.txt
prim_proc.txt
prim_scrap.txt
prim_screen.txt
prim_search.txt
prim_spell.txt
prim_string.txt
prim_syntax.txt
prim_var.txt
prim_window.txt
quickstart.txt

!
!=
%
%=
&
&&
&=
*=
+
++
+=
-
--
-=
/
/=
<
<<
<<=
<=
<=>
=
==
>
>=
>>
>>=
^
^=
__breaksw
__lexicalblock
__regress_op
__regress_replacement
_bad_key
_chg_properties
_default
_extension
_fatal_error
_init
_invalid_key
_prompt_begin
_prompt_end
_startup_complete
|
|=
||
~

A

A Quick Macro Tutorial
abort
above
above_eq
abs
access
acos
Appendix A-Error Codes
Appendix B-Backus Naur Form
Appendix C-ASCII Chart
Appendix D-BRIEF Macros
arg_list
Arithmetic Operators
 language.txt
 prim_arith.txt
array
Array Subscripting
ASCII Chart
asin
assign_to_key
Assignment Operators
atan
atan2
atoi
attach_buffer
attach_syntax
Authors
autoload

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

B

backspace
Backus Naur Form
basename
Basics
beep
beginning_of_line
below
below_eq
Bitwise Logical Operators
Bitwise Shift Operators
bless
bookmark_list
BookMarks
bool
borders
BPACKAGES
Braces
break
break statement
Brief
Buffer Attributes
Buffer Content
Buffer Flags
Buffer Identifiers
Buffer List
Buffer Primitives
Buffers
Built-in Functions

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

C

call_registered_macro
Callbacks
car
case
catch
cd
cdr
ceil
ctime
change_window
change_window_pos
Character Literals
Character Types
characterat
chdir
chmod
chown
close_window
color
color_index
Comma Operator
Command Prompt
command_list
Commands
Comments
Common Modules
compare
compare_files
Compatibility
Compilation Model
Compiler Usage
Compound Statements
compress
Condition evaluation
Conditional Compilation
Conditional Operator
connect
const
Constants
 prim_buffer.txt
 prim_macro.txt
continue
continue statement
copy
copy_ea_info
copy_keyboard
copy_screen
Copyright
cos
cosh
create_buffer
create_char_map
create_dictionary
create_edge
create_menu_window
create_nested_buffer
create_syntax
create_tiled_window
create_window
Crisp
ctype
cursor
cut
Cut and Paste
cvt_to_object

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

D

date
debug
debug_support
Debugging
Debugging Primitives
Declarations
declare
define_keywords
del
delete_block
delete_bookmark
delete_buffer
delete_char
delete_dictionary
delete_edge
delete_line
delete_macro
delete_nth
delete_to_eol
delete_window
detach_syntax
Dialog Primitives
dialog_create
dialog_delete
dialog_exit
dialog_run
Dialogs
dict_clear
dict_delete
dict_each
dict_exists
dict_keys
dict_list
dict_name
dict_values
Dictionaries
diff_strings
Directives
dirname
disconnect
display_mode
display_windows
distance_to_indent
distance_to_tab
do
do-while statement
Documentation
dos
double
down
dprintf
drop_anchor
drop_bookmark

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

E

echo_line
edit_file
edit_file2
ega
else
Encoding
end_anchor
end_of_buffer
end_of_line
end_of_window
Enumerated Types
Environment Primitives
Equality Operators
errno
appendixa.txt
prim_macro.txt

error
Error Codes
Escape Sequences
execute_macro
exist
exit
exp
expandpath
Expression Statement
Expressions
extags
extern

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

F

fabs
fclose
Features
feof
ferror
fflush
File and Buffer Manipulation
File Extension Macros
File Inclusion
File Modes
File Primitives
file_canon
file_glob
file_match
file_pattern
filename_match
filename_realpath
finally
find_file
find_file2
find_line_flags
find_macro
find_marker
ioctl
first_time
firstof
flex
float
Float Types
Floating Point Literals
flock
floor
fmktemp
fmod
fopen
for
for statement
foreach
format
fread
frexp
fseek
fstat
fstype
ftell
ftest
truncate
Function Calls
Function Declarations
Function Prototypes
Function Reference
Functions
fwrite

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

G

get_color
get_color_pair
get_mouse_pos
get_nth
get_parm
get_property
get_region
get_term_characters
get_term_feature
get_term_features
get_term_keyboard
getenv
geteuid
 getopt
getpid
getsubopt
getuid
getwd
glob
global
gmtime
goto_bookmark
goto_line
goto_old_line
GRBACKUP
GRDICTIONARIES
GRDICTIONARY
GRFILE
GRFLAGS
GRHELP
GRIEF Macros
GRIEF Software License
GRIEF-The Glorious Reconfigurable Interactive Editing Facility
grief_version
GRINIT_FILE
GRKBDPATH
GRLEVEL
GRLOG_FILE
GRPATH
GRPROFILE
GRRESTORE_FILE
GRSTATE_DB
GRSTATE_FILE
GRTEMPLATE
GRTERM
GRTERMCAP
GRTMP
GRVERSIONMAJOR
GRVERSIONMINOR
GRVERSIONS

H

Help
Hilite Regions
hilite_create
hilite_delete
hilite_destroy
History
hunspell

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

I

iconv
Identifiers
if
if statement
Implementation
Increment and Decrement Operators
index
input_mode
inq_assignment
inq_attribute
inq_backup
inq_backup_option
inq_borders
inq_brief_level
inq_btn2_action
inq_buffer
inq_buffer_flags
inq_buffer_title
inq_buffer_type
inq_byte_pos
inq_called
inq_char_map
inq_char_timeout
inq_clock
inq_cmd_line
inq_color
inq_command
inq_connection
inq_ctrl_state
inq_debug
inq_dialog
inq_display_mode
inq_echo_format
inq_echo_line
inq_encoding
inq_environment
inq_feature
inq_file_change
inq_file_magic
inq_font
inq_hilite
inq_home
inq_hostname
inq_idle_default
inq_idle_time
inq_indent
inq_kbd_char
inq_kbd_flags
inq_kbd_name
inq_keyboard
inq_keystroke_macro
inq_keystroke_status
inq_line_col
inq_line_flags
inq_line_length
inq_lines
inq_local_keyboard
inq_macro
inq_macro_history
inq_margins
inq_mark_size
inq_marked
inq_marked_size
inq_message
inq_mode
inq_modified
inq_module
inq_mouse_action
inq_mouse_type
inq_msg_level
inq_names
inq_position

inq_process_position
inq_profile
inq_prompt
inq_remember_buffer
inq_ruler
inq_scrap
inq_screen_size
inq_symbol
inq_syntax
inq_system
inq_tab
inq_tabs
inq_terminator
inq_time
inq_tmpdir
inq_top_left
inq_username
inq_vfs_mounts
inq_views
inq_window
inq_window_buf
inq_window_color
inq_window_flags
inq_window_info
inq_window_infox
inq_window_priority
inq_window_size
insert
insert_buffer
insert_mode
insert_process
insertf
int
int_to_key
Integer Literals
Integer Types
Introduction
is_array
is_float
is_integer
is_list
is_null
is_string
is_type
isalnum
isalpha
isascii
isblank
isclose
iscntrl
iscsym
isdigit
isfinite
isgold
isgraph
isinf
islower
isnan
isprint
ispunct
isspace
isupper
isword
isxdigit
Iteration Statements
itoa

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

J

Jump Statements

K

key_list
 key_to_int
 Keyboard Primitives
 keyboard_flush
 keyboard_pop
 keyboard_push
 keyboard_typeables
 Keyboards
 Keywords

L

Language Specification
 lastof
 Lazy Evaluation
 Idexp
 left
 length_of_list
 Lexical elements
 libarchive
 libbzip2
 libcharudet
 libguess
 liblzma
 libmagic
 Library Reference
 libregex
 libteken
 libz
 Line Numbers
 link
 list
 List Primitives
 List Types
 list_each
 list_extract
 list_of_dictionaries
 list_reset
 Literals
 load_keystroke_macro
 load_macro
 localtime
 log
 log10
 Logical Operators
 lower
 Istat
 Itrim

M

macro
 Macro Constants
 Macro Language differences
 Macro Language Primitives
 Macro Primitives
 Macro Resources
 Macro Tutorial
 macro_list

Macros
macros.txt
prim_arith.txt
prim_buffer.txt
prim_callback.txt
prim_debug.txt
prim_dialog.txt
prim_env.txt
prim_file.txt
prim_kbd.txt
prim_list.txt
prim_macro.txt
prim_misc.txt
prim_movement.txt
prim_proc.txt
prim_scrap.txt
prim_screen.txt
prim_search.txt
prim_spell.txt
prim_string.txt
prim_syntax.txt
prim_var.txt
prim_window.txt

main
Main Screen
make_list
make_local_variable
makedepend
mandoc
mark
mark_line
Marked Regions
message
Message Generation
Miscellaneous Primitives
mkdir
mktemp
mode_string
modf
module
Modules
move_abs
move_edge
move_rel
Movement
Movement Primitives

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

N

ND+-Natural Docs Plus
next_buffer
next_char
next_window
Notation
nothing
nth

O

Oniguruma
Operator Precedence
Operators
output_file

P

page_down
page_up
Parameter List
parse_filename
paste
pause
pause_on_error
pause_on_message
 perror
playback
Polymorphic Types
pop
post++
post--
pow
Predefined Symbols
Preprocessor
prev_char
previous_buffer
print
printf
process
Process Buffers
Process Management Primitives
process_mouse
profile
Punctuators
push
push_back
put_nth
put_parm
putenv

Q

Quick Start
quote_list
quote_regexp

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

R

raise_anchor
rand
re_comp
re_delete
re_result
re_search
re_syntax
re_translate
read
read_char
read_ea
read_file
readlink
realpath
Record and Playback
redo
redraw
ref_parm
refresh
register
register_macro
Registered Macros
Regular Expressions
Relational Operators
reload_buffer
remember
remove
rename
replacement
Replacement Macros
Replacment Macros
require
reregister_macro
restore_position
return
return statement
Return Value
returns
returns statement
right
rindex
rmdir
rtrim

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

S

save_keystroke_macro
save_position
Scope
Scope Rules
Scrap Buffer
Scrap Buffers
Scrap Primitives
Screen Primitives
screen_dump
Search and Replace
Search and Translate Primitives
search_back
search_case
search_fwd
search_list
search_string
searchpath
Selection Statements
self_insert
send_signal
set_attribute
set_backup
set_backup_option
set_binary_size
set_btn2_action
set_buffer
set_buffer_cmap
set_buffer_flags
set_buffer_title
set_buffer_type
set_calling_name
set_char_timeout
set_color
set_color_pair
set_ctrl_state
set_ea
set_echo_format
set_encoding
set_feature
set_file_magic
set_font
set_idle_default
set_indent
set_kbd_name
set_line_flags
set_macro_history
set_margins
set_mouse_action
set_mouse_type
set_msg_level
set_process_position
set_property
set_ruler
set_scrap_info
set_syntax_flags
set_tab
set_term_characters
set_term_feature
set_term_features
set_term_keyboard
set_terminator
set_top_left
set_window
set_window_cmap
set_window_flags
set_window_priority
set_wm_name
Setup
shell
shift
Signals
sin

sinh
sleep
sort_buffer
sort_list
Source Code
Source Information
Special Purpose Macros
Spell Checker Primitives
spell_buffer
spell_control
spell_dictionary
spell_distance
spell_string
spell_suggest
splice
split
split_arguments
sprintf
sqrt
srand
sscanf
Startup and main
stat
Statements
static
Storage Class
strcasecmp
strcasestr
strcmp
strerror
strfilecmp
strftime
string
String Literals
String Primitives
String Types
string_count
strlen
strnlen
strpbrk
strpop
strrstr
strsignal
strstr
strtod
strtodf
strol
strverscmp
substr
suspend
swap_anchor
switch
switch statement
symlink
Syntax Highlighting
Syntax Highlighting Primitives
syntax_build
syntax_column_ruler
syntax_rule
syntax_token
System Buffer

Index

\$#! · 0-9 · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

T

tabs
tagdb_close
tagdb_open
tagdb_search
tan
tanh
Text Editing
Third Party Packages
throw
time
tokenize
Tokens
top_of_buffer
top_of_window
transfer
translate
translate_pos
TRE
trim
try
Type Specifiers
typeof
Types
Types of Macros

U

ucpp
umask
uname
Unary Arithmetic Operators
undo
Undo and Redo
unlink
unregister_macro
unshift
up
upper
use_local_keyboard
use_tab_char

V

Variable Declaration Primitives
version
vfs_mount
vfs_unmount
view_screen

W

wait
wait_for
watch
while
while statement
widget_get
widget_set
Window Manipulation
Window Primitives
window_color
Windows
write_block

write_buffer