
FelixCAD 2.1

Customization and Programmer's Guide

Disclaimer and Limited Warranty

This document and the software contained herein may not be reproduced in any fashion or on any media without the explicit written permission of FCAD Inc..

EXCEPT AS OTHERWISE PROVIDED IN THIS AGREEMENT, FCAD INC. SPECIFICALLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE., TO DEFECTS IN THE DISKETTE OR OTHER PHYSICAL MEDIA AND DOCUMENTATION, OR TO OPERATION OF THE PROGRAMS AND ANY PARTICULAR APPLICATION OR USE OF THE PROGRAMS. IN NO EVENT SHALL FCAD INC. BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGE INCLUDING, BUT NOT LIMITED TO, SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR OTHER DAMAGES. ALL LIABILITY BY FCAD INC. HEREUNDER IS EXPRESSLY LIMITED TO ANY AMOUNTS PAID TO FCAD INC. PURSUANT TO THIS AGREEMENT.

Notwithstanding any provision of this Agreement, FCAD Inc. owns and retains all title and ownership of all intellectual property, including but not limited to all software and any and all derivative software; all documentation, manuals and related materials; all master diskettes or CD ROMs on which such software may be transferred, and all copies of any such diskettes or CD ROMs, and any and all derivative works of FCAD Product. FCAD Inc. does not transfer any portion of such title and ownership, or any goodwill associated therewith; and this Agreement shall not be construed to grant any right or license, whether by implication, estoppel or otherwise, except as expressly provided herein.

FCAD INC.

7428 Redwood Blvd.
Novato CA 94945

FELIX C.A.T. GmbH
Nestorstraße 36a
10709 Berlin, Germany

© Copyright 1995-96 Felix CAT GmbH, FCAD, Inc.
and FCAD Canada Software, Inc.
All Rights Reserved.

FCAD, GDE, FelixCAD are trademarks of Felix CAT GmbH.
MS, MS-DOS, and Windows as well as WMF are either registered trademarks or trademarks of Microsoft Corporation and are registered in the United States of America and/or in other countries.

DXF and AutoCAD are a trademarks of Autodesk, Inc.
Adobe Acrobat and PostScript are a trademarks of Adobe System, Inc.
All other product names or trademarks mentioned herein are trademarks of the respective owners.

Table of Contents

Introduction	1
Manual Organization	1
CHAPTER 1	3
Directory Structure and File Type Overview File-Locking and Auto-Save	3
Directory Structure and Path Configuration	3
The Directories Configuration Dialog	3
Directory Structure and Paths	4
File Types	5
Overview	5
File Locking and Auto-Saving	6
File Locking	6
Auto-Save	7
CHAPTER 2	8
Desktop Configuration: Colors and Command Line Font	8
Desktop Colors	8
Command Line Font	10
Command line parameters for FELIXCAD.EXE	10
CHAPTER 3	11
Command Customization: Alias Commands, Hotkeys and Macros	11
Alias Commands	11
HotKeys: Function Key Allocations	13
Macros	15
Creating Macros	15
Using the clipboard	15
Utilizing Macros	16
CHAPTER 4	17
Template Drawings	17
Creating a Template Drawing	17

CHAPTER 5	19
Defining and Using Line Types and Hatch Patterns	19
Linetype Definition	19
Line Type File Syntax	19
Hatch Pattern Definition	20
Hatch Pattern File Syntax	20
CHAPTER 6	25
The Dialog and Menu Editor	25
Using Menu and Palette Files	25
Loading Menus and Palettes	26
Creating and Saving Application Resource Files	26
CHAPTER 7	28
Creating and Editing Palettes	28
Creating or Editing Palettes	28
Palette Properties	29
Allocating a Button Label	29
Allocating Commands	30
Help / Status Bar Entries	33
Saving and Closing Palettes	34
Creating and Editing Menus	35
Editing Pull-down Menus	37
Mouse and Keyboard Allocations	38
Editing and Creating Menus	39
Editing a Menu Structure	39
Editing Properties	39
Menu Properties	39
Menu Entry Type	40
Command	41
Style Properties	42
Style	42
Variable / Value	43
Help Parameters	43
Saving and Closing Menu Files	44
CHAPTER 9	47
Creating and Editing Dialog Boxes	47
Creating New Dialog Files / Opening Dialog Files	48

Saving and Closing Dialog Files	48
Editing Dialog Files and Dialog Boxes	48
The Dialog Editing Window	49
Mouse and Keyboard Allocations	50
Size and Coordinates	51
Grid Alignment	51
Dialog Window Properties	51
Caption	52
Position	52
Attributes	52
The Controls	52
Properties / Attributes	54
General Attributes	55
Control Types	57
Standard Action Buttons: OK, Cancel, Help	57
Command Button (Push Button)	57
Bitmap Button	58
Radio Button	59
Check Box	60
List Box	61
Combo Box	62
Group Box	63
Static Control (Label)	63
Text Edit Control	65
Filtered Input Control	67
Image Window Control	68
Slider Control	68
Control Positioning Aids (Grid etc.)	69
Control Alignment	71
Testing a Dialog	72
CHAPTER 10	73
LISP Programming	73
Data Types in FLISP	74
Loading Lisp Files	75
Error Handling and Error Tracing	75
FLISP Function Overview: Thematically ordered	77
Function definition	77
Error Handling	77
System Functions	77
Geometric Utilities	77
User Input	78
Conversion	78
Coordinate System Transformation	78
Display Control	79

Table of Contents

Selection Sets	79
Entity Handling	80
Symbol Tables	80
Extended Entity Data	80
Arithmetical Functions	80
Symbol Handling	81
Text Strings	82
Conversion	82
Equality / Conditional	82
List Manipulation	83
File Handling	84
Function Handling	84
Memory Management	84
Miscellaneous	85
FDT Application Handling	85
Help	85
FLISP Functions: Reference	86
error	87
settrace	87
+ (Addition)	88
- (Subtraction)	88
* (Multiplication)	88
/ (Division)	89
= (equal)	89
/= (not equal)	90
< (less than)	90
<= (less than or equal)	91
> (greater than)	91
>= (greater than or equal)	91
1+ (Increment)	91
1- (Decrement)	92
~ (Bitwise NOT)	92
abs	92
actcmd	93
alert	93
and	94
angle	94
angtof	94
angtos	95
append	96
apply	96
ascii	96
assoc	97
atan	97
atof	98
atoi	98
atom	98

atoms-family	99
boole	99
boundp	100
car	100
cdr	100
caaaaar ... cddddr	101
chr	101
close	101
command	102
cond	102
cons	103
cos	103
defun	103
delcmd	104
distance	105
distof	105
dlg_***	106
entcheck	106
entdel	107
entget	107
entlast	108
entmake	108
entmod	110
entnext	111
entpos	111
entsel	111
entupd	112
eq	112
equal	112
eval	113
exit	113
exp	114
expt	114
fdt	114
findfile	115
fix	115
float	116
flxnames	116
foreach	116
gc	116
gcd	116
getactvport	117
getangle	117
getcorner	118
getdist	119
getenv	119
getfiled	120

Table of Contents

getint	122
getkeyword	122
getorient	122
getpoint	124
getreal	124
getstring	124
getvar	125
graphscr	125
grclear	125
grdraw	126
grrread	126
grtext	128
handent	128
help	128
if	129
initget	129
inters	131
itoa	131
lambda	131
last	132
length	132
list	132
listp	132
load	133
log	133
logand	133
logior	134
lsh	134
mapcar	134
max	134
mem	135
member	135
min	135
minusp	135
nentsel	136
nentselp	136
not	137
nth	137
null	137
numberp	138
open	138
or	138
osnap	139
pi	140
polar	140
prin1	140
princ	140

print	141
progn	141
prompt	141
quit	141
quote	141
read	142
read-char	142
read-line	142
redraw	142
regapp	143
rem	143
repeat	143
reverse	144
rtos	144
set	144
setactvport	145
setfunhelp	145
setq	145
setvar	146
sin	146
sqrt	146
ssadd	146
ssdbno	147
ssdel	147
ssget	147
sslength	149
ssmemb	150
ssname	150
strcase	150
strcat	150
stringsort	151
strlen	151
subst	151
substr	151
sybntos	152
tbldel	152
tblmake	152
tblmod	153
tblnext	154
tblpurge	154
tblrename	154
tblsearch	155
tblset	155
terpri	156
textbox	156
textscr	156
trans	157

type	157
ver	158
wcmatch	158
while	159
write-char	159
write-line	160
xload	160
xunload	160
zerop	161

CHAPTER 11 **162**

Programming Dialog Boxes	162
Design of a Dialog	162
Loading and Displaying a Dialog	164
Initializing Dialog Controls	164
Basic Functions to initialize a Dialog	165
Special Functions to initialize a Dialog	165
Retrieving User Input and Reacting to it	168
The Action Function for Dialog Controls	168
Functions to Evaluate Dialog Control Elements	168
Functions to Terminate a Dialog	169
Overview	169
Loading and Unloading of Dialog Files	169
Opening and Closing Dialog Boxes	169
General Operations for Control Elements	169
List Boxes and Combo Boxes	169
Slider Control	170
Image Controls	170
Dialog Functions: Reference	171
Dlg_DialogDone	171
Dlg_DialogLoad	171
Dlg_DialogNew	172
Dlg_DialogStart	172
Dlg_DialogTerm	173
Dlg_DialogUnload	173
Dlg_ImageBmp	173
Dlg_ImageEnd	174
Dlg_ImageFill	174
Dlg_ImageStart	175
Dlg_ImageVector	176
Dlg_ImageWmf	176
Dlg_ListAdd	177
Dlg_ListEnd	178
Dlg_ListGet	178
Dlg_ListSetColumnWidth	178

Dlg_ListSetTabstops	179
Dlg_ListStart	179
Dlg_SliderGet	180
Dlg_SliderSet	181
Dlg_TileAction	181
Dlg_TileClientData	182
Dlg_TileDimX	183
Dlg_TileDimY	183
Dlg_TileGet	184
Dlg_TileMode	184
Dlg_TileSet	185
Dlg_TileSetFont	186
Overview: Alphabetically Ordered	187

CHAPTER 12 **189**

System and Drawing File Variables	189
Global and Local Variables	189
Retrieving and Setting the Variables	189
Scheme of the System Variables Reference	190
Global Variables: Reference	192
ACTDB	192
AREA	192
CDATE	192
CIRCLERAD	193
CIRCLERES	193
CMDACTIVE	193
CMDECHO	194
CMDNAMES	194
DATE	194
DEFANGBASE	195
DEFANGDIR	195
DEFAUNITS	195
DEFAUPREC	195
DEFDIMZIN	196
DEFLUNITS	196
DEFLUPREC	196
DIASTAT	197
DISTANCE	197
ERRNO	197
EXPERT	197
FCTEMPLATE	198
FCVERSION	198
FILEDIA	198
HIGHLIGHT	198
LANGUAGE	199

Table of Contents

LASTVAR	199
OFFSETDIST	199
PALETTE1 ... PALETTE _n	199
PANSCALE	200
PDMENUNAME	200
PERIMETER	200
PLATFORM	200
POLYSIDES	201
RINGDIA1	201
RINGDIA2	201
SCREENMODE	201
SCREENSIZE	202
SELECTBOX	202
SERNUMBER	202
SNAPBOX	203
TABMENUNAME	203
UNDOCTL	203
ZINSCALE	203
ZOUTSCALE	204
Local Variables: Reference	204
ANGBASE	204
ANGDIR	204
ATTDIA	204
ATTMODE	205
ATTREQ	205
AUNITS	206
AUPREC	206
CECOLOR	206
CELTYPE	206
CHAMFERA	207
CHAMFERB	207
CLAYER	207
CVPORT	207
DBMOD	208
DIM _{xxx}	208
DWGNAME	208
DWGPREFIX	208
DWGWRITE	208
ELEVATION	209
FILLETRAD	209
FILLMODE	209
GRIDMODE	209
GRIDUNIT	210
HPANG	210
HPDOUBLE	210
HPFILE	210
HPNAME	211

HPSCALE	211
HPSPACE	211
HPUSRANG	212
INSBASE	212
INSNAME	212
LASTPOINT	212
LIMMAX	213
LIMMIN	213
LSPALOAD	213
LTSCALE	213
LUNITS	214
LUPREC	214
MIRRTEXT	214
ORTHOMODE	215
OSMODE	215
PDMODE	216
PDSIZE	216
PLINEWID	216
PREVCMD	217
SNAPBASE	217
SNAPMODE	217
SNAPUNIT	218
SPLFRAME	218
TEXTSIZE	218
TEXTSTYLE	218
TRIMMODE	219
UCSFOLLOW	219
UCSNAME	219
UCSORG	220
UCSXDIR	220
UCSYDIR	220
USERI1 ... USERI5	220
USERR1 ... USERR5	221
USERS1 ... USERS5	221
VIEWCTR	221
VIEWDIR	221
VIEWSIZE	222
VIEWTWIST	222
WORLDUCS	222
Dimension Variables: Reference	223
DIMALT	223
DIMALTD	223
DIMALTF	223
DIMAPOST	223
DIMASO	223
DIMASZ	224
DIMBLK	224

Table of Contents

DIMBLK1	224
DIMBLK2	224
DIMCEN	224
DIMCLRD	224
DIMCLRE	224
DIMCLRT	224
DIMDLE	225
DIMDLI	225
DIMEXO	225
DIMEXE	225
DIMGAP	225
DIMLAYER	225
DIMLFAC	225
DIMLIM	225
DIMLINE	226
DIMPOST	226
DIMRND	226
DIMSAH	226
DIMSCALE	226
DIMSE1	226
DIMSE2	227
DIMSHO	227
DIMSOXD	227
DIMSTYLE	227
DIMTAD	227
DIMTFAC	227
DIMTIH	227
DIMTIX	228
DIMTP	228
DIMTM	228
DIMTMSTR	228
DIMTOFL	228
DIMTOH	228
DIMTPSTR	228
DIMTSTYLE	228
DIMTSZ	229
DIMTVP	229
DIMTXT	229
DIMZIN	229
Entity and Table Group Codes	231
Entity Group Codes	231
Common Entity Codes	231
3DFACE	232
ARC	232
ATTDEF	232
ATTRIB	233

CIRCLE	234
DIMENSION	234
INSERT	235
LINE	235
POINT	236
POLYLINE	236
SEQUEND	236
SOLID	237
TEXT	237
VERTEX	238
Table Group Codes	239
APPID	239
BLOCK	239
DIMSTYLE	240
LAYER	241
LTYPE	242
STYLE	242
UCS	243
VIEW	243
VPORT	243
APPENDIX B	244
Command line versions for Menus, Macros, Lisp-(command ...)'s and FDT-Functions	245

Table of Contents

Introduction

The *Programmer's Guide* contains detailed information on how to personalize the program and create a working environment which suits the individual requirements of the user. The techniques and procedures used to achieve this goal assume that the user has attained a moderate level of application knowledge and programming skills.

The configuration of the user interface, the allocation of **function keys** and **alias-commands**, and the usage of **template drawings** should not present a problem to any user, regardless of their programming experience. This also applies to functions in the **Dialog and Menu Editor**. This program module is particularly flexible and is an important instrument for customizing menus, tool boxes and dialogs. It not only makes the creation and alteration of menus and palettes very easy, but, as far as these functions are concerned, it alleviates the need for specialized programming skills.

LISP programming, however, requires a basic knowledge of appropriate programming techniques.

Manual Organization

Chapter 1

... provides a description of the program **Directory structure**. Understanding the directory design and path layout is helpful for many customization and programming items discussed in this manual. Also, there is a **File type overview**. At the end of the chapter the **File locking** and **Auto-save** features are described.

Chapter 2

... is aimed at those users who wish to personalize the program and its user interface in order to meet the specific requirements of their individual CAD construction and drawing environment by setting up the **Screen and desktop colors** and the **Command line font**.

Chapter 3

... provides hints and instructions on how to increase program efficiency by utilizing features such as **Hot keys** and by defining **Alias commands**. Another part of the chapter discusses the usage of **Macros**.

Chapter 4

... details the user configuration options available in the program when using **Template drawings**. These can be used to pre-define the appropriate drawing parameters, such as scaling, text styles, layers, line types, hatch patterns, views, user coordinate systems and other settings.

Chapter 5

... describes further options for a more specific drawing environment which can be achieved by defining and using individual tailor made **Line types** and **Hatch patterns**.

Chapter 6

... introduces the **Dialog and Menu Editor**.

Chapter 7

... describes the Dialog and Menu Editor to generate individual **Palettes** (tool boxes) or to customize existing palettes.

Chapter 8

... explains how to modify individual **Pull-down menus** by using the Dialog and Menu Editor.

Chapter 9

... is also dedicated to working with the Dialog and Menu Editor and describes the use of the editor when creating individual **Dialog boxes**. This part is easy to understand. However, linking the results into the program does require some minimum of programming knowledge and experience.

Chapter 10

... is aimed at the **LISP** programmer and describes the programming possibilities which are offered with the aid of LISP routines. The individual functions are not only listed according to subject and topic but are also shown in alphabetical order.

Chapter 11

... discusses the **LISP programming functions for dialog boxes control**: display dialog boxes, fill or evaluate dialog control elements.

Chapter 12

... outlines the **Global and local system variables** contained within the program and offers both the experienced user and LISP application programmer further options to configure and shape the program to their own requirements.

Appendix A

... contains a table of supported DXF Group Codes

Appendix B

... provides a list of commands designed for Menus and Macros

CHAPTER 1

Directory Structure and File Type Overview

File-Locking and Auto-Save

This chapter provides a description of the program directory structure. Understanding the directory design and path layout is helpful for many customization and programming items discussed in this manual. Also, there is a file type overview. At the end of the chapter the file-locking and auto-save features are described.

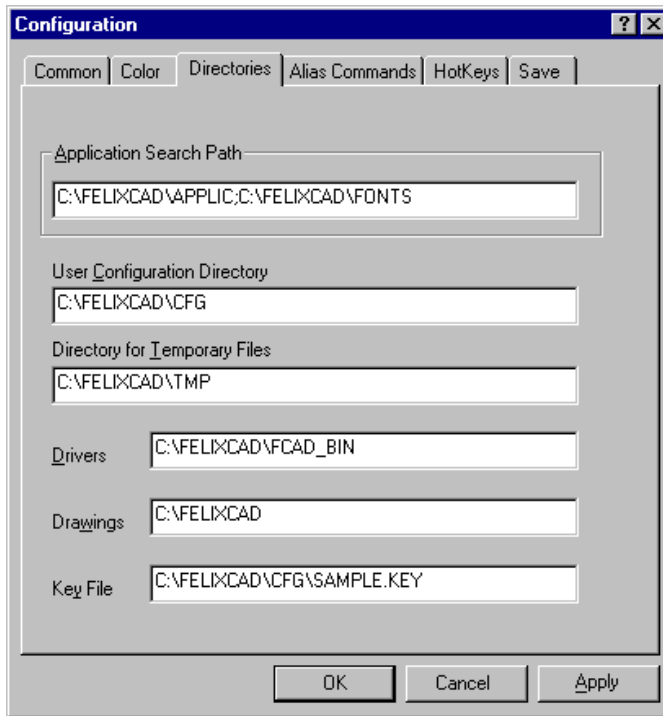
Directory Structure and Path Configuration

The *Directories* dialog tab which is part of the **CONFIG** command is used to set up a number of working directories which *FelixCAD* can access during the course of program operation. These various directories enable the program to find files containing information about menus, device drivers for input/output devices etc. which it needs to execute the program correctly.

The directories are created at the time of program installation and are initialized according to a pre-set pattern. By selective alteration of these directories, it is possible to customize the program and as such, mold it to your own personal requirements. User defined pull-down menus which are needed during program operation are for example stored in special directories.

The Directories Configuration Dialog

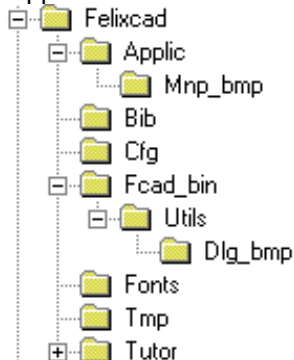
The **Directories** tab is part of the Configuration dialog box and is called by selecting **Configuration...** in the *File* pull-down menu or by typing the command **CONFIG** at the command line.



In order to enter or edit the directory settings, move the cursor to the appropriate input field and either type in the directory and exact path details, or change the existing entries to suit your own needs. Further information regarding the individual directories and the files contained therein can be found by referencing the appropriate subjects, for instance menu files, device drivers, etc.

Directory Structure and Paths

The directory structure is broken down into program, configuration, and support directories.



Application Search Path

The directory \APPLIC contains the support files required for help, pull-down menu, palettes, linetype and hatch patterns, as well as the FLISP example files. And, another sub-directory is located in the directory. It contains the bitmap files for palettes (\MNP_BMP).

System Directory

The directory \FCAD_BIN under the main directory are of no importance to the user for customization as they contain program command files and files of the graphics engine. The directory also houses the driver interface required if a digitizer is utilized.

User's Configuration Directory

The configuration directory (usually \CFG) contains the current drawing editor configuration details. In a network environment this directory should be private or local for an individual user.

File Types

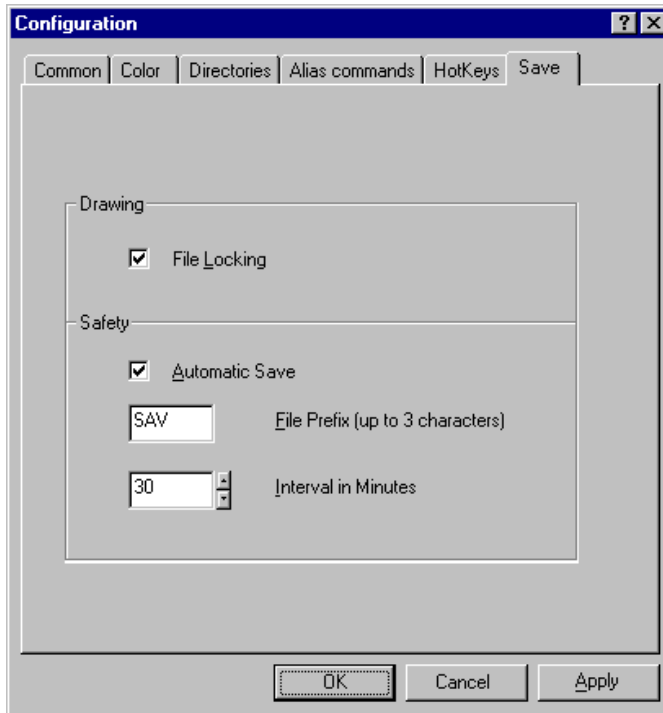
Overview

File Type	File Location	File Extension	Refer to Chapter
Auto-Saving, File Locking	\CFG	FLK*	1
Desktop Settings (Colors etc.)	\CFG	CFG*	2
Shortcuts, Alias Commands	\CFG	KEY	3
Macro Files	\APPLIC	MCR	3
Template Drawing	\APPLIC	FLX*	4
Text Fonts Binary (for drawing)	\FONTS	FSH*	4
Line Type Definitions	\APPLIC	LIN	5
Hatch Pattern Definitions	\APPLIC	PAT	5
Palettes	\APPLIC	MNP	6
Pull-down Menu	\APPLIC	MNU	7
Tablet Menu	\APPLIC	MNT	User's Manual
Dialog Description	\APPLIC	DLG	8, 10
Lisp Routines	\APPLIC	LSP	9, 10
Global system variables	Drawing	CFG*	11
Local system variables	Drawing	FLX*	11

- Note: CFG, FLX, FLK, and FSH files are binary files and cannot be edited with a text editor. A number of LSP files might be protected.

File Locking and Auto-Saving

The section **Save** tab of the Configuration dialog box allows you to activate or deactivate the functions governing the automatic backup of drawing files, as well as locking drawing files so that they cannot be used simultaneously.



File Locking

The *File Locking* option controls the creation of so called lock files. This function is activated or deactivated by clicking the check box. A check mark indicates that the function is active. Lock files are backup files which the program creates as soon as the drawing file is opened. They have the same file name as the original file, but are allocated with the **.flk** default file extension. Lock files are automatically removed as soon as the original drawing file is correctly closed. When the program is used in a net-work environment, this option prevents files from being used simultaneously by two or more users. **Note:** The same situation is encountered should the program come to an unexpected or abrupt end (for instance, due to a sudden power failure or program error), before the open drawing files could be properly closed. If you attempt to open a locked file, the program will display the locked file notification message.

Delete Locked Files

In order to retrieve your drawing you must first delete the corresponding lock file. This may be done by either selecting the function **Delete File...** which is located in the *File* menu, or by entering the command DELFLK. Further information can be found in the section entitled Deleting Files in chapter 1 of the *User's Guide*.

Auto-Save

When activated, the Automatic Save function will save all open drawing files to your disk at regular intervals. This ensures that your data is not lost in case of a system fault and also reduces the effects of follow-on defects.

The *Save* dialog tab enables you to activate or deactivate the automatic backup utility, specify the backup file prefix and enter the time intervals at which backups are to be made. *Automatic Save* is activated by clicking the appropriate check box. A check mark indicates that the function is active. The text box below the check box is used to enter a filename prefix for the backup files. The default setting is **SAV** for the file prefix. The intervals at which backup copies are to be made must be entered in the time edit field. You may type in any number of minutes, or use the directional arrows to select the desired time.

CHAPTER 2

Desktop Configuration: Colors and Command Line Font

This chapter is aimed at those users who wish to personalize the program and its user interface in order to meet the specific requirements of their individual CAD construction and drawing environment by setting up the screen and desktop colors and the command line font.

Desktop Colors

By choosing the **Color** dialog tab in the *Configuration* dialog box called by the command CONFIG, a sub-dialog will appear in which you may define the colors to be used in the following user interface elements:

- Text window background

- Text displayed in the *Command History / Lisp Interpreter* text window

- Drawing window

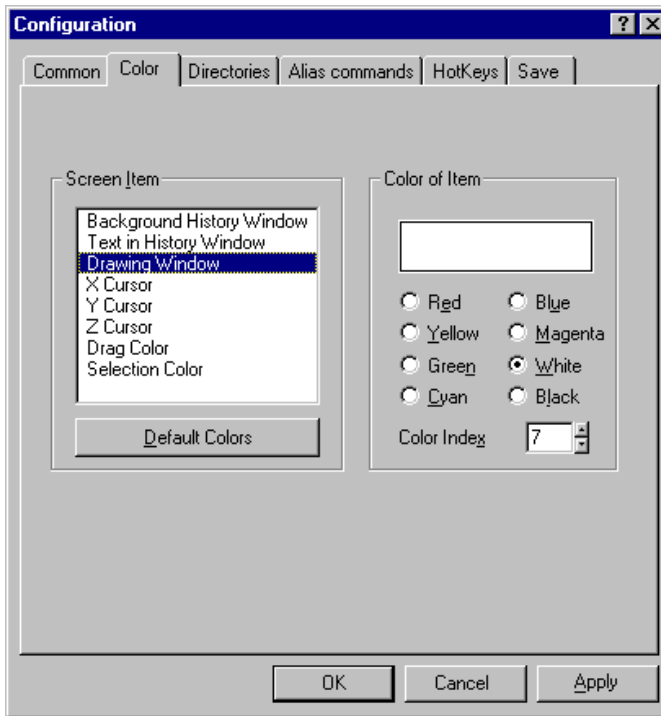
- Graphic cursor X,

- Graphic cursor Y,

- Graphic cursor Z

- Dynamic preview elements highlighting (Drag color)

- Object selection highlighting (Selection color)



First select the element which you wish to alter from the list box. The current color for the chosen element is shown in the area *Color of Item*. Please bear in mind that the cursor axis color setting is identical to that displayed in the coordinate axis symbol. The coordinate symbol color will only change shade after refreshing or regenerating the drawing view. The color values for the background and for the font of the text window titled *Command History / Lisp Interpreter* are shown based on the primary colors RED, GREEN and BLUE. The colors of the above mentioned windows can be altered by adjusting the appropriate RGB values. You may also use the **Color...** button in this area which is located in the control panel to allocate a new color. This will then cause the available Windows control panel options to be displayed. Further information regarding the selection and definition of these colors can be obtained from your Windows system documentation. The color selection for the remaining screen elements (Drawing window background, Crosshair colors, Drag and Selection Color) can be made from the list of standard drawing colors. It is also possible to define further colors by entering a corresponding color index (1...255). In order to return to the program default colors, select the **Default Colors** button shown in bottom right-hand corner. The changes will only become effective once the OK button has been selected. No changes will become effective if you close the dialog window by pressing either the Cancel button or ESC key.

Command Line Font

By choosing the **Text Font...** button on the main page of the *Configuration* dialog box called by the command CONFIG, a dialog will appear in which you may define the desktop font. The settings available in the Font dialog box affect the way in which information in the command line is displayed. This includes output in the command line, notification and warning displays and input fields. It does not have any influence on Text objects, such as titles which are entered using the text command and inserted into drawings.

Using the options available in the font dialog window it is possible to alter the display parameters utilized in the text window. The options include font type, font style and size. The desired selections can be made consecutively by clicking the appropriate selection fields. The example text displayed in the preview window shows what changes the selected parameters will make. These changes will only become effective once the OK button has been selected. No changes will become effective if you close the dialog window by either clicking the Cancel button or pressing the ESC key.

Command line parameters for FELIXCAD.EXE

You can add some parameters to the command line when starting FelixCAD.

-c

... specifies an application INI file (NOTE: **without extension**), e.g. "mycad". This tells FelixCAD to search for an MYCAD.INI file in the Windows directory. And, a MYCAD.CFG file is created in the configuration directory.

-m

... specifies an macro-script-filename (with or without extension; for example MYCAD.MCR). This can contain commands (which are useful when loading the program) or optionally can contain Lisp expressions, for example to load a Lisp file using a command like (load "mycad.lsp").

-d

... can specify a drawing to be opened when FCAD starts (filename with or without extension)

Example: `felixcad.exe -c mycad -m daelim.mcr`

Example: `felixcad.exe -d test`

CHAPTER 3

Command Customization: Alias Commands, Hotkeys and Macros

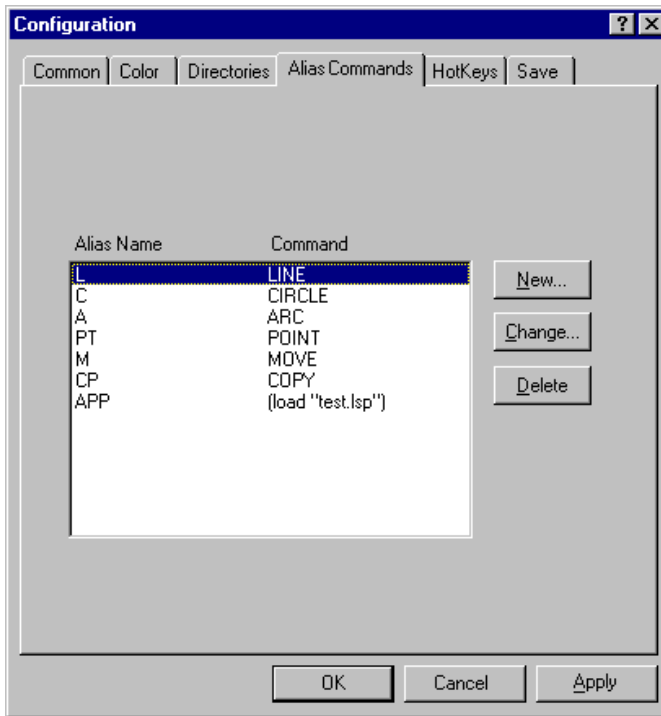
This chapter provides hints and instructions on how to increase program efficiency by utilizing features such as Hotkeys and by defining Alias-commands. Another part of the chapter discusses the usage of Macros.

Alias Commands

As is the case with function keys, alias commands also offer a very simple and effective way of customizing the program to meet the individual needs of the user. Alias commands can contain command entries, which are not only made up of a command itself, but also corresponding command arguments, such as parameters. Alias commands can be defined freely so that the command sequence

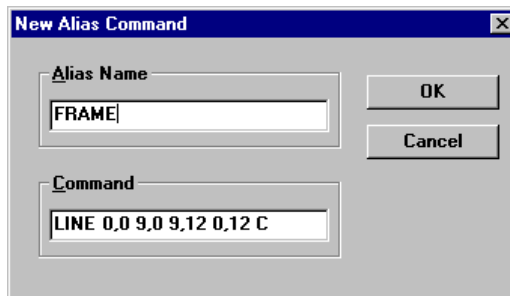
```
LINE 0,0 9,0 9,12 0,12 C
```

could be replaced with the alias command `FRAME`. Using this method it would no longer be necessary to input the parameters to draw a rectangle to represent an A4 sheet of paper. The same result could be achieved by simply typing `A4`. This method of operation not only increases the speed and efficiency with which the command is carried out, but also greatly reduces the risk of input errors. Alias commands can be defined by selecting the Alias Commands function from the *Configuration* dialog box. To do so, first choose **Configuration...** from the *File* menu or type in `CONFIG` at the command line, and then select the **Alias Commands** tab. Alias commands can then be defined as shown in the following description.



All previously defined alias commands are displayed in a list box.

- By selecting **New...** it is possible to define a new alias command. Enter the name and command line, including parameter arguments if necessary.



-
- In order to change an already defined alias-command select an item from the list box and click the **Change...** button. In the dialog box *Change Alias Command* enter a new alias name and the built-in command, including parameter arguments if necessary, into the input window.
- In order to **Delete** an alias command select an entry in the list box and click the delete button. Note: the alias command will be deleted immediately, you will not be requested to re-confirm the deletion.

The entries or changes only become effective once the OK button has been selected. No changes will be made if you close the dialog window by pressing either the Cancel button or ESC key.

HotKeys: Function Key Allocations

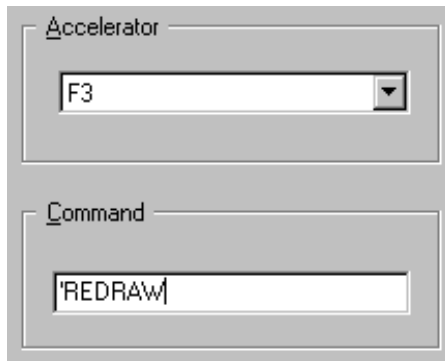
The function key allocation can be used as a very simple and flexible method of personalizing the program to suit individual requirements and work styles. Using either the individual F1 ... F12 function keys, or combining these with the CTRL and ALT keys, it is possible to automate up to 48 functions and commands which then become available at the “touch of a button”. Most of the 48 alternatives can be freely defined. Note that some function keys are reserved corresponding to Windows conventions and can not be altered:

The F1 key is allocated to the help function and reserved.

The CTRL+F4 key combination will close the active drawing.

The ALT+F4 key combination will close the program.

By allocating regularly used commands, such as drawing or editor instructions, object capture, or object selection functions to these keys it is not only possible to simplify your construction work, but also to dramatically increase speed and efficiency. From the *Hot Keys* dialog tab, select the function key, or key combination to which you want to allocate a command from the drop-down list. If this key has already been allocated, the corresponding command will appear in the command edit box. If the display remains empty then no command or has yet been allocated to this key combination.



By moving the cursor into the Input field and typing in your desired command or function it is possible to allocate a new command to the selected key. Should you wish to allocate a function or command to a key which already contains an entry, just overwrite the entry with the desired command. Please note that the parameter changes only become effective once the OK button of the *Configuration* dialog has been selected. No changes will be made if

you close the dialog by either using the Cancel button or pressing the ESC key.

Macros

Macros are designed to automate the execution of a number of specific instructions or commands one step at a time, thus eliminating the need to enter the commands and entries each time they are required.

Creating Macros

Macros can be written using any text editor or word processing program which is able to store pure text format (ANSI) i.e. without the any formatting. Enter the desired commands and instructions and their corresponding input values into this file in the same order in which the instructions are to be executed. A description of the individual commands and their necessary inputs can be found in the corresponding chapters of the *User's Guide*. Command execution takes place line by line or instruction by instruction when separated by a blank space. Both a carriage return or a space character are interpreted as confirmation of the previous entry. This means that either of the following macros could be used to draw a circle with an 0,0 mid point and a radius of 10.

```
CIRCLE  
0,0  
10
```

or

```
CIRCLE 0,0 10
```

Save the file in text format (ANSI) and use **.mcr** as the file extension name.

Using the clipboard

A simple and effective method of creating a macro file is made available by using the enlarged command line window titled *Command History / Lisp Interpreter* and the Windows clipboard utility. This text window contains a list of the commands and data entries which have been made at the command line. This enables the user to reference and reconstruct procedures carried out so far. By either using the scroll bar located on the right hand side of the text window, or alternatively pressing the upward / downward directional keys, the user can “scroll” through the listed entries. The Windows clipboard utility can then be used to copy entries into an editor program where they can be stored as a macro file. Use the following procedure:

1. Carry out the command(s) steps which you wish to store.
2. Open the text window and use the mouse cursor to mark the command sequence.
3. Copy the highlighted section onto the clipboard using the CTRL+C key combination.
4. Call up your editor program, for example Windows Notepad.
5. Insert the contents of the clipboard into the editor using CTRL+V.
6. Store the file in the support directory using any file name followed by the **.mcr** file extension.

Utilizing Macros

The macro utility can be called by entering the MACRO command. The program will then request the file name. It utilizes the pre-defined **.mcr** file if not specified otherwise.

CHAPTER 4

Template Drawings

When creating a new drawing you are given the option of using pre-defined templates. These templates are a form of specimen drawing, which either act as a basis for your finished work, or set up the required drawing environment. The main advantage of using templates is that all elements and settings of an already existing drawing can be adopted and as such do not need to be re-entered. This is particularly useful for parameters such as drawing boundaries, frames, guidelines, measurement settings, views etc. In order to facilitate this, a number of local variables containing stored settings are loaded along with the template. These settings are then used to display the drawing, drawing elements and objects. A table of available local variables can be found in chapter 7 of this manual.

Creating a Template Drawing

One standard template, saved under the name `template.flx` is included with the program and is stored in the drawing directory. It is also possible to define any existing drawing as a template. The method used to create a special template in which each individual working environment can be loaded, is no different to that used when creating a normal drawing:

- Create a new drawing, or open an existing one.
- Using the available options of the command `SETTINGS`, define the drawing, editing and display parameters. The dialog `Settings` allows the user to set all relevant drawing variables defaults like snap and grid spacing, zoom and pan factor, line type scale factor, point creation mode, etc.
- Set up the dimension parameters to tailor them to the need of the application with the `DIMTYPE` command.
- Provide a default setting for a hatch pattern file used with the `HATCH` command and set the parameters for pattern spacing and pattern angle.
- Create the layers for several purposes (`Construction Lines`, `Dimensioning`, `Cross-Hatching`, etc.) by using the `LAYER` command and assign to the individual layers the desired colors and the required line types. To load the needed line types from file uses the `LINETYPE` command.
- Create the drawing objects, e.g. the drawing boarder, or set up the other elements which you want to be available for use when starting a new drawing.
- Store the drawing as you would normally or by using the command `SAVEAS`.

This drawing can then be used as a starting point for further drawings which are to be based on the same parameters.

Loading a template drawing when creating a new drawing

In order to activate a new drawing using an existing template, mark the **Load Template** check box located in the file dialog **New**, and then click the button **Template...**. This will open a file dialog box entitled *Template Drawing*. Select the required drawing from the list and confirm your selection. The program will then return to the dialog box *New* where you should then enter the new drawing name. The template will be opened using the newly allocated filename. As a programmer you can pre-set a template drawing by setting the global system variable `FCTEMPLATE` with the `setvar` function.

CHAPTER 5

Defining and Using Line Types and Hatch Patterns

This chapter describes further options for a more specific drawing environment which can be achieved by defining and using individual tailor made **Line types** and **Hatch patterns**. Creating your line types and hatch patterns can be done with the aid of a text editor.

Line Types

Before a line type can be incorporated into a drawing it must have been defined in a line type file and loaded into the current drawing. The line type is loaded using the LINETYPE command. The files **inch.lin** and **mm.lin** already contain a number of predefined line types, which can also be used as a basis for other line types.

Hatch Patterns

The creation of hatch patterns is a little more complicated than the definition of a line type. Hatches are designed by repeating or duplicating of a number of lines. These lines can be rotated at any angle, start at various points and be spaced at different intervals.

Linetype Definition

Line type definitions are stored in ANSI format in a file using the **.lin** extension.

Line Type File Syntax

The file can be documented by beginning the file with a semicolon, followed by comment lines.

A line type format itself is made up of two lines.

- The first line, or header which starts with an asterisks, contains the name of the line type, followed by a comma and a symbolic representation of the line type.
- The second line, or definition line, contains a symbolic description of the line type. This description is made up of a string of dashes, spaces and dots.

Header: *Line type [, Line type description]

Definition line: Ls1, Ls2, ..., Lsn

Ls1,Ls2,...,Lsn: Line segment length 1, Line segment length 2, ..., Line segment length n

The line type description must not be more than 47 characters long, while the line segment length n must not contain more than 12 definitions with a total line length of less than 80 characters.

The following example shows how a dash-dot line type is defined:

```
*Dashdot, _ . _ . _ . _ . _ . _ . _ . _ . _ .  
A, 0.5, -0.25, 0, -0.25
```

The name of the line type to be defined (Dashdot) follows the asterisks (*). A comma then separates the symbolic representation of the line type ("_ . _ .").

The following line contains the geometric definition of the line type.

All definitions of dashes, dots, and spaces are separated by a comma:

- **Dash:** The drawing elements are shown as positive figures, for instance 0.5 represents a dash (PEN DOWN) with a length of 0.5 drawing units.
- **Space:** The negative units (-0.25) represent the space part in which no line should be drawn (PEN UP).
- **Dot:** Dots in the line type definition are shown as 0 values.

The length specifications represent drawing units when the line type scale factor (system variable LTSCALE) is set to 1.00.

Hatch Pattern Definition

The creation of hatch patterns is a little more complicated than the definition of a line type. Hatches are designed by repeating or duplicating of a number of lines. These lines can be rotated at any angle, start at various points and be spaced at different intervals. It is also possible to generate dash dot sequences instead of elongated dashes. FelixCAD contains a number of predefined hatch patterns which are stored in the **fcad.pat** file. In order to generate your own pattern file you must first of all use a text editor to create a file with a **.pat** extension name. Hatch patterns defined in pattern files can then be loaded using the HATCH command, and the desired hatch can be selected from the list box of the *Hatch* dialog. Once a new pattern file has been set up it will be used as the default file until another pattern is selected. The last selected pattern file is stored in the local system variable HPPFILE and will be stored along with the drawing.

Hatch Pattern File Syntax

Similar to line type definitions, hatch patterns are made up of a header, followed by the name of the hatch pattern. An asterisk (*) indicates the beginning of the hatch pattern name, this can be followed by a comma and the pattern description. The next line contains the actual hatch pattern definition. Every hatch pattern definition starts on a separate line and contains the following elements.

The angle of the line

The X and Y coordinates of where the line is to begin

X and Y spacing of the offset

Line segment length

The line segment lengths are only needed for elongated and / or dotted hatch lines and are not required for continuous lines.

As documentation it is possible to enter comments into the hatch file header.

These comment lines begin with a semicolon.

Header:

*Hatch pattern name [,Description of the pattern]

1st hatch line:

Angle, X starting point, Y start point, X spacing, Y spacing [,Ls1, Ls2,..., Lsn]

2nd hatch line:

-ditto-

n-th hatch line:

-ditto-

[Ls1,Ls2,...,Lsn]

Line segment length 1, Line segment length 2, ..., Line segment length n

The hatch pattern name must be less than 15 characters long, while the description can contain a maximum of 79 characters.

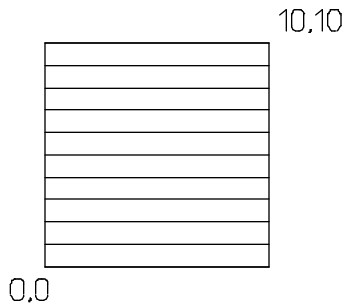
No more than 6 line segment length entries are permitted per hatch line definition.

Example 1: Horizontal hatching

The following example shows how to define a hatch pattern with the name HLINE using a line angle of 0 degrees. The start point is located at 0,0 and the line offset Y is 1 drawing unit wide.

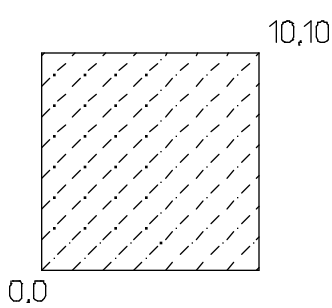
*HLINE, Horizontal line

0, 0,0 , 0, 1.0



Example 2: Dash Dot Hatching

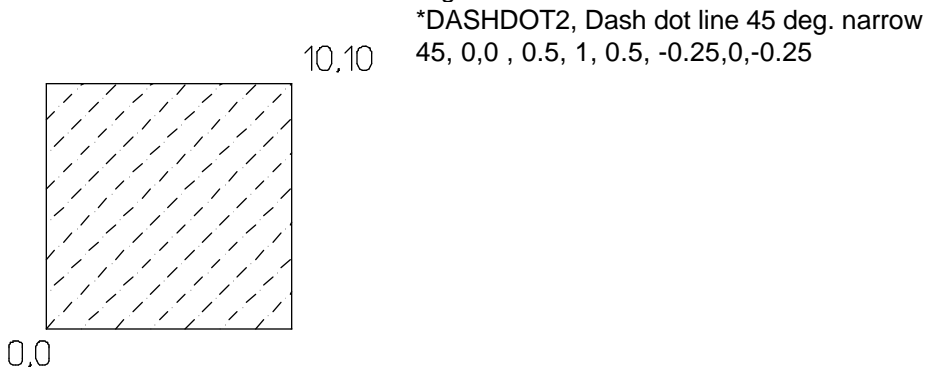
The next example is designed to show a dash dot hatch pattern drawn at an angle of 45°. The origin is located at point 0,0 whilst the hatch line offset is 1 drawing unit wide. The dash length is 0.5 drawing units. A space of 0.25 drawings elements is to be left between each dot and the following dash.



*DASHDOT, Dash dot line 45 degrees
45, 0,0 , 0,1,0.5,-0.25,0,-0.25

Example 3: Offset Dash Dot Hatching

The above example is now going to be changed to show a 0.5 drawing element offset. In order to achieve this, the line offset spacing x is increased to 0.5. All other values remain unchanged.

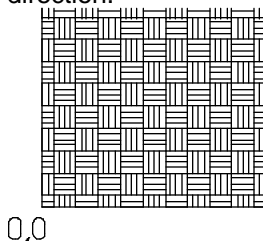


*DASHDOT2, Dash dot line 45 deg. narrow
45, 0,0 , 0.5, 1, 0.5, -0.25,0,-0.25

Example 4: Parquet Hatching

The next example defines a parquet hatch pattern which is made up of a number of hatch lines. The pattern should contain 4 parquets with a width of 0.25 drawing units.

This pattern is then used to create a pattern containing 5, 0 degree hatch lines and 5 hatch lines drawn at 90 degrees. The starting points are offset by 0.25 units in the Y direction at 0 degrees and by 90 degrees in the X direction.



* PARQUET, 1 x 1 parquet pattern
0, 0,0.00 ,1,1, 1.0,-1.0
0, 0,0.25 ,1,1, 1.0,-1.0
0, 0,0.50 ,1,1, 1.0,-1.0
0, 0,0.75 ,1,1, 1.0,-1.0
0, 0,1.00, 1.1, 1.0,-1.0
90, 0,0.00,0,1,1, -1.0,1.0

90, 0.25,0 ,1,1, -1.0,1.0

90, 0.50,0 ,1,1, -1.0,1.0

90, 0.75,0 ,1,1, -1.0,1.0

90, 1.00,0 ,1,1, -1.0,1.0

CHAPTER 6

The Dialog and Menu Editor

The next three chapters describe the options by which the Dialog and Menu Editor (DME) can be used to customize the program and to set up, create or modify palettes, pull-down menus and dialog boxes.

DME is an independent program delivered with FelixCAD. This module allows the user to create their own pull-down menus, palettes (tool boxes) and dialog boxes, or change existing ones.

No programming skills are assumed, apart from basic ones which are needed to integrate dialog boxes.

This means that all users have a simple but powerful method available to easily configure the program to meet their own particular needs. It is therefore possible to increase speed and efficiency by grouping regularly used, or additional functions into an individual menu or tool palette.

In addition, the Dialog and Menu Editor provides a powerful tool for rapid prototyping of applications.

LISP Programming

The use of LISP programming for user configuration is described in later chapters. As this method requires far more programming skills it is mainly designed for use by experienced operators or application developers.

File Types

Palette files are saved using the **.mnp** file extension.

Menu files are saved using the **.mnu** file extension.

Dialog files are saved using the **.dlg** file extension.

Using Menu and Palette Files

FelixCAD is supplied with one standard menu and a number of standard palettes. These standard elements enable the user to access every command and function and in general are adequate for complete and effective use of the program. Both the menu and the various palettes are stored in individual files. The file extension **.mnu** is used for all menu files, whilst the palette files use the **.mnp** extension name. The menus and palettes which are created or changed by the user with the aid of the Dialog and Menu Editor are also stored using these extension names. This process allows the user to create and choose menus and palettes independently and thus select those best suited to their particular work situation requirement. The program allows up to ten different palettes to be used simultaneously. These can be moved at random on the screen. Should you wish to open additional palettes you must first close the corresponding number of

displayed palettes. For obvious reasons only one pull-down menu file can be opened at any one time.

Loading Menus and Palettes

The MENU command is used to load Menu and Palette files. This function can either be called up by typing in the MENU command or by clicking the corresponding symbol displayed on the symbol bar. A standard *File Open* dialog box is displayed when the function has been called:

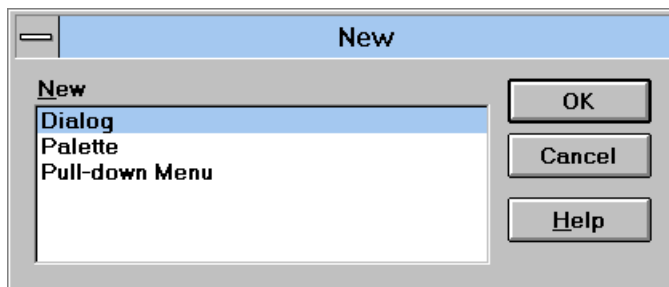
- First select the file type, depending on whether you wish to load a Pull-down menu, Palette or Tablet menu.
- Then choose the required file name from the list box and check it using the click function. The directory and drive selection fields enable you to move to the correct file location.
- Confirm your selection with OK, or ignore the selection of a menu or palette using the cancel button.

Palettes may also be activated by typing the PALETTE command, or by selecting the Palette symbol from the symbol bar. A pull-down menu may also be activated by typing the PULLDOWN command.

Creating and Saving Application Resource Files

Creating a New File

To create a new file using the Dialog and Menu Editor, select the **New...** function from the menu *File* and then choose one of the options from the list box of the dialog which appears:



Opening Existing Files

An existing palette, pull-down menu or dialog file can be opened by selecting the **Open...** function from the menu *File*. Using the options available in the file type list, find the **.mnu**, **.mnp**, or **.dlg** file format and select the file which you wish to open. Confirm your selection with OK. Before altering a standard file we suggest that you always make a backup copy.

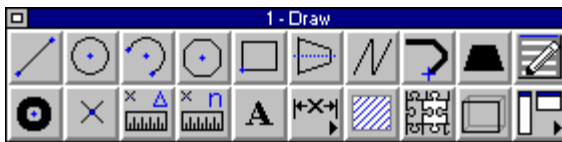
Closing Files / Saving Files

To close and / or save a menu, palette or dialog file use the **Close**, **Save** or **Save As** functions which are standard in all Windows programs. The **Close** function will shut the active file. Should changes have been made to the file, you will be asked if you wish to save the file before closing. The **Save** function does not close the current file, but will store it using the allocated name and file path parameters. If no file name yet exists then the Save as function will be carried out. The **Save As...** function allows the file to be saved under any valid drive and directory path. The file extension (**.mnu**, **.mnp**, or **.dlg**) is automatically added to the file name before saving takes place.

CHAPTER 7

Creating and Editing Palettes

Palettes are tool boxes which contain a number of buttons linked to various commands and instructions. These palette windows can be placed anywhere on the screen and act as an alternative method of calling up commands, in addition to those already offered by menus or direct input. Thanks to the direct command selection process, tool palettes offer a very effective method of dealing with frequently used commands or instructions. The options described below show how the Dialog and Menu Editor can be used to create new palettes, or amend existing ones and how they are implemented as a very efficient method of customizing not only the program itself, but also the drawing environment and working configuration.



The steps needed to create or amend a palette can be broken down as follows:

- Setting up of a palette
- Command allocation
- Definition of help functions and status bars

You will be guided through these steps with the aid of input and selection windows. If the selection of a specific file, e.g. a command file or a help file is asked for, this may be done by selecting the corresponding option button. This will open the correct file dialog box containing the standard options available for selecting the desired file format, drive and / or directory path.

Creating or Editing Palettes

Palette configuration consists of defining the number of buttons required, how they are to be arranged in rows and columns as well as their size. As a further step, the buttons are either allocated a text caption, or designated a graphic symbol (bitmap) which corresponds to the assigned command. The method needed to create a new palette, or amend an existing one is almost identical and is described below.

Select the New... option from the menu *File* or click the corresponding button. From the selection box which appears, choose the Palette option and confirm with OK. Type in the name to be allocated to the new palette. Please adhere to the standard operating system file conventions. Using the selection window allocate the drive and directory path to be used. Palettes are stored using the **.mnp** file extension identifier.

Existing Palettes

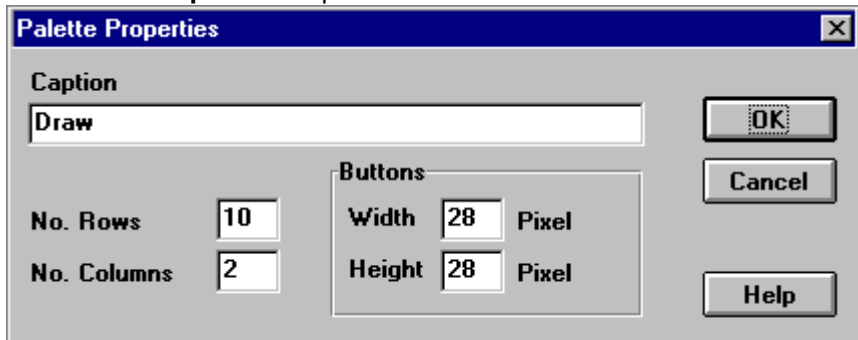
To open an existing palette, select **Open...** from the menu *File* or click the corresponding button. In the selection box set the file format to *Palette (*.mnp)* and then choose the required file.

Palette Properties

A new palette must first of all be configured. To do so, define the number of rows and columns needed. This process is called allocating the properties to a Palette.

As soon as you attempt to create a new palette, the *Palette Properties* dialog box will appear.

If you want to change the properties of a currently loaded palette please use the **Palette Properties...** option listed in the menu *Palette*.



In both cases a number of different entries need to be entered into a special dialog window. e.g. the Palette title or name, the number of rows and columns as well as the size of the individual buttons (in pixels). The number of rows and columns entered, will of course dictate the maximum number of buttons available in the Palette.

Allocating a Button Label

It is important to choose a button label which clearly describes the command or instruction which has been allocated to that particular button. This is imperative for efficient use of the palette. There are two methods of allocating a button label.

- A text entry, i.e. a description of the button (Text Button)
- A graphical representation or symbol (Bitmap Button)

For the functionality of the button, it is irrelevant whether a label is allocated before, or after the button has been assigned to a command. The allocation of a text label or graphical representation is made by selecting the corresponding radio button located in the *Palette Button Properties* dialog

box. The active choice is checked with a black dot.

Bitmap Button

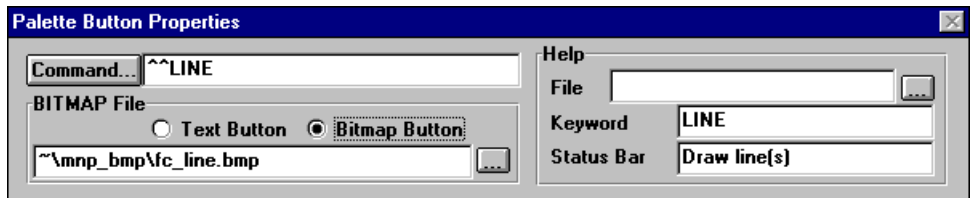
The allocation of a graphical representation (symbol or icon) assumes that a suitable bitmap file is available in the program, or that a suitable one has been created using a Windows paint application.

1. Click the **Bitmap Button** radio button.

Type in the bitmap file name including the (relative) path, or use the file selection dialog box, to help you locate and select the desired BMP file.

Important Note

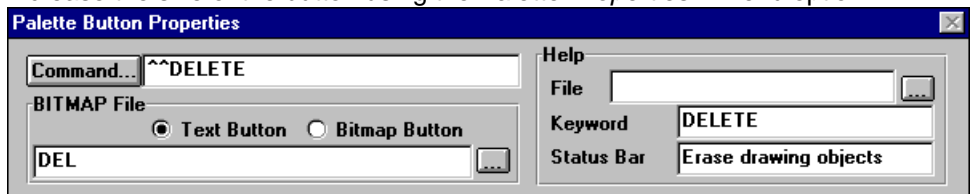
It is recommended to store the bitmaps used in a palette in the same directory or in a sub-directory relative to the directory where the palette file (mnp) is saved to. Use a tilde as placeholder for the path to the MNP-file when specifying the location of the BMP-file as shown in the illustration above.



Text Button

1. Click the radio button entitled **Text Button**, which is shown in the *Title* section of the dialog box.
2. Type the desired label of the text button into the text box.

The label should of course make clear reference to the instruction or command which will be carried out when the button is selected. In the following example the command DELETE has been labeled to DEL. Please note that the font size of the text label cannot be changed. Should the allocated label be larger than the button itself, then either shorten the label, or increase the size of the button using the *Palette Properties...* menu option.



Allocating Commands

After having allocated, changed or copied the palette properties, the next step is to link the button or switch to a command. To assist you with this task use the Palette preview and dialog selection window. Both windows are automatically displayed once you confirm the new palette properties, or choose to load an existing palette file. It is possible to open the button

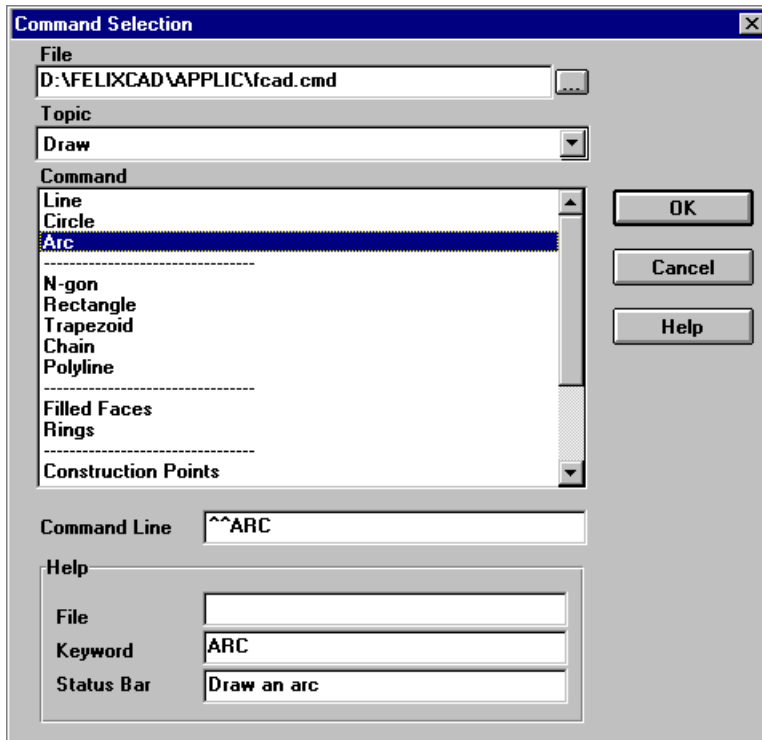
properties at any time by selecting the Button Properties option, located in the Palette, Pull-down menu.

Button Selection

The first step is to choose the button to which the command is allocated. Using the Palette preview window, click the button in question. The selected button is displayed as "depressed" whilst the remaining buttons remain unchanged.

Allocating or Selecting a Command

The allocation of a command to the selected button is made using the Command selection and input field. This can be found in the Palette, button Properties. The well versed and experienced user can type the command directly into the text box. The following explanation, however, describes in detail the method by which an individual command can be selected from the command file. Click the **Command...** button, which is located to the left of the text edit box. In the following window you will find a text box which is reserved for the command file. Please type in the name and path of the command file. Alternatively, click the button located to the right of the text box. This will open a file dialog from which you can select and load the prototype command file. The command file can easily be identified by the **.cmd** file extension identifier. This file is stored in the \APPLIC directory of FelixCAD. Once having loaded a template command file, the *Command Selection* dialog (see illustration below) can be used to select a required command from a template file.



As commands are usually grouped according to topic, use the *Topic* drop down list to choose the required topic, for example Draw or Edit. From the command list which appears, select the instruction which is to be linked to the chosen button. In the lower part of the dialog you find text edit boxes for the following parameters:

- the command line expression for the selected instruction
- the name of the help file containing the relevant information for the selected instruction.
- the key word used to search the help files
- the status line message entry.

When selecting an instruction from the command file you will notice that in most cases the above information will already have been entered by default. If this is the case you will be unable to amend the entries at this stage of the program. However if you use the instruction input field to type in the name of the instruction directly, you will not only find that you are able to branch to any help file, but also enter a self defined status bar message. Confirm the instruction selection process with OK, the selection window will close automatically. The name of the chosen instruction, the help file information and the status line message will be entered into the relevant fields of the input and Button properties selection window.

This concludes the procedure needed to link an instruction to a Palette button.

Help / Status Bar Entries

As a matter of default, the keyword and status line input field entries will be displayed as soon as an instruction or function is selected. Whereas you were not given the option to change the help file or status line entries in the *Command Selection* dialog, it is possible to amend them in the button properties selection and input window. The relevant dialog fields are located in the help section on the right hand side of the Button properties selection and input window. To select a different help file, or change the Help Keyword and Status Bar entries, place the cursor in the relevant text box and type over the existing entries.

Help

This text box **Help File** is used to define which help file contains the relevant information for the instruction or function be linked to the button. The help file will be opened as soon as the F1 help function is activated, or the corresponding symbol is selected from the function bar. Enter the name of the help file including the path details into the text box, or open the relevant file selection window and chose a file from the list. This entry **Keyword** determines which help file topic, relevant to the linked instruction, will be shown when the help function is activated.

Status Bar / Tooltip

By moving the cursor onto any palette symbol displayed on the program desktop, you will notice that a brief description of the symbol will appear in the status line. Using the Status Bar text box it is possible to stipulate which message will appear for each button of a palette. In order to change the status bar message, move the cursor to the text box and type over the current entry.

Saving and Closing Palettes

Repeat the steps as described until all of the desired buttons have been allocated and set up. Save your new or amended palette using the standard Windows options available in the menu *File*. In general it is a good idea to make regular copies during the palette creation and modification process.

Save

This function saves the active file under its current name and file path, but does not close the file. Select the **Save** command from the *File* menu, or click the corresponding symbol from the symbol bar.

Save As

This function allows you to save the menu file under a different name and / or path. The active file remains unaffected by this action. Select the **Save As...** option from the *File* menu. The standard file dialog will appear. Choose the desired path and select or type in the name under which you wish to save the file.

Save All

This option allows you to save a number of different menu, dialog, or palette files using only a single command. The files are stored using their current name and path parameters. Each individual save process is preceded by a confirmation request.

Close

Select the **Close** command from the *File* menu, or click the corresponding symbol from the tool bar. Before the menu file is closed, the system will ask you to confirm whether you wish to save the file or not. If you have made changes which you wish to keep, confirm this request by clicking OK. The file will then be stored and closed.

CHAPTER 8

Creating and Editing Menus

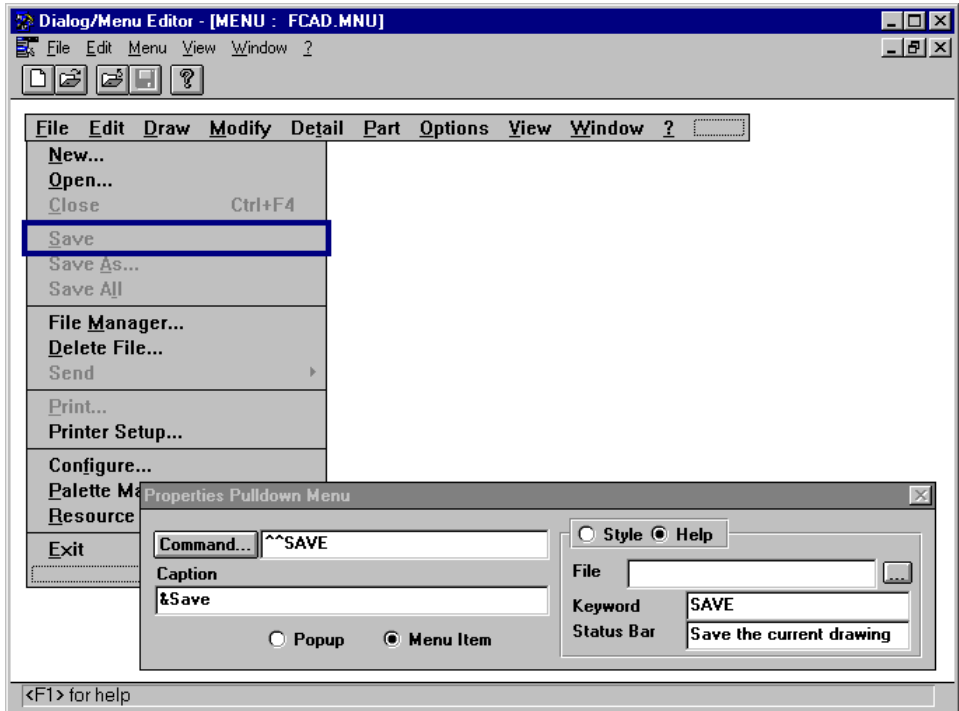
The Dialog and Menu Editor enables you to create or alter pull-down menus. These can be embedded into the program using the MENU command. This offers increased flexibility and can either allow additional commands and functions to be incorporated into the menus, enable menu commands and functions to be combined, or alter menus according to your own preferences.

Menu Editor

Pull-down menus can be created using the tools available in the graphic interface of the Dialog and Menu Editor (DME). This method is easy to learn and simple to use and combines the effectiveness and clarity of a graphic interface, a fact which also makes this method suitable for less experienced users. One main advantage of using this method is that the results can be seen immediately and are easily checked and corrected. Experienced users, especially those with programming skills, will be able to create new menus, or redesign existing ones by enlisting the help of a text editor.

Menu Files

Menu files are saved using the **.mnu** file extension. They may be opened, edited and saved using the Dialog and Menu Editor. Before altering the standard menu file we suggest that you always make a backup copy.



Creating New Menu Files / Opening Menu Files

To create a new menu file using the menu editor, select the **New...** function from the menu *File* and then choose the **Pull-down Menu** option from the list box which appears. An existing Menu file can be opened with the aid of the Menu editor by selecting the **Open...** from the menu *File*. Using the options available in the file selection window, find the **.mnu** file format and select the menu file which you wish to open. Confirm your selection with OK.

Closing Menu files / Saving

To close and / or save a menu file use the **Close**, **Save** or **Save as...** functions which are standard in all Windows programs. The **Close** function will shut the active menu file. Should changes have been made to the file, you will be asked if you wish to save the file before closing. The **Save** function does not close the file, but will store it using the allocated name and file path parameters. If no file name yet exists then the Save as function will be carried out. The **Save As...** function allows the file to be saved under any valid drive and directory path. The **.mnu** file extension is automatically added to the file name before saving takes place.

Editing Pull-down Menus

Once having created a new menu file, or opened an existing one, the following two windows become available to the user and allow the pull-down menus to be edited (illustrated below)

- An **Editing window** in which the menu structure can be configured or changed. This option allows existing Pull-down menus, menu points and sub menus to be added, changed or deleted as well as existing ones to be renamed.
- A **Properties window** in which the properties and link references of each menu option can be defined. The current settings of the active window option are displayed in the editing window.

When working with either of the Menu Editor windows please observe the following points.

Mouse and Keyboard Allocations

Mouse

Click a menu option with the left mouse button:	Select the information entries displayed in the "Properties Pull-down Menu"
CLICK + Drag	Move the menu option to a new location in the menu
CLICK + Drag + CTRL	Duplicate an entry

Keyboard

Arrow left	Move the menu option to the left
Arrow right	Move the menu option to the right
Arrow up	Move the menu option upwards
Arrow down	Move the menu option downwards
Pos1	Move the menu option to the top left position (first pull-down menu)
End	Move the menu option to the top right position (last pull-down menu)
Scroll up	Move the menu option to the top position in the active pull-down menu
Scroll down	Move the menu option to the last position in the active pull-down menu
INS	Inserts and selects an empty menu option following the selected option
DEL	Deletes a menu option from the menu
RETURN	Activates the Properties window of the selected option

Function keys

F1	Help
F2	Activate / Deactivate the symbol bar
F3	File: Save as ...
F4	Activate / deactivate the status line

Editing and Creating Menus

Editing a Menu Structure

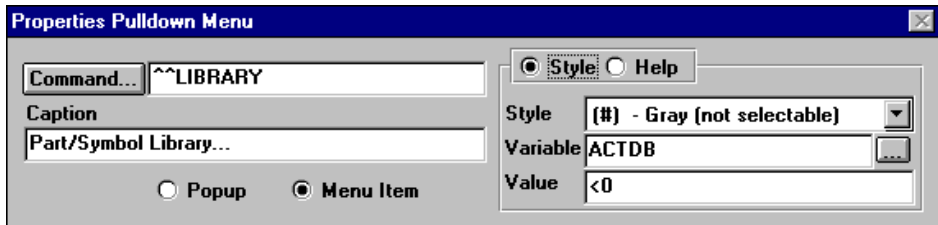
Select the menu option to be edited. If you want to extend the menu structure by inserting a new option, either move the cursor to one of the empty fields which automatically appear at the bottom and at the right hand side of the current menu level. The empty option boxes are displayed with a surrounding dotted line. Use the INS key to insert an additional menu field between the existing options of a pull-down menu. By actuating this key a new menu field will be inserted before the option which is currently activated. Apart from being able to insert additional menu options, it is also possible to define sub menus of a certain option. Selecting one of these menus during program operation will automatically cause the sub menus and additional menu options to be displayed and made available for selection. By combining commands into related groups, it is possible to structure the sub menus more clearly. To insert a sub-menu, first select the menu option from which you wish to start a sub menu and then choose the POPUP setting in place of MENUITEM from the Properties window. This will attach a sub menu to the active menu option. To show that a sub menu is available, a right directional arrow is displayed in the menu option. Additional menu fields can be added using the method described above. To insert a Separator between two menu items first insert an empty field. This field can later be defined as a separator line in the properties window.

Editing Properties

To edit the properties of each individual menu option, move the cursor to the properties window. This is most easily done by clicking the selection field corresponding to the required property. The individual menu items and their user definable properties are described in detail in the following section. Please note the following points. In most cases menu item properties can only be edited once the menu option has been given a valid name. Empty menu fields (pop-up menus) cannot be saved. If you wish to save a menu structure containing empty fields, these must have been given a title or name.

Menu Properties

The following menu entry or menu option properties can be defined or edited in the *Properties* window. In certain instances the *Properties* window could be closed, e.g. when editing a number of pull-down menus. Should this be the case it can be reopened by selecting the Properties option from the *Menu* pull-down.



Menu Entry Type

You may select one of the following menu option types.

MENUITEM

The MENUITEM option is the system default. Menu items are linked to commands and sometimes to help files, or to a status line message. When selected, the menu option in the user program carries out the command to which it has been linked.

POPUP

This POPUP type of menu is designed to call up sub menus. When selected these automatically open and display further menu options for selection. For this reason, pop up menu fields cannot be linked to, or defined as program commands. It is, however, possible to link them to help topics and to define status line messages to them. A control type should be defined by clicking the corresponding button located at the bottom edge of the properties window.

Caption

The name which is to be allocated and used in the Menu option is defined using the Caption command. Move the cursor by either using the tab key, or clicking the mouse to the Caption option. Then type in the desired name. Please take the following points into consideration.

Upper / Lower case

Upper and lower case characters do have an effect on the titling of the menu option. The title may contain umlauts, spaces, and special characters. Exceptions to this rule are the & symbol and \t .

Number of characters

The maximum number of characters which can be entered as a field name is only limited by the width or amount of space available in the pull-down menu. This is governed by the longest menu title and can be extended at will or whenever practical.

Highlighted Characters

The & symbol is used to highlight a letter in the menu item so that this item can be called up using the key combination, ALT + Highlighted key. The menu editor is only in a position to display these key combinations; their definition has to be programmed separately.

The & character can be located at any position in the character sequence.

The following entry **Precision &Aids** would produce this result:

Precision Aids.

Tab stop

The **\t** key combination causes any following characters to be displayed with a right hand justification or alignment. It can be used to greatly increase the legibility of a particular menu item. The menu editor is only in a position to display these key combinations; their definition has to be programmed separately. The **\t** character combination can also be combined with the **&**, so that the character sequence **&End\tAlt+F4** would display the following result :

Exit	Alt+F4
------	--------

Command

The definition of a command for the chosen menu option is carried out using the **Command** selection and input field located in the properties area. An experienced user would be able to type the specific command directly into the input field. The following section describes how to use the program driven support option to select instructions from the command file.

Command Selection

Click the **Command...** button which is located to the left of the input field. In the dialog box which appears, you will see an input field in which you can enter the name and path of the command file. Alternatively, you can click the button located to the right of the input field to open a file selection window. This window will enable you to select and load the command file **.cmd** which is stored in the **APPLIC** sub-directory. After having selected the command file, the following window appears. This window contains a list of the individual commands, and can be used to select the desired instruction. As the commands are listed according to topics, first select the topic which you need from the Theme selection field, for example **File**, or **Editing**. Click the directional arrow located by the side of the dialog box. You may then select the command which is to be linked to the menu option from the list. In the lower part of the selection window you will find fields in which to define the following parameters.

- The command used to select the chosen instruction;
- The name of the help file which contains information regarding the selected instruction.
- The key word for searching the help file
- The message which is shown in the status line

When selecting an instruction from the command file this information has usually already been entered by the manufacturer and cannot be changed or edited at this point of the program. If you, however, type the instruction directly into the instruction input field, then you will be able to refer directly to a help file, or type in the self defined message which is to appear in the status line. Confirm your selection with **OK**. This will close the selection window. The name of the selected instruction, the entries referring to the help file and the status line message will be stored in the corresponding input fields of the, Pull-down properties window.

Style Properties

A number of properties and attributes are grouped together under the style heading. These define the face and availability of each menu option, depending on particular program conditions. In accordance with definable program variables it is possible to determine that a particular menu option only becomes active when for instance a drawing is opened or created and that the drawing contains at least one drawing element. This type of configuration does assume a certain knowledge of the available program variables. Relevant information can be found in later chapters of this manual. The next sections are merely designed to show how, with the help of the Dialog and Menu editor, it is possible to select the style, the program variables and comparative values. The selection and input fields located in the right hand part of the properties window are used to define both the style and help properties. To switch between the two editing possibilities, use the style and help radio buttons shown at the top of the input fields. First of all check whether the **Style** button is active (default position). If not, click the button to change the selected option.

By doing so, the selection and input fields shown below enable you to alter the style, variable and value parameters.

Style

The style selection window allows you to change the style using the following parameters.

Standard

This style is used for menu options which can be activated at any time and are not intended to be especially marked. It is therefore **not** linked to program variables. The standard style is the default option.

Gray (Not selectable)

Allocate this style to the current menu option, if it is to be temporarily deactivated, according to the result of a comparison between a comparative value and a definable program variable. Once this style has been chosen, it becomes possible to open the previously locked, selection window variable and value parameters (see below).

Separator

By allocating this style the current menu option is no longer displayed as a caption, but is shown as a horizontal line. Any previous caption entry will be overwritten and deleted by the separator line. The height of the caption will also be reduced accordingly. Separators are seen as visible dividing elements and as such help to improve the clarity of the pull-down menu.

Checked

If you want to temporarily mark the current menu option with a check symbol, in accordance with the result of comparing the input value with a pre-defined program variable, then you may allocate this style to the menu option. This

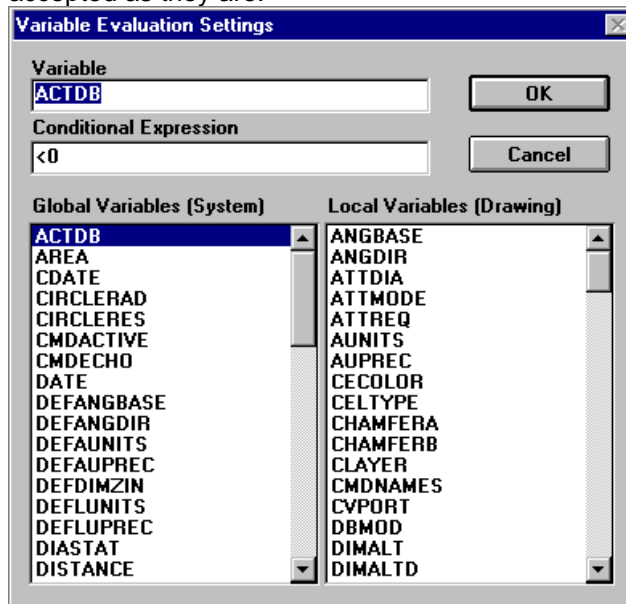
could for example be used to indicate the active or inactive state of a particular function. Once the checked style has been allocated you are able to open the previously locked selection window variable and value parameters.

Variable / Value

These selection or input fields are used to store the program variables and comparative values which when compared are used to determine the availability or checking of a selected menu option. In order to be able to access these selection or input fields, they must have been allocated either of the following style definitions.

- Gray (Not selectable)
- Checked

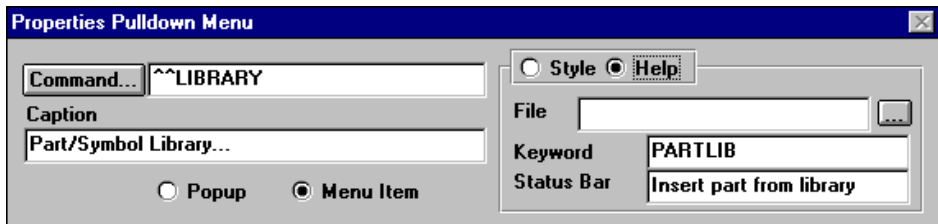
During the variable selection process you will be aided by a selection window, in which all of the possible variables are clearly listed. The *Variable Evaluation Setting* dialog box (see illustration below) can be opened by clicking the button located to the right of the variable text box. Select one of the global or local variables and enter the desired comparison expression into the corresponding text box. Confirm the selection with OK. The variables and their corresponding comparative values will then be automatically taken over and stored in the properties window. Here they can then be edited or accepted as they are.



Help Parameters

The definition and editing of help properties is carried out using the selection and input fields located on the right hand side of the properties window. The

same option is used to define the style properties. By selecting either the style, or help radio button it is possible to switch between the two editing modes. The style mode is the system default setting.



When activated, the Help file keyword and Status bar field can be edited. In most instances the keyword and status bar field will already contain default entries. Whereas in the *Command Selection* dialog box no opportunity is given to alter the entries shown in the help file and status bar, it is possible to make these changes in the *Properties* window. In order to select a different help file, or change the entries in the key word and status bar, simply type over the existing entries.

Help File

This input field is used to define which help file information is to be displayed. The choice depends on the command or function which is linked to the menu option. The selected file will be set as the standard and shown when F1, or the help symbol from the function bar is called up. Either enter the name of the help file (including the path) into the input field, or open the corresponding file selection window and select the required help file.

Keyword

The command linked to this entry field, dictates which help file topic will be displayed when the help function is selected.

Status Bar

Moving the cursor onto any symbol located in the user interface will cause a brief description of that symbol to be displayed in the status bar. Pull-down menu, status bar descriptions can be defined in the **Status Bar** text box. Generally speaking, the default entry which is displayed at the time of selecting the command or function will be kept as the standard. To amend the status bar message simply type over the existing message.

Saving and Closing Menu Files

Repeat the steps just described until all of the desired menu options have been set up and defined. Then save the new or amended menu file using the standard options available in all Windows applications.

Save

This function saves the active file under its current name and file path, but does not close the file. Select the **Save** command from the menu *File*, or click the corresponding symbol from the symbol bar.

Save As

This function allows you to save the menu file under a different name and / or path. The active file remains unaffected by this action. Select the **Save As...** option from the *File* pull-down menu. The standard file selection dialog box will appear. Choose the desired path (drive, directory and sub-directory) and select or type in the name under which you wish to save the file. Confirm your selection with OK.

Save All

The option **Save All** allows you to save a number of different menu, palette, or dialog files using a single command. The files are stored under their current name and path parameters. Each individual save process is preceded by a confirmation request. In general, it is advisable to use the dialog and menu editor to store your files regularly during each work sitting.

Close

Select the **Close** command from the menu *File* or click the corresponding symbol from the symbol bar. Before the menu file is closed, the system will ask you to confirm whether you wish to save the file or not. If you have made amendments and want them to take effect, then confirm this request by clicking OK or pressing the Return key. The file will then be stored and closed.

CHAPTER 9

Creating and Editing Dialog Boxes

The Menu and Dialog Editor (DME) enables the user to create and edit dialog boxes. These can be called up at pre-defined times or during specific actions. Dialog boxes can contain up to 256 control elements which direct certain specific actions. Some of these actions are described below.

- Input and Output of information
- Activation and deactivation of options
- Set or amend parameters
- Selection of elements (files, variables, etc.)

Dialog files are integrated into the application with the aid of program interfaces. The following chapter concentrates on the creation and editing of dialog files and windows. The methods used for working with the various program interfaces are summarized in the relevant chapters of this manual.

Experienced users, especially those with programming skills, can create, open or edit dialog files using a text editor. To help the user with this, a complete list of dialog elements and their correct syntax are shown at the end of this chapter.

Alternatively, dialog windows can also be created using the graphic interface tools available in the Dialog and Menu Editor. This method is easily understood by less experienced users and offers the added advantage of allowing the results to be viewed immediately. This in turn reduces errors and increases overall efficiency.

Dialog File Type

Dialog files are stored using the **.dlg** file extension. These files can contain numerous Dialogs, all of which can be displayed in the application when specific criterion are called up or met.

Creating New Dialog Files / Opening Dialog Files

By selecting **New...** from the menu *File*, the user can use the dialog editor to create a new dialog box. Follow this step up by selecting the option **Dialog** from the selection list. To open an existing dialog file, select the **Open...** command from the menu *File*, choose the designated file format (.dlg), file path (drive/directory), and file name.

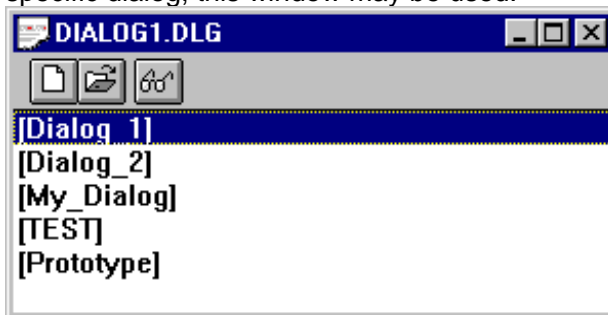
Saving and Closing Dialog Files

To close or store a dialog file the user may use any one of the following standard Windows functions.

- The **Save** function will store the file under the current name and file path, but will not close it. If a file name has not yet been allocated then the Save As function will be initiated.
- The **Save As...** function enables the user to store a file under any chosen file path (drive/directory) and name. The **.dlg** file format will be added automatically.
- The **Close** function shuts the active dialog file. If changes have been made to the file the user will be asked to confirm these changes before the dialog is closed.

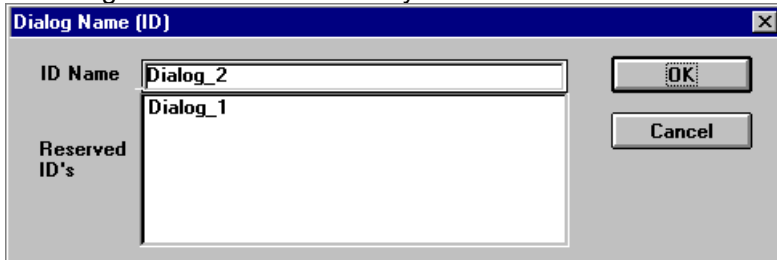
Editing Dialog Files and Dialog Boxes

When either creating a new dialog window, or naming or re-naming the specific dialog, this window may be used.



- If the user is creating a new dialog file, no dialogs will yet exist. Only the **New Dialog...** function (shown in the left hand side of the symbol bar) will be active. To create a new dialog in the form of a window click the relevant symbol.
- With existing files, a list of the available dialogs will be shown in the selection area of the window. Check the file which is to be opened and confirm the user selection with **Open Dialog...**. The user can, of course, add a new dialog at this point of the program.

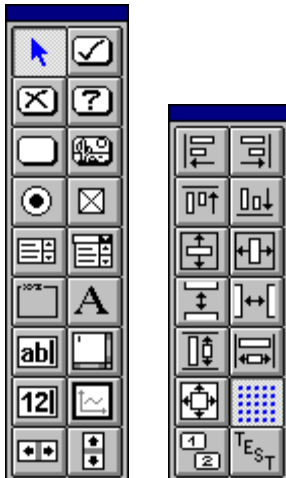
- By clicking this symbol an additional dialog window will be opened (see following illustration). This window allows the user to allocate individual dialog names or change existing ones. Check one of the dialogs and click the shown symbol.



Enter the designated name into the *ID Name* text edit box. Any illegal names will be displayed in the *Reserved ID's* list, as these names have already been allocated and cannot be reused.

The Dialog Editing Window

The application will automatically open a dialog editing window each time a dialog is selected. As with all Windows interfaces, the window can be maximized to fill the screen or minimized down to a symbol. The dialog editing window is used to create and design the dialog window. At this stage the user may take advantage of the typical commands contained in the File, (e.g. Open, Save, Close) Edit, (Cut, Copy, Paste, Delete) and Window, Pull-down menus. Specialized commands which may be used to edit the window or insert and format control elements can be found in the other Pull-down menus. A large number of the commands can also be activated using the tools or formatting Palettes. The individual commands and functions used during the configuration of the dialog window are described according to their use in the design process.



Mouse and Keyboard Allocations

Work in the dialog editing window can be made more effective by adhering to the following mouse control and function keys instructions.

Clicking the left mouse button ...

... will:

... on a control element ...

CLICK	Select a control element, de-select all previous control elements.
DOUBLE-CLICK	Open the Properties window of the corresponding control element.
SHIFT + CLICK	De-select the control element, if it was previously selected. The control element will be added to those already selected. (Multi-selection). Note: The element orientation is based on position of the last selected element.
CLICK + DRAG	The control element will be relocated within the dialog. By keeping the STRG key pressed at the time of releasing the mouse key, the user can duplicate the control element or elements.

... onto the border of a control ...

CLICK + DRAG	Changes the size of the control element.
--------------	--

... on any dialog or empty window (except CAPTION) ...

CLICK + DRAG	Will select all of the control elements contained within the frame at the time of releasing the mouse key.
--------------	--

Keyboard

F1	Help
F2	Show symbol bar on/off
F3	File: Save As
F4	Show status bar on/off
F7	Shows properties window of the selected control element
F8	Tool Palette on/off
F9	Alignment Palette on/off
F10	Grid on/off
DEL	Deletes the control element from the dialog

TAB	Selects the next control element from the list and de-selects the current one.
TAB + SHIFT	Selects the preceding control element from the list and de-selects the current one.

Size and Coordinates

All data regarding the size and position of dialog fields is given in dialog units. This is an internal measurement used by the Menu and Dialog Editor.

Grid Alignment

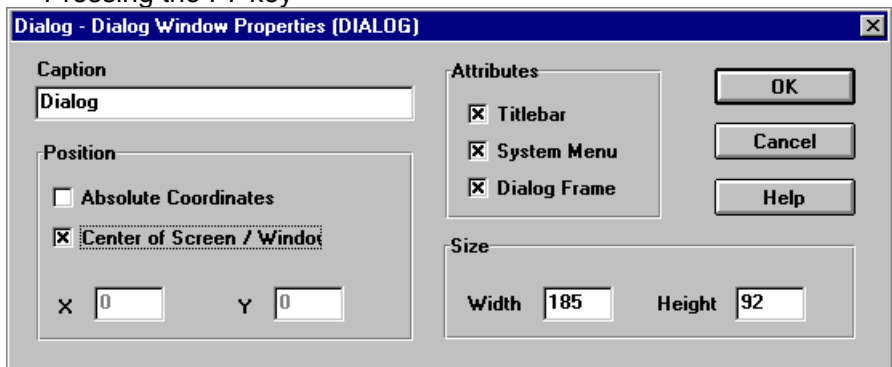
An alignment grid can be revealed in the Dialog window to help the user position or align a control element within that window. The grid line can either be activated in the menu *View*, by pressing the F10 function key, or with the grid button found in the Alignment Palette. Further information about the grid and snap-to functions can be found later in this chapter among the descriptions relating to the positioning and alignment of control elements.

Dialog Window Properties

The size, position and use of the dialog window which is to be created or altered is determined in the Window properties dialog.

This dialog window can be opened by:

- Double clicking the window surface, without touching any of the elements contained within the window
- Selecting the Properties function from the Element Pull-down menu
- Pressing the F7 key



Text and numeric values can be allocated to the following properties by using the alpha-numeric input fields and control buttons which govern the activation and deactivation of further options. Switching is possible by clicking the specific control field. A check mark in the control field indicates that the option is active, a blank signifies that it is deactivated.

Caption

Type in the desired heading or title which is to appear at the top of the dialog window. This text will only be shown once the Title bar attribute function has been activated (see: Attributes).

Position

Absolute Coordinates

This control button is used to enter the positioning and alignment reference values. If the *Absolute Coordinates* option is activated, then all further definitions will relate to positions within the full screen area. In this case the (X,Y) window coordinates will originate in the top, left hand corner of the screen. Any centered window alignments will be made according to the size of the full screen. If the *Absolute Coordinates* option is deactivated, then centering will be done with reference to the position of a major window. In this case the window positioning coordinates-ordinates will be ignored.

Center of Screen / Window

This option will ensure that the dialog window is placed in the center of the display area, both horizontally and vertically. This reference point will alter depending on whether the Absolute coordinate-ordinate option is calling for the full screen or major window.

X/Y Coordinates

The coordinates, which are measured in dialog units, define the horizontal and vertical origin of the top left hand corner of the dialog window. As above, this is directly related to the absolute coordinate option setting and will either use the top left hand corner of the display area, or the major window as a reference point.

Size

The default values for the horizontal and vertical size of a new dialog window are 185, 92. These can, of course, be overwritten with the user's personal preferences.

Attributes

Title Bar / System Menu / Dialog Frame

This will determine whether the dialog window contains a title bar or not. This option must be active if the user wants the window caption and system menu button to be visible.

The Controls

The following section contains instructions on how to insert a control element and define its properties. General, commonly used control element attributes are covered. This is followed by an outline of the different types of control element and a detailed review of the specific properties of each element. There is also a syntax section. This is designed for experienced users who prefer to edit the menu files using a

text editor. The options available to position and align the control elements within a dialog window are covered in a later section.

Insertion of a Control Element

The insertion and positioning of control elements is carried out in two steps:

1. Selection of the control element type
2. Positioning the control element within the dialog window

Control Element Selection

Two options are available by which a control element type can be selected.

- Select the *Controls* option from the menu bar. This will cause a pull-down menu containing the various element types to appear. An arrow behind the element type indicates that a further sub menu is available in which other type specifications can be made.

Please note that this pull-down menu will only be displayed if a dialog window has already been opened.

Select the required element type with the mouse pointer or cursor direction keys

- The Menu and Dialog Editor contains the tool Palette as illustrated above. This Palette is used to select the control element type and is opened by selecting the *View* menu option or by pressing the F8 function key. Using the mouse, the Palette can be moved to any position on the screen.

By moving the cursor onto the tool Palette elements, the user will cause a brief description of each button to appear in the status line.

To select a specific control element type just click the corresponding button.

Placing a Control

Once a control element has been selected, the cursor display will change from an arrow to a cross and a symbolic representation of the control element will appear. Move the cursor to the position at which the user wishes to insert the element and click the left mouse button. The control element will then be positioned using the cursor cross as a reference point.

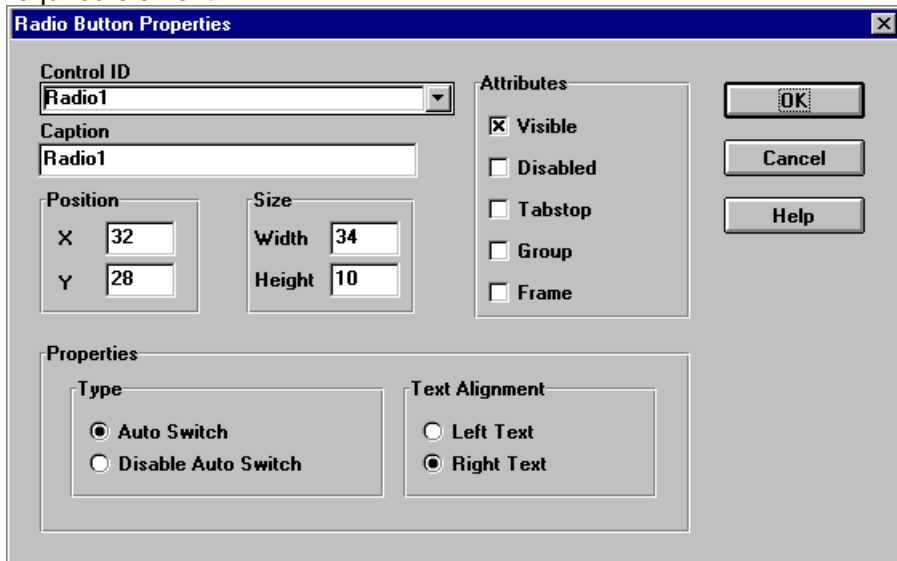
Note: The program offers a number of versatile functions which enable the user to place the elements directly below one another, or at a particular reference position within the window. These functions are introduced and explained in a later section.

Properties / Attributes

The definition of control element properties and attributes is carried in the Properties window using the same methods as described for dialog windows.

It is essential that an element has been placed within the dialog window before the editing window can be opened. To mark or select the element, click it with the mouse pointer. Further notes on how to mark an element can be found in the *Editing Window* section earlier on in this chapter.

Either open the properties window using the properties option in the Elements, Pull-down menu, press the F7 function key, or double click the required element.



A number of definable properties and attributes can be found in almost all control elements. They will be summarized together, and exceptions to these properties and attributes will be covered later.

Control ID

The Control ID (also called Control Name or Control Key) is a unique identifier given to a control element. This name is used as a reference between the program code and the control element. Accessing the dialog element by its name will cause the appropriate event routines to be called up, inputs to be read, or specific entries to be output.

Note:

- A control name may consist of up to 32 characters.
- A distinction is made between upper and lower case.

Control names which have been used previously in the same dialog will be listed in the input and selection window and cannot be reused.

Caption

The caption entry is used to allocate a heading or title. This will be displayed in, or on the control element. Bitmap buttons (see below) contain a graphical representation instead of a text header.

Position / Size

These entries, measured in dialog units, define the X and Y coordinates-ordinates of the top left hand corner of the control element as well as its height and width.

General Attributes

Visible

This determines whether a control element should be visible or hidden as a default. Dependent on certain actions (calling up of specific commands, definition of variables, clicking certain buttons) this attribute could change status during program execution. The element will then change from visible to invisible or vice versa

Unavailable

Sets a default as to whether the element can be accessed, or if it is to remain inactive. Inactive elements cannot be selected and are displayed accordingly.

Dependent on certain actions (calling up of specific commands, definition of variables, clicking certain buttons), this attribute could change status during program execution. The element will then change from available to unavailable or vice versa.

For example, the editing functions of a dialog box could remain unavailable as long as the corresponding drawing does not contain at least one element.

Tab

Specifies whether the control element can be accessed with the TAB key. In most control elements this attribute is set to active. This does not, however, apply to Radio buttons, static elements or the scroll bar. All elements can be selected using the mouse pointer, or as part of a group with the cursor direction keys.

The sequence in which the elements are selected, or activated using the TAB key is usually directly related to the order in which the elements were inserted into the dialog box. This order can be amended if required.

The element sequence option can be found in the menu *Dialog*, or in the Alignment Palette. The set sequence can be altered by changing the allocated reference number.

Group

This attribute allows the user to group together a number of separate elements. The first element in the group must be activated with the group attribute. The group will include all elements up until the point at which the next group attribute is activated. The sequence can be defined by using the method described above.

The individual elements contained within a group can be selected using the cursor direction keys. The next group, however, can only be accessed with the TAB key, or mouse pointer.

Only one button within a group can be activated at any one time.

Frame

The frame attribute, when selected, will draw a frame around the control element. Some elements (e.g. the OK standard action button, HELP, CANCEL) have a default frame which cannot be deactivated.

Text Attributes

These attributes are used in

- Control elements, displaying a variable text caption.

The text can either be aligned to the left or to the right of the control element symbol.

- Control elements used to enter or output text.

In this case, both the text alignment and character case can be altered.

Property: Pre-defined Button

Some control element types can be set to reflect a redefined option. This has the effect of setting the control element to active, as soon as the dialog window is opened.

In dialog windows which contain default or standard values it is usual that the OK button is set up as a pre-defined button. In this case the user will only need to press the Return key to confirm the selections.

This property can only be allocated once within a single dialog window.

Control Types

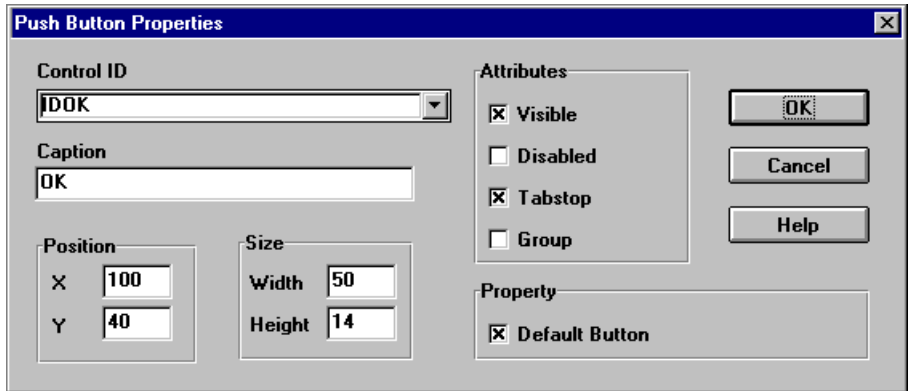
Standard Action Buttons: OK, Cancel, Help

Action buttons are control elements which instigate a specific action, in other words, carry out a standard command or function.

These buttons action a pre-defined function and can be inserted into a dialog window without the need for any further action.

Standard action buttons are:

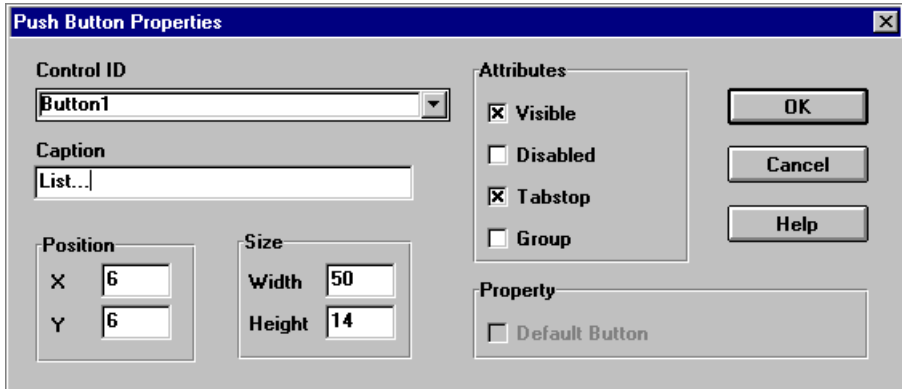
- OK,
- Cancel, and
- Help.



Command Button (Push Button)

As opposed to the standard buttons, these buttons can carry a title made up of free flow text. This should mirror the command or function which will be carried out when the button is actuated.

Type in the text into the Caption text box, but remember that the button size will not automatically increase to accept the user text. The title will automatically be centered within the button's size.

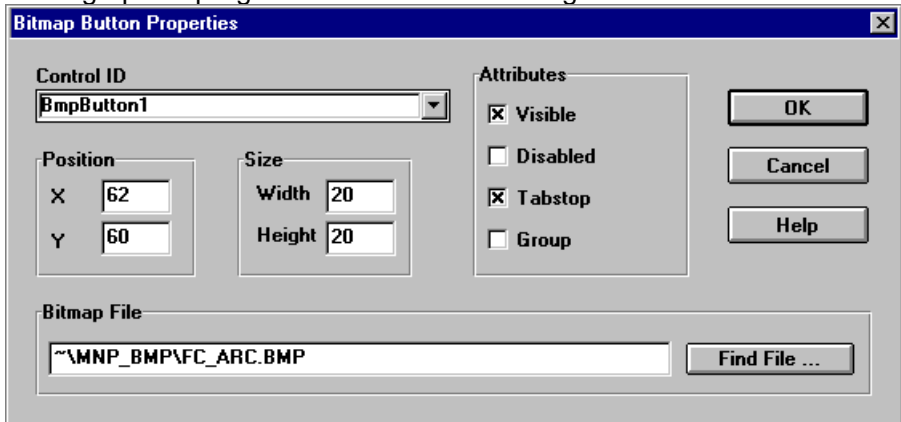


These action buttons can be allocated a control name within the program. This name can then be used to call up any command to which the buttons have been cross referenced or linked. The user-definable keys may be allocated a text title, or a graphical representation (icon).

Bitmap Button

This type of button depicts either a **.bmp** or a **.dip** file icon designed to mirror the functionality of the button.

Use the file selection window to select a suitable bitmap file. A number of such files are contained within the application. If required the user can use a graphics program or Icon editor to design the user own files.

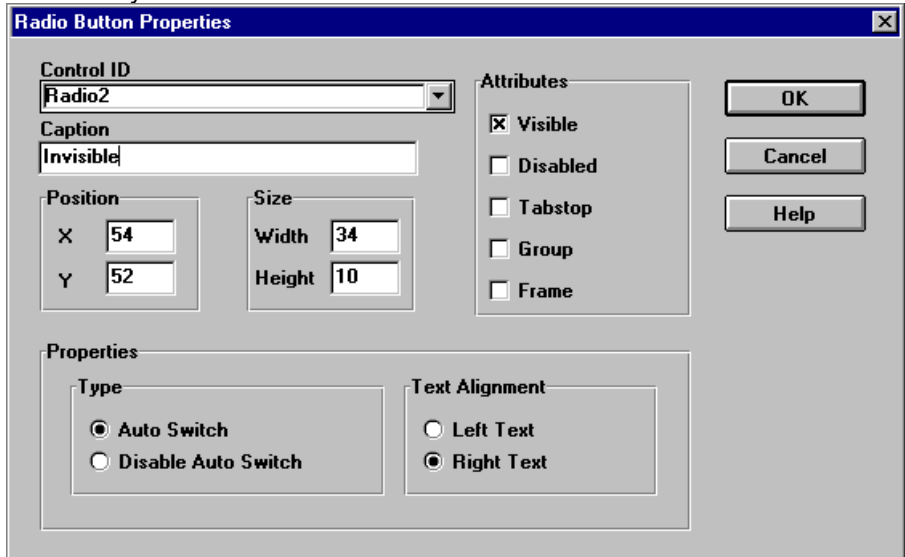


These action buttons can be allocated a control name within the program. This name can then be used to call up any command to which the buttons have been cross referenced or linked. The user-definable keys may be allocated a text title, or a graphical representation (icon).

Radio Button

Radio Buttons serve to switch between a number of pre-defined options, but only allow one of the available possibilities to be selected.

When a number of button fields are used, these are usually grouped together. Selecting an option from one group will cause the option which is currently active to be switched off.



In addition to the standard properties and attributes, the following settings may also be made:

Type

Select a button type from the following options:

Auto Switch causes the button to switch between the possible selection modes. At the same time each switch within a specific group will automatically change mode. Selecting one button will automatically de-select all others in the same group.

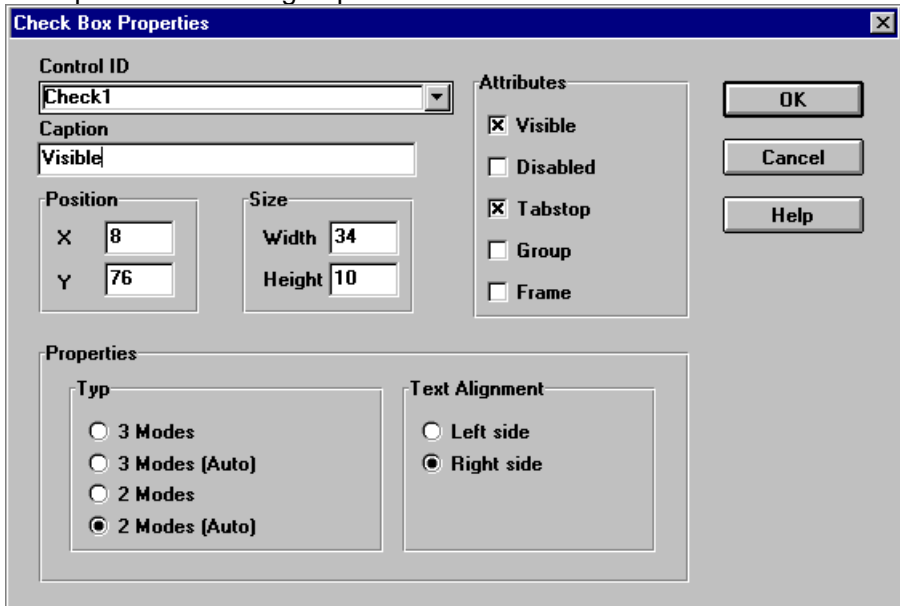
Disable Auto-Switch has no automatic effect. Switching only takes place in accordance with pre-defined program steps. The switching operation is merely displayed in the dialog window.

Text Alignment

Choose between having the text displayed to the right (default setting), or to the left of the Radio Button.

Check Box

The Check Box control allows a program option to be switched on or off. Unlike the previous types, it is possible to select or deselect more than one option box from a group.



In addition to the standard properties and attributes, the following settings may also be made:

Button Type

Four possible button types are available, and a distinction is made between buttons with two or three selection modes, with or without automatic switching.

Buttons with two selection modes can only be used to choose an active or inactive state.

Three button switches allow a third option mode. This third option can be designated to carry out any particular action within the application.

An automatic switching button will change mode as soon as it is selected. Selecting non automatic switching will cancel this effect. Switching will only take place in accordance with pre-defined program steps. The switching operation is merely displayed in the dialog window.

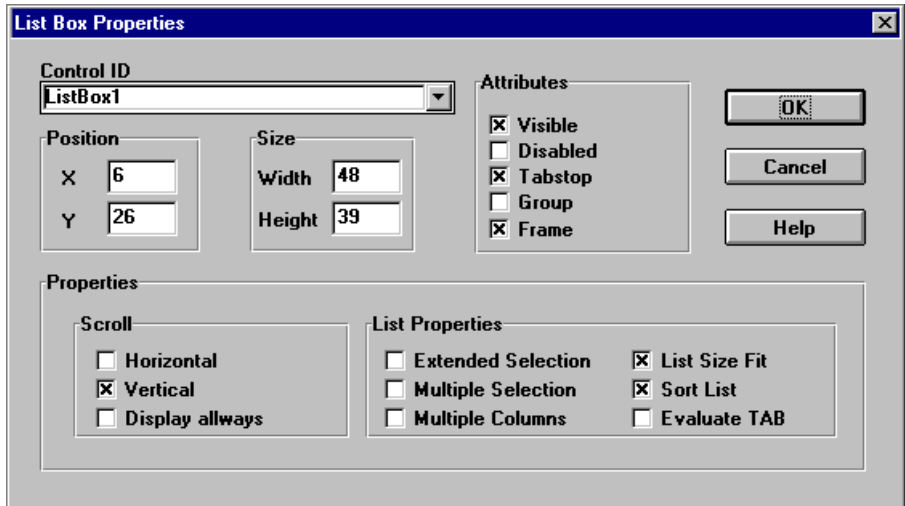
The check boxes can be selected with the cursor keys or the mouse pointer. The switching operation is activated with the space bar or mouse button.

Text Alignment

Choose between having the text displayed to the right (default setting), or to the left of the Check Box.

List Box

List boxes can display a number of different elements and are generally used for selection purposes. A typical example of a list box is a layer from a list box.



In addition to the standard properties and attributes the following settings may also be made:

Scroll Bars

By selecting the appropriate options, it is possible to add either a *Horizontal* or *Vertical* scroll bar to the list box.

The *Display Always* option has the effect of displaying scroll bars, even though they may not necessarily be needed. For example, when enough room is available within the list box to display all of the available options.

Extended Selection

When activated, this setting allows more than one element or entry to be selected from the list box. This can be achieved by pressing the Shift key during the selection process. A prerequisite is that the items are located immediately adjacent to one another in the list.

Multiple Selection

This option is similar to the extended selection, however, it does not require that the items are displayed below one another.

Multiple Columns

Allows the elements or entries to be displayed in more than one column. The entries can then be viewed by using the horizontal scroll function.

Height Reliant

This option automatically selects the required height so that all of the possible options or elements can be viewed at one time. If this parameter is deactivated, the list will open to the dimensions which were defined at the time of its creation.

Sort List

The elements or entries will appear in alphabetical order.

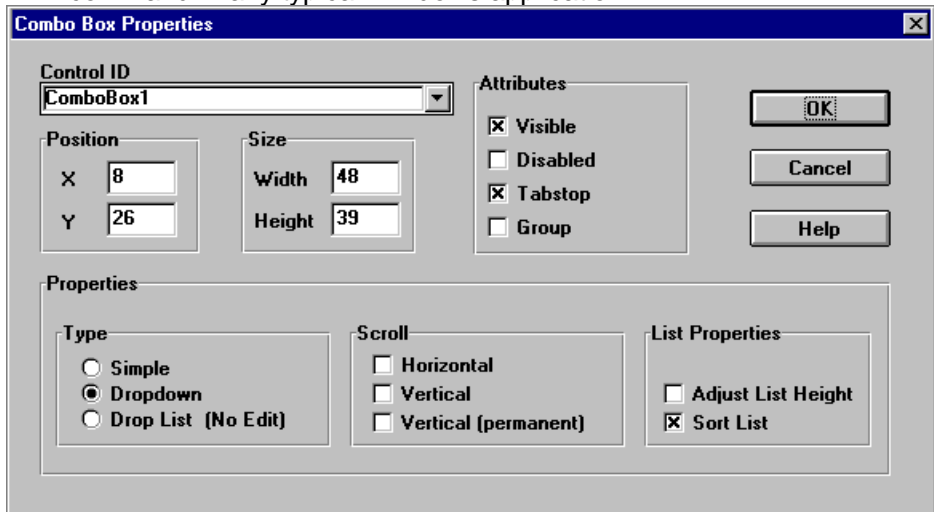
Tabulated (Evaluate TAB)

If active, this function will cause the display to be tabulator sensitive. This allows the elements or entries to be shown in list format.

Combo Box

A combo box links the properties of the list box with those of an input field (shown below). By doing so, it offers the user the added option of not only being able to select an element, but also allows him or her to add a new element or entry to the list.

The combo box consists of a combination of both a display field and an input field. If activated, a complete list of entries will appear as soon as the open button, which is located at the side of the combo field, is clicked. An example of a combo box can be seen by selecting the *Save As...* command in any typical Windows application.



In addition to the standard properties and attributes, please activate or deactivate the following settings:

Type

This option determines what will be displayed within the combined input and list box.

The option *Simple* combo box, dictates that the input and list box is shown to the maximum. The input field will contain the list element which is currently selected.

Unfold will only display the list box if the open symbol located at the side of the input field is clicked.

Unfold (non editable) enables the combo box and displays the selected item from the list, but does not permit the user to edit or make a new entry. The listed entries will only be displayed once the open symbol has been clicked.

Scroll

This option dictates whether horizontal and / or vertical scroll bars are to be included in the combo box. Activate or deactivate the appropriate check box.

List Box Entries

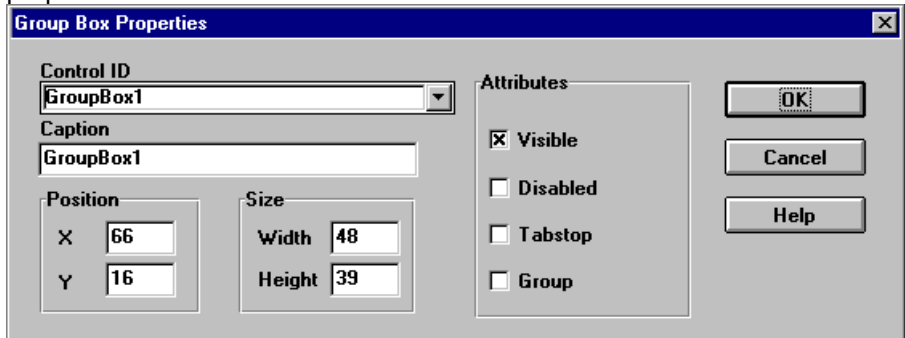
These will enable or disable the automatic height reliance option which adjusts the list box to suit the size and alphabetical order of the elements or entries. When deactivated, the list box will always be displayed according to the default size settings.

Group Box

Group frames can be inserted into the dialog window and help create a more logical and clear structure within the dialog box.

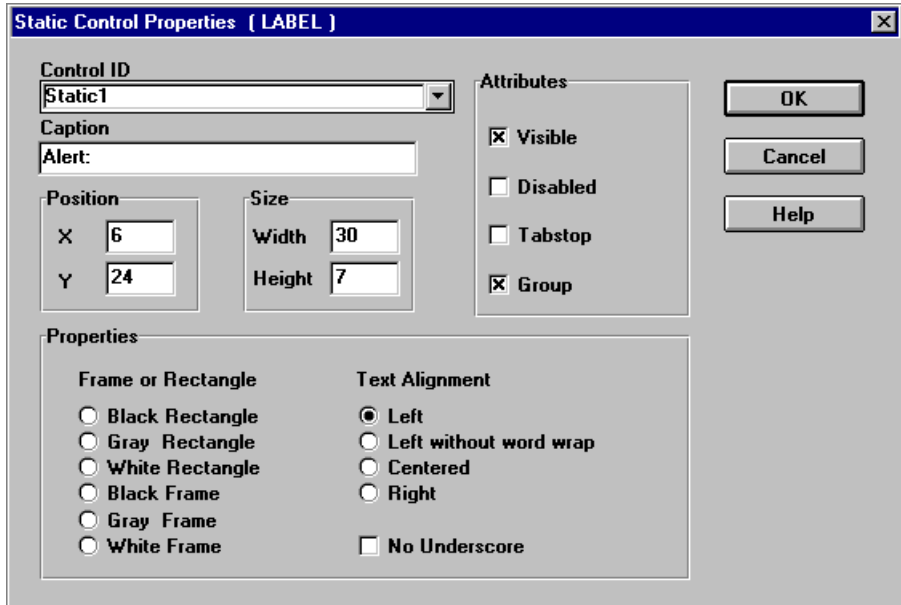
These group boxes help the user to identify which elements are directly related to one another.

Only the control identifier, a caption (title), the standard properties (position and size), and the attribute settings are required to define the Group Box properties.



Static Control (Label)

A static element (LABEL) is either used to insert a text block or a raised surface respectively a layout element in a dialog window. This enables you to “title” other control elements or raise the surface of a button so as to give it a more textured and predominant look.



In addition to the standard properties and attributes, static elements also include options to enable different graphical representations and text types within the control element.

Only one of the above options can be selected at any one time.

Frame or Rectangle

Selecting one of these options will insert a frame or rectangle static element. In this case no text will be displayed. Options are available to define the background color of the rectangle, or the color of the frame.

Text Alignment

This option sets the available parameters for the text block. Left alignment, left alignment without wrap around, centered or right alignment

No Underscore

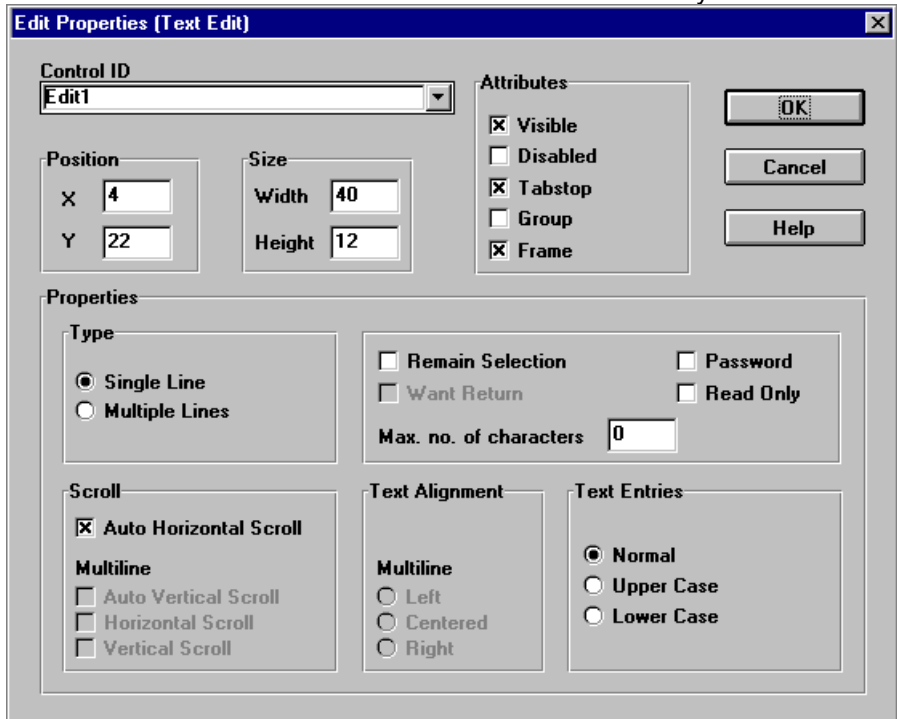
This option deactivates the “underline next character” option set with a preceding &-symbol. The &-symbol will now print normally.

Note: The selection of any one option from the text group will cause the static element color or design settings to be reset.

Text Edit Control

The Text Edit control allows alpha-numeric data to be typed in and processed by the application.

The control element can be defined as a single or multi-line input field and allocated vertical and horizontal scroll bars as necessary.



In addition to the standard properties and attributes, please activate or deactivate the following settings:

Single / Multi-input field

Use the available options to define either single or multi-line input.

Scroll

The horizontal scroll bar function can be assigned to single line input fields. Multi-line fields can be allocated both a horizontal and / or vertical scroll bar. In this case, the text will automatically scroll horizontally or vertically as soon as the input text becomes longer than the space available within the field. Select the appropriate options in the scroll bar group.

Text alignment

The text alignment options are only available for multi-line inputs. The possible options are left alignment, right alignment and centered. Click the desired alignment with the mouse pointer.

Text effects

This option can be used to correct the case of the entered text. For instance, upper case characters can be altered to lower case, and vice versa, if so desired.

Maintain highlight

Normally the text within a selected input field is only highlighted as long as the element is active, or being focused on. The NOHIDSEL attribute maintains the highlight effect even after the input focus has moved to another element.

Automatic Horizontal

If the user enters more text than can be displayed in the available space, then this option will automatically scroll or move the text 10 characters horizontally.

Password

The password option will cause every character entered into the input field to be displayed as an asterisks (*). This is useful if the user does not want entries to be seen on the screen, such as when entering a password.

CR Key Evaluation

When activated, this will cause the enter key to act as a carriage return or new line key, rather than as a send or confirmation key.

Read Only

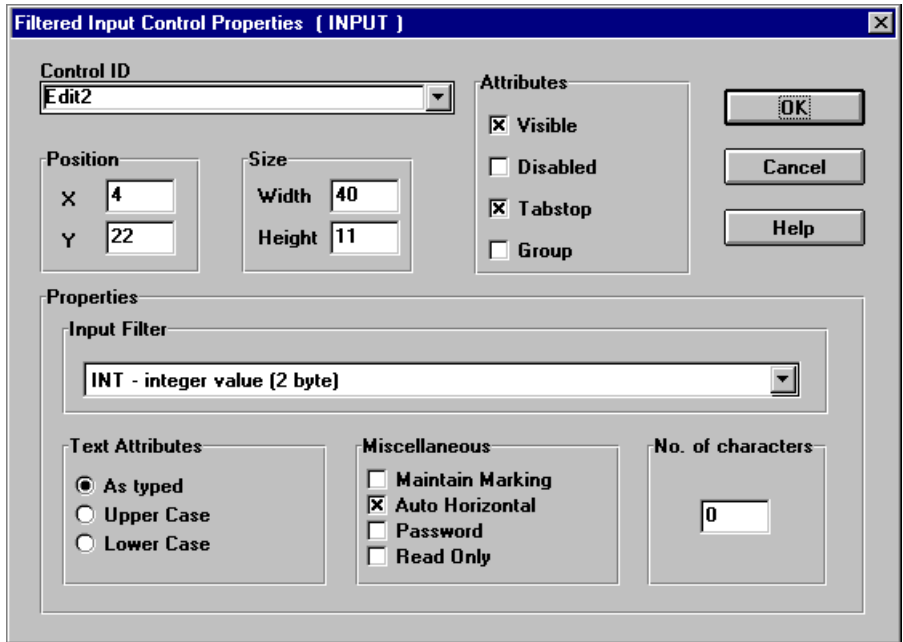
Text within this type of input field can be read, but not edited. This allows the user to restrict the situations where information can be entered.

Maximum Character restriction

It is sometimes necessary to restrict the number of characters which can be entered, this function has a default setting of 0 which allows up to a maximum of 32 KB to be stored.

Filtered Input Control

The INPUT control allows data of a specific type to be typed in and processed by the application. The properties and attributes of this control element are very similar to those in the Text Edit control. The difference is that the filtered text input can be used to select specific information, while ignoring those entries which do not meet the filter criterion.



Information regarding text displays and text properties can be read in the previous section on Input field control elements. The following selection options can be used when defining input filters:

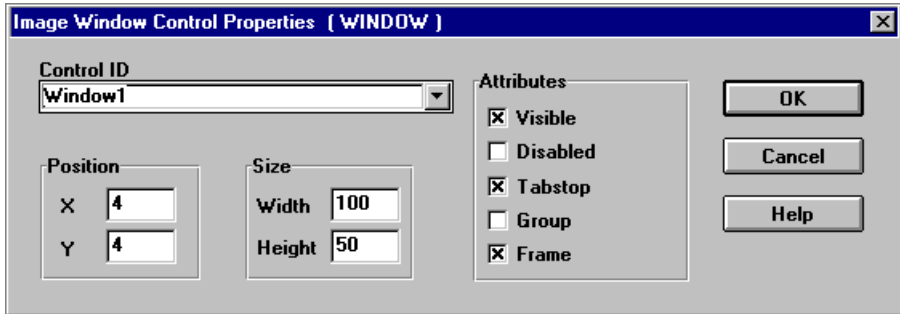
INT	Integer (2 byte)
WORD	Positive integer (2 byte)
LONG	Integer (4 byte)
DOUBLE LONG	Positive integer (4 byte)
FLOAT	Floating point (basic accuracy)
DOUBLE	Floating point (double accuracy)
LONG DOUBLE	Floating point (very accurate)
TEXT	Free flow text

The filter can be selected from a list containing the standard filter options. This list appears as soon as the open button located at the side of the FILTER display window is clicked.

Image Window Control

An image WINDOW or output field can be inserted into the dialog. This enables graphics (vectors, filled rectangles, BMP files, and WMF files) to be displayed as required. The window is defined by setting the following standard properties and attributes :

- Control name, position and size
- Visible, Not available, TAB, Group and frame



Slider Control

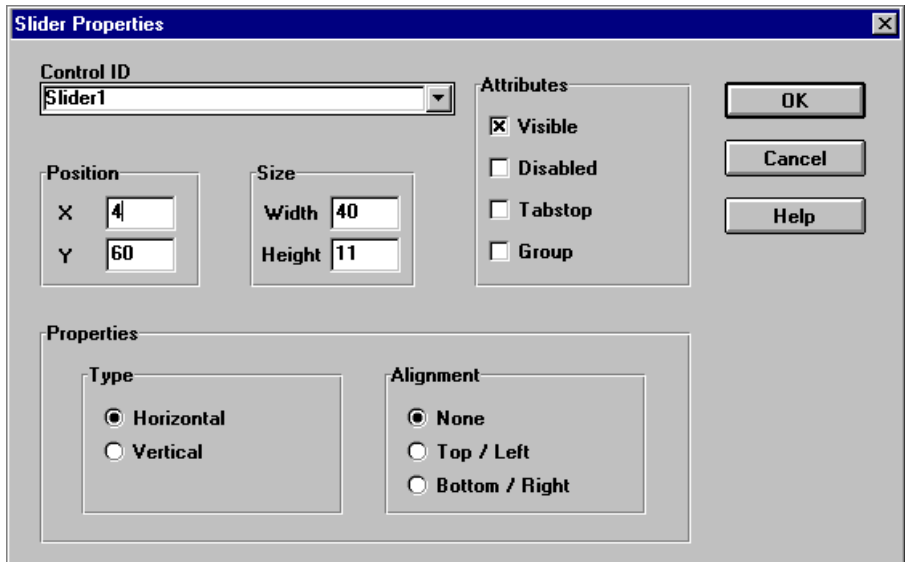
Some of the described elements, such as list boxes, combo boxes and input fields can contain scroll bars to enable a specific section of text to be displayed within the window boundaries.

The Scroll control element is designed to display a range of values for selection. Instead of typing in a (numeric) value, the user can use the scroll slider to move up or down until the desired figure is reached.

The slider position, as viewed within the scroll bar limitations, can be evaluated by the program and converted to a numeric value. This value could then be made to appear within a combo box display.

To enable this function it is necessary to program the slider range and step parameters into the application. The relevant information on how this is done can be found in the chapter, Programming Interfaces for the Application.

The scroll bar properties window, contains the options required to enter the standard properties and attributes.



Control Positioning Aids (Grid etc.)

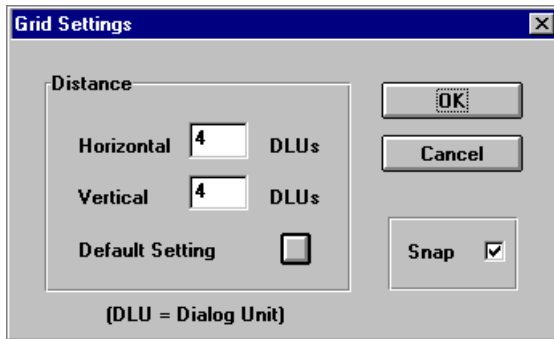
A number of features are available to help the user position and align control elements within a dialog window. These features make it easy to achieve excellent results, with a minimum of effort and time investment.

The *Grid* option can be activated or deactivated at will during the dialog creation process. The following methods can be used to display the grid:

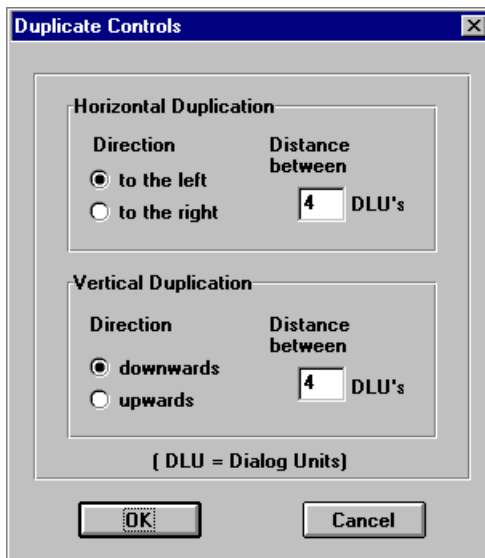
- Select the **Grid** option from the menu *View*
- Press the F10 function key
- Click the GRID button in the Alignment Palette

The snap and grid functions allow control elements to be inserted into a dialog aligned at absolute positions. This ensures that all elements can be aligned on both a horizontal and vertical plain. The grid parameters and snap-to-capture options can be set in the option **Grid...** shown in the menu *Options*. The following dialog window will open once this option is selected.

Enter the grid spacing values (horizontal / vertical) into the appropriate input fields. The unit of measurement is stated in dialog units (DLU's). The standard DLU value is four. By clicking the button *Default Setting*, the user will return both the horizontal and vertical spacing to this standard value. The snap function can be activated or deactivated by clicking the corresponding check button. A check mark in the control box indicates that the snap function has been activated.



Dialog design grid settings



Preferences for the Duplicate option

Control Alignment

The Dialog and Menu Editor offers a number of different methods by which the control elements can be positioned and aligned, both below one another and in reference to the window itself.

The following results can be obtained by using the positioning and alignment options:

- centered horizontally, or vertically in relation to the boarder limitations of the dialog window,
- below one another to the right, upper or lower left alignment,
- standard sizing
- standard spacing

Alignment Palette

The Alignment palette can be activated or deactivated by using either the *View* menu option, or by pressing the F9 function key.

The *Alignment* menu offers the added ability of allowing the palette to be displayed in a one, two, or three column mode.

The positioning and alignment functions are self explanatory and easy to use. By moving the cursor onto one of the palette icons, the user will notice that a brief explanation of the button function will be shown in the status line



Control Selection

Please note that the control elements which the user may want to position or alignment have been selected or marked prior to the allocation on any alignment parameters.

The easiest way to select a control element is by cursor selection. To select a number of different elements at one time, hold the (shift) key until all elements have been highlighted, or use the cursor to draw a selection frame around the required elements. Further instructions regarding the marking and selection of control elements can be found in the section entitled Editing Window, located at the beginning of this chapter.

Positioning and Aligning a Number of Control Elements

Functions which are used to position two or more elements will only become available when the appropriate number of elements have been marked or selected.

Alignment functions which position items below one another will only work when at least two buttons or windows have been selected. The equal

spacing function assumes that at least three elements are present and that they have been marked.

Testing a Dialog

Both during and after editing a dialog, the user is given the option of checking the design, functionality, position and alignment of the user window.

This feature will display the window exactly as it will appear within the application. All switch, selection and input properties can be checked for correct operation.

To call up the test function use either of the following options:

- Select the **Test** option from the menu *Dialog*.
- Click the Test button located in the Alignment Palette.

CHAPTER 10**LISP Programming**

With FLISP your CAD system provides a flexible LISP-Interpreter. The Lisp programming language suits the needs of technically-skilled people. FLISP is a complex tool to enhance the power of your application in many ways. The built-in LISP Interpreter allows you to add functions and routines tailored to your individual needs. One of the advantages of LISP is the short learning curve. In a short time you will be able to exploit the capabilities provided by this programming interface when working with your CAD application. The high value of providing the LISP interpreter with the system is not at least summarized by the following features:

- Easy to understand and use syntax
- Short learning curve
- Error handling and tracing is easy to implement
- Pre-testing of lisp expressions at command line
- High functionality from simple arithmetic calculations up to complex operations on drawing entities, selected objects, and drawing database tables
- Complex functions to display, fill and evaluate dialog boxes
- Create individual commands

Data Types in FLISP

Symbol	Symbols are named data objects. Symbols serve to name functions and variables. You can assign values to symbols to set variables. The name of a symbol can be specified out of any printable characters with the exception of: . ' " ; () Naming symbols is not case-sensitive. Example to set a symbol x to a value: (setq x 33)
String	A string expression may be of any length and contains a sequence of characters enclosed by " (double quotes).
Real Number	Floating point numbers (with one or more decimal places)
Integer	Number without decimal places in the range of -2147483648 through 2147483647
List	Assigns elements to a symbol, e.g. (1 2 3) or ("1" "2" "3") For example, 2D points and 3D points are expressed in lists. The list elements are real numbers representing the drawing coordinates like (1.0 2.0) or (1.0 2.0 3.0)
File Handle	Unique identifier for a file which has been opened with a Lisp function
Entity Name	Unique identifier for a drawing object
Selection Set Name	Unique identifier for a group of drawing objects (a set of selected entities)
Internal Function	Built-in function or a function defined with Lisp itself
External Function	Function provided by the application or defined by the Lisp programmer which may be executed like a built-in command

Loading Lisp Files

You can enter FLISP expressions or call FLISP functions at the command line. Another choice is to load lisp expressions from a file.

A file containing LISP code must be stored in an ASCII text file. The commonly used file extension is **.LSP**. However, you can use any file extension. But note, that an extension different to LSP needs to be passed to the load function.

To load a LISP file the function (**load ...**) is used. A detailed description is found in the functions reference. Example:

```
(load "test")
```

Normally the file contains one or more function definition. To define an individual function the LISP function (**defun ...**) is used. This allows you both to create your own functions and your own commands. Detailed information on the function (**defun ...**) is found in the function reference. Example:

```
(defun C:XYZ ()
  (princ "My new command 'XYZ' !")
  ...
)
```

Comments in LISP files

A semicolon (;) in a program's line indicates that the following text part to the line end is interpreted as comment. This text part is not evaluated. The string ;| starts a multi-line comment. This following text lines are ignored up to the closing string |; which terminates a comment.

Automatically loading a LISP file when opening a drawing

The local system variable LSPALOAD may contain a string specifying a Lisp filename. The value of this variable can be set in any drawing and is saved with the drawing file. By this, the local variable allows to automatically load a specified Lisp file the next time the drawing will be opened. Individual applications may use this possibility to auto-load a Lisp file by setting the variable LSPALOAD in a template drawing.

Error Handling and Error Tracing

FLISP reacts with printing an error message when an error occurred if an expression is evaluated. In many cases you want make sure that certain states and modes are restored when a function has been canceled and printing only the system's error message is not sufficient.

The function ***error*** allows to define your own error handler. The function has only one argument: a string to which the error message text of the system is passed. Instead of printing the message FLISP executes your error handler. Within your error handler you can evaluate the message and perform

functions to restore the setting of modified system variables, set back the state found before your routine has been performed, etc.

Using the expression (setq *error* nil) restores the error handler of the system.

Example for a function definition for error handling:

```
(defun *error* (msg)
  (if (= msg "User break")
      (setvar "OSMODE" osmode_sav)
      (progn
        (princ "The following error occurred: ")
        (princ msg)(terpri)
      )
  )
  (setq *error* nil)
  (princ)
)
```

When coding lisp programs it is often a great help to trace step by step the operations evaluated by the LISP interpreter. The function ***settrace*** provides several modes to view the code interpretation and to localize errors. For example, it is possible to display all evaluated lisp expressions until an error is recognized by the system. Detailed information is provided in the description of the function (***settrace* ...**) .

FLISP Function Overview: Thematically ordered

Function definition

(defun *symbol argument_list term...*) Defines a function

Error Handling

(alert *string string string*) Displays a message box
1 message
2 title
3 system icon

(*settrace* [*<integer> [integer]*]) Sets the mode for tracing of LISP routines and functions.
1. trace mode flag
2. react time in seconds

(*error* *<string>*) .. is the function for Error-Handling

System Functions

(command *arguments...*) Executes a built-in command with the arguments supplied

(delcmd *string*) De-Activates a specified command

(actcmd *string*) Re-Activates a specified command

(getvar *string*) Allows to retrieve system variable settings

(setvar *string value*) Sets a system variable

(findfile *string [value]*) Search for a file
1. file name
2. condition

(getfiled *string string string integer*) Provides a dialog for file selection
1. Dialog caption
2. Preset filename
3. preset extension
4. control flags

(osnap *point_list string*) "Snaps" a given point on an object

Geometric Utilities

(distance *point_list point_list*) Calculates the distance between two given points

(angle *point_list point_list*) Calculates the angle between two point

<code>(polar point_list real real)</code>	Returns a 3D point 1. point 2. distance 3. angle
<code>(inters point_list point_list point_list point_list [value])</code>	Returns the intersection of two lines
<code>(textbox list)</code>	Returns the bounding box corners of a text

User Input

<code>(initget [integer] [string])</code>	Initializes the next user input function
<code>(getreal [string])</code>	Prompts the user to enter a real number
<code>(getstring [value] [string])</code>	Prompts the user to enter a string
<code>(getpoint [point_list] [string])</code>	Prompts the user to specify a point
<code>(getcorner point_list [string])</code>	Prompts the user to specify a second corner of a rectangle
<code>(getdist [point_list] [string])</code>	Prompts the user to specify a distance
<code>(getangle [point_list] [string])</code>	Prompts the user to specify an angle
<code>(getorient [point_list] [string])</code>	Prompts the user to specify an angle (takes into account ANGBASE)
<code>(getkeyword [string])</code>	Prompts the user to select an option
<code>(getint [string])</code>	Prompts the user to enter an integer value

Conversion

<code>(rtos real [integer [integer]])</code>	Real number to String conversion 1. float 2. mode 3. precision
<code>(distof string [integer])</code>	Distance to Float conversion
<code>(angtos real [integer [integer]])</code>	Angle to String conversion 1. angle 2. mode 3. precision
<code>(angtof string [integer])</code>	Angle to Float conversion
<code>(symboltos value)</code>	Returns any lisp expressions in a string

Coordinate System Transformation

<code>(trans point_list value value integer)</code>	Transforms a point from one coordinate system to another 1. point 2. from 3. to 4. transformation
---	---

Display Control

(prin1 <i>[object [FileDescriptor]]</i>)	Prints a message on the command line or writes it to an open file
(princ <i>[object [FileDescriptor]]</i>)	Prints a message on the command line or writes it to an open file
(print <i>[object [FileDescriptor]]</i>)	Prints a message on the command line or writes it to an open file (like <i>prin1</i> , but with a preceding CR and a Space following)
(prompt <i>string</i>)	Displays a message on the command line
(terpri)	Outputs a new-line at the command line
(gread <i>[integer [integer]]</i>)	Reads from an input device <ol style="list-style-type: none"> 1. draw mode 2. cursor type
(redraw <i>[string [integer]]</i>)	Refresh current drawing display or redraw an entity <ol style="list-style-type: none"> 1. entity name 2. mode
(graphscr)	Closes the „Commands History / Lisp Interpreter“ window
(textscr)	Opens the „Commands History / Lisp Interpreter“ window

Selection Sets

(ssget <i>[string] [point [point]] [point_list] [assoc_list]</i>)	Creates a selection set. <ol style="list-style-type: none"> 1. mode 2. point 3. point 4. point list 5. filter list
(ssadd <i>[Ename [SelSet]]</i>)	Adds an entity to a selection set <ol style="list-style-type: none"> 1. entity name 2. selection set
(ssdel <i>Ename SelSet</i>)	Deletes an entity from a selection set <ol style="list-style-type: none"> 1. entity name 2 selection set
(sslength <i>SelSet</i>)	Returns the length of a selection set
(ssname <i>SelSet integer</i>)	Returns the n-th element of a selection set
(ssmemb <i>Ename SelSet</i>)	Verifies if an entity is member of a selection set
(ssdbno <i>SelSet</i>)	Returns the drawing database number (id) the selection set belongs to

Entity Handling

(entget <i>Ename</i> [<i>list</i>])	Returns entity information
(entmod <i>list</i>)	Modifies an entity
(entmake [<i>list</i>])	Creates an entity
(entdel <i>Ename</i>)	Deletes an entity
(entnext [<i>Ename</i>])	Returns the entity name which follows the given entity in the drawing database
(entlast)	Returns the last entity added to the drawing database
(handent <i>string</i>)	Returns the entity name to the entity that has the specified reference
(entsel [<i>string</i>])	Prompts the user to select an entity
(entupd <i>Ename</i>)	Updates a complex entity after modifications
(entpos <i>integer</i>)	Sets the database pointer

Symbol Tables

(tblDEL <i>string string</i>)	Deletes a table-entry
(tblMOD <i>list</i>)	Modifies a table-entry
(tblMAKE <i>list</i>)	Generates a new table-entry
(tblSET <i>string string</i>)	Sets an entry as the current
(tblNEXT <i>string</i> [<i>value</i>])	Returns the next table-entry
(tblPURGE <i>string integer</i>)	Deletes all not referenced entries of a table
(tblSEARCH <i>string string</i>)	Searches a table-entry
(tblRENAME <i>string string string</i>)	Renames a table-entry

Extended Entity Data

(regAPP <i>string</i>)	Registers an application name
--------------------------------	-------------------------------

Arithmetical Functions

(+ <i>number number ...</i>)	Sum of all numbers
(- <i>number number ...</i>)	Difference
(* <i>number number ...</i>)	Product
(/ <i>number number ...</i>)	Division
(~ <i>integer</i>)	Bitwise NOT
(1+ <i>number</i>)	Increment of number
(1- <i>number</i>)	Decrement of number
(abs <i>number</i>)	Absolute value of a number
(atan <i>number</i> [<i>number</i>])	Arc tangent of an angle supplied in radians
(cos <i>number</i>)	Cosine of an angle

(exp <i>number</i>)	Returns <i>e</i> raised to power <i>number</i> , where <i>e</i> is the base of the natural logarithm
(expt <i>number1 number2</i>)	Returns <i>number1</i> to the power of <i>number2</i>
(fix <i>number</i>)	Converts a real number to an integer
(float <i>number</i>)	Converts a given number into a real number
(gcd <i>number number ...</i>)	Returns the greatest common denominator of the given numbers
(log <i>number</i>)	Returns the natural logarithm of a given number
(logand <i>integer integer ...</i>)	Logical bitwise AND
(logior <i>integer integer ...</i>)	Logical bitwise OR
(lsh <i>integer integer</i>)	Logical bitwise shift
(max <i>number number...</i>)	Returns the largest number
(min <i>number number...</i>)	Returns the smallest number
(minusp <i>number</i>)	Tests if the number is negative
pi	Constant 3,141...
(rem <i>number1 number1</i>)	Divides <i>number1</i> by <i>number2</i> and returns the remainder
(sin <i>number</i>)	Sine of an angle
(sqrt <i>number</i>)	Square root of a number
(zerop <i>number</i>)	Zero or floating-point zero

Symbol Handling

(atom <i>term</i>)	Returns T if <i>term</i> is an atom
(atoms-family <i>0 1 [string_list]</i>)	Returns a list of all atoms currently defined
(boundp <i>value</i>)	Tests if the specified atom has a value
(not <i>argument</i>)	Returns T if argument is nil.
(null <i>object</i>)	Returns T if argument is nil.
(numberp <i>object</i>)	Returns T if its argument is any kind of number.
(quote <i>object</i>)	simply returns <i>object</i>
'object	Identical to (quote ...)
(set <i>symbol value</i>)	Sets the value of an quoted symbol to an expression
(setq <i>symbol value [symbol value]...</i>)	Sets the value of a symbol to an expression
(type <i>object</i>)	Returns the data type

Text Strings

<code>(read string)</code>	Retrieves the first atom or list from the given string and returns it according to its data type
<code>(read-char [FileDescriptor])</code>	Reads a single character from either the keyboard buffer or from an open file optionally specified by <i>FileDescriptor</i>
<code>(read-line [FileDescriptor])</code>	Reads a string from the keyboard buffer or from an open file optionally specified by <i>FileDescriptor</i>
<code>(strcase string [value])</code>	Converts a given <i>string</i> to a new upper case or lower case string.
<code>(strcat string string ...)</code>	Returns a new string concatenating two or more strings.
<code>(strlen string string...)</code>	Returns the sum of the length (number of characters) of all strings given as argument to the function.
<code>(substr string integer [integer])</code>	Returns a new - partial - string based on a given string
<code>(write-char integer [FileDescriptor])</code>	Writes a single character to file or to the command line.
<code>(write-line string [FileDescriptor])</code>	Writes a string to file or to the command line.
<code>(stringsort list)</code>	Sorts a list of string-items alphabetically, ascending
<code>(wcmatch string string)</code>	Allows a wild card pattern match search on a <i>string</i>

Conversion

<code>(ascii string)</code>	Converts the first character of <i>string</i> into its ASCII-Code, which is returned as integer
<code>(atof string)</code>	Converts a given string to a floating-point number, which is returned
<code>(atoi string)</code>	Converts a given string to an integer.
<code>(chr integer)</code>	Converts an ASCII code (given as integer) into its equivalent single-character-string
<code>(itoa integer)</code>	Returns the given <i>integer</i> as string

Equality / Conditional

<code>(= atom atom ...)</code>	Comparison: equal
<code>(/= atom atom ...)</code>	Comparison: not equal
<code>(< atom atom...)</code>	Comparison: less than
<code>(<= atom atom...)</code>	Comparison: less than or equal

<code>(> atom atom...)</code>	Comparison: greater than
<code>(>= atom atom...)</code>	Comparison: greater than or equal
<code>(and term term...)</code>	Logical AND
<code>(boole bit_value integer integer...)</code>	Performs a general bitwise Boolean function
<code>(cond</code> <code>(value value value ...)</code> <code>[(value value value...)]....</code> <code>)</code>	Is the primary conditional function
<code>(eq term term)</code>	Evaluates if two terms are identical
<code>(equal value value [number])</code>	Tests if two expressions have the same result
<code>(if</code> <code><value> <value></code> <code>[<value> value value [value]</code> <code>)</code>	Performs an conditional evaluation 1. test expression 2. then expression 3. [else expression]
<code>(or value value...)</code>	Logical OR
<code>(repeat integer value...)</code>	Executes a given expression <i>n</i> times
<code>(while value value...)</code>	The while construct allows iteration to continue until the specified expression evaluates to nil

List Manipulation

<code>(append list list...)</code>	Adds any number of lists together. Returns a new list
<code>(assoc value assoc_list)</code>	Searches an association list <i>assoc_list</i> for key
<code>(car list)</code>	Returns the first element of an list
<code>(cdr list)</code>	Returns a new list without the first element of a given list
<code>(c????r list)</code>	Combinations of car and cdr up to four levels
<code>(cons value list atom)</code>	Constructs a new list
<code>(foreach symbol list value...)</code>	Steps through a given list assigning each element to the specified symbol and evaluating the specified expression for each element of the list
<code>(last list)</code>	Returns the last element of a list
<code>(length list)</code>	Returns the number of elements in a list
<code>(list value...)</code>	Creates a list out of expressions supplied as arguments to the function
<code>(listp value)</code>	Verifies if <i>element</i> is a list

(mapcar <i>func_symb</i> <i>list_1 ... list_n</i>)	Operates on successive elements of the lists <i>list_1 ... list_n</i>
(member <i>value list</i>)	Searches the given <i>list</i> for an <i>element</i> and returns the remaining portion of the <i>list</i>
(nth <i>integer list</i>)	Returns the <i>nth</i> element of <i>list</i> . (<i>n</i> counting starts with 0)
(reverse <i>list</i>)	Reverses the given <i>list</i> and returns a new list
(subst <i>old_element new_element list</i>)	Copies a <i>list</i> substituting <i>old_element</i> by <i>new_element</i>

File Handling

(close <i>FileDescriptor</i>)	Closes the open file specified by <i>FileDescriptor</i>
(load <i>FileName [value]</i>)	Loads an existing lisp file
(open <i>FileName FileMode</i>)	Open a file given by <i>FileName</i> to read or write. <i>FileMode</i> can be: "r" open file to read "w" open file to write "a" open file to append

Function Handling

(apply <i>func_symbol list</i>)	Executes a function, where arguments are taken from <i>list</i>
(eval <i>value</i>)	Returns the result of a Lisp expression
(exit)	Terminates the current application
(lambda <i>argument_list value...</i>)	Allows the definition of an anonymous function
(progn <i>value value...</i>)	Evaluates one expression after the other grouped in progn .
(quit)	Terminates (cancels) the current application and returns to the command prompt.
(*settrace* [<i>integer [integer]</i>])	Sets the mode for tracing of Lisp routines and functions. 1. Trace mode 2. Time in seconds

Memory Management

(gc)	Garbage collection
(mem)	Displays the memory status of FLISP and returns nil

Miscellaneous

(getenv <i>string</i>)	Returns a dotted pair list containing the settings of the system paths and system files as defined in the application INI file
(ver)	Returns a string containing the information on the current version of the FLISP interpreter

FDT Application Handling

(fdt)	Returns a list of strings containing the loaded external FDT C-language applications
(xload <i>string</i> [<i>string</i>])	Loads an "external" FDT application's DLL
(xunload <i>string</i> [<i>string</i>])	De-activates an "external" FDT application

Help

(help [<i>string</i> [<i>string</i> [<i>string</i>]])	Calls WinHelp with a specified help topic <ol style="list-style-type: none"> 1. Topic 2. Help filename 3. Command
(setfunhelp <i>string string string</i>)	Set a help topic belonging to a function <ol style="list-style-type: none"> 1. Function 2. Topic 3. Helpfile

FLISP Functions: Reference

Within this section you find detailed descriptions of all functions provided by FLISP.

A paragraph to discuss a function contains normally the following items:

- Function name
- Short description of the functionality and operation
- Function syntax
- Parameters / Arguments to be passed to the function (detailed description on data types and meaning)
- Cross reference to other Lisp functions
- Example code applying the function

The functions reference is sorted alphabetically.

error

The function ***error*** allows handling of errors.

(*error* string)

A re-definition of this built-in functions allows you to implement application specific error handling.

For example, you can make sure that certain system variables are reset if a function has been canceled by the user or an error occurred in your function.

The argument to the function is a string which retrieves the error message passed to the function by the LISP interpreter.

The internal routine for error handling is restored by executing the expression
(setq *error* nil)

Example:

```
(defun *error* (msg)
  (princ "Error: ")
  (princ msg)
  (princ)
)
```

settrace

The function ***settrace*** sets the mode for tracing of LISP routines and functions.

Note: As soon as protected FLISP files are loaded, the function (*settrace* ...) does not function (by intention, for FCAD GDE developers).

When writing your own programs or when porting you should avoid loading protected lisp files.

(*settrace* variant [seconds])

The integer value *variant* may be combined (as follows):

0	No trace
1	Display allocated memory
8	Display function which caused an error
16	Display in case of an error all expressions up to function which caused the error
32	Display any LISP expression evaluated
32+64	Display any LISP expression evaluated (each line of code)

	needs to be confirmed by pressing any key
32+128	Display any LISP expression evaluated. Halts at the end of a line for a certain time specified by the argument <i>seconds</i> . The default value is 5 seconds.

+ (Addition)

This function returns the sum of all numbers given as argument(s).

(+ *number1 number2 [number...]*)

Examples:

(+ 5 2)

7

(+ 1.2 3.4)

4.60

(+ 5 1.2 2.8)

9.00

(setq n1 (+ 1.2 5))

6.20

(+ 3)

3

(+)

0

- (Subtraction)

This function subtracts the second number from the first and returns the difference.

(- *number1 number2 [number...]*)

Examples:

(- 5 2)

3

(- 5 2.0)

3.0000

(- 5 1.2 3.2)

0.6000

(- 5 1.2 3.4 5.6)

-5.2000

(- 5)

-5

* (Multiplication)

This function returns the product of all numbers given as arguments.

(* *number1 number2 [number...]*)

Examples:

(<i>* 5 2</i>)	(<i>* 5 2.0</i>)
10	10.0000

(<i>* 5 2 3</i>)	(<i>* 5</i>)
30	5

/ (Division)

This function divides the first number by the second and returns the quotient.

(/ *number1 number2 [number...]*)

Examples:

(/ 6 3)	(/ 5)
2	5

(/ 5 2)	(/ 5.0 2.0)
2	2.5

(/ 6 3.0)	(/ 7.0 2 2)
2.0000	1.7500

(/ 5 1.5)	(/ 6 2 2)
3.3333	1

(/ 7.135 2 1.2)	(/ 7 2 2)
2.9729	1

(/ 5.0 2)	(/ 9.9 3 3)
2.5000	1.1000

(/ 5 2.0)	(/ 9.9 6)
2.5000	1.6500

= (equal)

This function compares the given arguments. If they are equal, it returns T, otherwise nil.

(= *atom1 atom2 [atom...]*)

This function is valid for numbers and strings.

Examples:

(= 12.0 12)
T

(= 12 11.0)
nil

(= 12 12 12)

```
T
(= 12 12 11)
nil
(= "Test" "test")
nil
(= "TEST" "TEST")
T
```

See also:

eq
equal

/= (not equal)

This function compares the given arguments. If they are not equal, it returns T, otherwise nil.

(/= atom1 atom2 [atom...])

Examples:

(/= 2.0 2.00)	(/= 2 3)
nil	T
(/= 2.11 2.22)	(/= 2 2)
T	nil
(/= 2 2 2)	(/= "Test" "test")
nil	T
(/= 2 2 3)	(/= "TEST" "TEST")
T	nil

< (less than)

This function compares the given arguments. If each atom is less than the atom to its right, it returns T, otherwise nil.

(< atom1 atom2 [atom...])

Examples:

```
(< 1 2)
T
(< 1 2 3)
T
(< 2 3 1)
nil
(< "A" "B")
```


T

<= (less than or equal)

This function compares the given arguments. If each atom is less than or equal to the atom to its right, it returns T, otherwise nil.

(<= atom1 atom2 [atom...])

Examples:

(<= 1 2)

T

(<= 1 1 2 3 3)

T

(<= 1 1 2 1 3)

nil

> (greater than)

This function compares the given arguments. If each atom is greater than to the atom to its right, it returns T, otherwise nil.

(> atom1 atom2 [atom...])

Examples:

(> 1 2)

nil

(> 3 2 1)

T

(> 3 1 2)

nil

(> "A" "B")

nil

>= (greater than or equal)

This function compares the given arguments. If each atom is greater than or equal to the atom to its right, it returns T, otherwise nil.

(>= atom1 atom2 [atom...])

Examples:

(>= 2 1)

T

(>= 3 3 1 2)

nil

(>= 3 3 2 1)

T

1+ (Increment)

The increment function adds one to a given number and returns the result.

(1+ *number*)

Examples:

(1+ 1)

2

(1+ 1.0)

2.0000

1- (Decrement)

The decrement function subtracts one from a given number and returns the result.

(1- *number*)

Examples:

(1- 2)

1

(1- 2.0)

1.0000

~ (Bitwise NOT)

This is the bitwise NOT function. It returns the one's complement of a given integer.

(~ *integer*)

See also:

boole

logand

logior

lsh

Examples:

(~ 5)

-6

(~ -3)

2

abs

This function returns the absolute value of a given number.

(abs *number*)

Examples:

(abs 10)

10

(abs -10)

```
10
(abs -10.11)
10.1100
```

actcmd

The function **actcmd** re-activates a built-in command of the FCAD GDE (Graphic Developer's Engine), which has been de-activated prior by the function **delcmd**, for the user.

(actcmd *string*)

If the given argument is not a valid name of a system command, it returns nil, otherwise T.

Example:

```
(delcmd "LINE")          ; LINE is not available
T
line
Unknown command
(actcmd "LINE")          ; LINE is available
T
```

See also:

delcmd

alert




The function **alert** allows to display a *message* in an alert dialog box. To display Alert Boxes with certain icons like Exclamation, Question, Stop, Information specify the argument *mode*.


(alert *message* [*title* [*mode*]])

Parameters:

message is a string to be displayed in the dialog box
title is an optional string to be displayed in the header of the box
mode is an optional string to specify the type of the box.

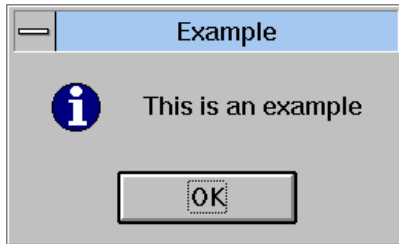
The following shows the various system icons that can be used:

Mode	Symbol	Buttons	Return Value
""	None	None	nil
"STOP"		OK	nil
"INFORMATION"		OK	nil
"EXCLAMATION"		OK	nil

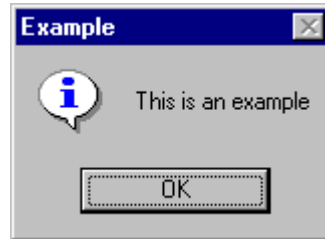
"QUESTION"		YES NO	T nil
------------	---	-----------	----------

Examples:

(alert "This is an example" "Example" "INFORMATION")



Windows NT 3.5, Windows 3.1x



Windows 95

and

This function returns the logical AND of multiple terms in a list. If all terms are bound (not nil) it returns T, otherwise nil.

(and [term] ...)

As soon as one of the terms evaluated is nil the function returns nil.

Examples:

(and 1.2 3 32)

T

(and 123 xyz 22) ; xyz is nil

nil

angle

The function **angle** returns the angle between the axis specified by two points.

(angle point_list point_list)

The function ignores Z coordinates different to zero of *point1* and *point2*.

The angle is measured counterclockwise from the x-axis of the current user coordinate system (UCS) . The value of the angle is returned in radians.

Examples:

(angle '(10.0 5.0) '(20.0 7.0))

0.1974

(angle '(2.0 3.33) '(4.6 5.7))

0.7392

angtof

The function **angtof** converts a *string* interpreted as an angle based upon a given *string-format* for a unit mode to a floating point number *value*. It returns the result expressed in radians.

(angtof string [mode])

If the argument *mode* is omitted, the function uses the current value of the system variable AUNITS (Angular Units).

If **angtof** not succeeds, it returns nil.

Angular units values

Mode	String format
-1	The current value of AUNITS is used.
0	Degrees
1	Degrees/minutes/seconds
2	Grads
3	Radians
4	Surveyor's units

Examples:

```
(angtof "31" 0)
```

```
0.5411
```

angtos

The function **angtos** converts a floating point *value* representing an angle to a string based upon the argument *mode* to specify the target unit system and the argument that specifies the desired *precision*.

(angtos value [mode [precision]])

Parameters:

value A floating point number representing an angle, specified in radians.

mode This argument specifies the unit system in which the string is formatted.

The value of *format* is interpreted as follows:

Mode	String format
-1	The current value of AUNITS is used.
0	Degrees
1	Degrees/minutes/seconds
2	Grads
3	Radians
4	Surveyor's units

precision: This argument specifies the number of decimal places for the resulting string.

Examples:

```
(angtos 0.5411 0 2)
```

```
"31.00"
```

```
(angtos 0.5411 1 2)
```

"31d"

append

The function **append** adds any number of lists together and returns a new list.

(append list [list...])

All specified arguments must be lists.

Examples:

```
(append '(1 2 3) (4 5 6))
```

```
(1 2 3 4 5 6)
```

```
(append '((abc) (def)) '((ghi) (jkl)))
```

```
((ABC) (DEF) (GHI) (JKL))
```

apply

The function **apply** executes a *function*, where arguments are taken from *list*.

(apply function list)

Examples:

```
(apply '+ (list 1 2.2 4.5))
```

```
7.7000
```

```
(apply 'strcat ('a "b" "cde"))
```

```
"abcde"
```

ascii

The function **ascii** converts the first character of *string* into its ASCII-Code, which is returned as integer.

(ascii string)

Examples:

```
(ascii "A")
```

```
65
```

```
(ascii "ABC")
```

```
65
```

```
(ascii "a")
```

```
97
```

```
(ascii "")
```

```
0
```

See also:

chr

assoc

The function **assoc** searches an association list *assoc_list* for *key*.

(assoc key assoc_list)

It returns the list entry if successful, otherwise nil.

Examples:

```
(setq article (list '(NAME "telephone") '(PRICE 149,00) '(COLOR "red")))
```

```
(assoc 'name article)
```

```
(NAME "telephone")
```

```
(assoc 'price article)
```

```
(PRICE 149,00)
```

```
(assoc 'color article)
```

```
(COLOR "red")
```

```
(assoc 'number article)
```

```
nil
```

atan

The function **atan** returns the arc tangent of an angle supplied in radians.

(atan number1 [number2])

If the 2nd argument *number2* is given, the function returns the arc tangent of *number1/number2* in radians. If this argument is zero, it returns an angle of $\mu 1.570$ radians.

The range of angles returned is $-\pi/2$ to $+\pi/2$ radians.

Examples:

```
(atan 0.8)
```

```
0.6747
```

```
(atan -0.8)
```

```
-0.6747
```

```
(atan 1)
```

```
0.7854
```

```
(atan 1 3)
```

```
0.3218
```

```
(atan -1 3)
```

```
-0.3218
```

See also:

angtos

atof

The function **atof** (ASCII to Float) converts a given string to a floating-point number, which is returned.

(atof *string*)

Examples:

(atof "32.1")

32.1000

(atof "12")

12.0000

(atof "55abcde")

55.0000

(atof "abcde")

0.0000

atoi

The function **atoi** (ASCII to Integer) converts a given string to an integer, which is returned.

(atoi *string*)

Examples:

(atoi "32.1")

32

(atoi "32.7")

32

(atoi "55")

55

(atoi "55abcde")

55

(atoi "abcde")

0

atom

The function **atom** tests if a given term is an atom or list. It returns T if *term* is an atom, otherwise nil.

(atom *term*)

Examples:

(atom '(1 2 3))

nil

(atom 123)

T

(atom (setq xyz "abc"))

T

atoms-family

The function **atoms-family** returns a list of all atoms currently defined.

(atoms-family *format* [*symbols*])

Parameters:

format 0 = returns a list of currently defined symbols
 1 = returns a list of currently defined symbols as strings

symbols optional list of strings that specify symbol names to search for.

Examples:

```
(atoms-family 1)
(!DO_PRINT "" **ERROR** **SETTRACE** .... "ZEROP" "~")
(atoms-family 0)
(!DO_PRINT **ERROR** **SETTRACE* ..... ZEROP ~)
(atoms-family 1 ("defun" "apply" "abc"))
("DEFUN" "APPLY" "ABC")
(atoms-family 1 ("defun" "apply" "dummy"))
("DEFUN" "APPLY" nil)
(atoms-family 0 '(defun apply abc))
Error: Invalid parameter
```

boole

The function **boole** performs a general bitwise Boolean function.

(boole *function* *number1* *number2* ...)

Parameters:

function an integer representing one of 16 possible Boolean functions

number1 integer

number2 integer

Boolean Truth Table

Function-Bit	number1	number2	Boolean function	
8	0	0	NOT	
4	0	1	XOR (6)	OR (7)
2	1	0		
1	1	1	AND	

Examples:

```
(boole 6 1 2)
3
(boole 1 1 2)
0
```

```
(boole 7 1 2)
3
```

boundp

The function **boundp** tests if the specified *atom* has a value. It returns T if a value is bound to *atom* (term), otherwise nil.

(boundp atom)

Examples:

```
(setq x 32)
32
(setq y nil)
nil
(boundp 'x)
T
(boundp 'y)
nil
(boundp 32)
nil
```

car

The function **car** returns the first element of a list.

(car list)

If *list* is empty, it returns nil.

Examples:

```
(setq lst1 '((a b c)(d e f g)))
((A B C) (D E F G))
(car lst1)
(A B C)
(setq lst2 '(a b c d e f g))
(A B C D E F G)
(car lst2)
A
```

cdr

The function **cdr** returns a new list without the first element of the list given as argument to the function.

(cdr list)

If *list* is empty, it returns nil.

Examples:

```
(setq lst1 '((a b c)(d e f g)))
((A B C) (D E F G))
```

```
(cdr lst1)
(D E F G)
(setq lst2 '(a b c d e f g))
(A B C D E F G)
(cdr lst2)
(B C D E F G)
```

caaaar ... cddddr

Combination of **car** and **cdr** up to **four levels**.

(c????r *list*)

Examples:

```
(setq x (list (list (list (list 1 2) 3 4) 5 6) 7 8))
```

```
((((1 2) 3 4) 5 6) 7 8)
```

```
(caaaar x)
```

```
1
```

```
(cdaaar x)
```

```
(2)
```

```
(cdar x)
```

```
(5 6)
```

```
(setq x (list (list 1 2) 3 4))
```

```
((1 2) 3 4)
```

```
(cadr x)
```

```
3
```

chr

The function **chr** converts an ASCII code (given as integer) into its equivalent single-character-string.

(chr *integer*)

Examples:

```
(chr 65)
```

```
"A"
```

```
(chr 33)
```

```
"!"
```

close

The function **close** closes the open file specified by *file_descriptor*.

(close *file_descriptor*)

Return value: nil

Example:

```
(setq d (open "abc.txt" "w"))
```

```
<File: #1e7f:4624>
```

(close d)
nil

command

The function **command** allows direct execution of one or more FCAD commands.

(command *command_string* [*argument ...*])

The built-in command to be executed must be expressed in **one** function call:

For example:

```
(command "line" "0,0" "1,0" "") .
```

The specified arguments are evaluated and passed to the command interpreter as responses to the command's request.

Inside of a command function you can call LISP expressions, for example, to retrieve user input. A null string ("") is equivalent to responding with RETURN to a command request. Example:

```
(command "move" (entsel "Select object:") "" p1 "1,1")
```

The parameters provided to the command function must not be complete to terminate the command invoked. If so, the system remains in the command to accept further user input until the command is finished.

Example: The following command expression starts the command LINE and supplies for the starting point the coordinates 0,0. The other points to draw line segments will be specified by the user:

```
(command "line" "0,0")
```

The system variable CMDECHO allows to control prompt display when executing the command:

- If the system variable CMDECHO is set to 1, the command(s) executed, the prompts of the built-in command(s), and the responses supplied within the function (command ...) are displayed at the command line.
- Otherwise, any prompt display is suppressed.

Return value:

nil

Examples:

```
(command "line" point1 "10,15" "")
```

```
(command "move" (entsel "Select object:") "" "0,0" "1,1")
```

cond

The function **cond** is the primary conditional function.

(cond (*test1* [*action1...]*) ...)

The function evaluates the first item in each given list until one of these items returns a value other than nil. It then evaluates each of the following items in this list. The function returns the value of the last expression in the sublist.

Examples:

```
(cond
  (< x 0)      (princ "X is less than 0")
  (= x 0)      (princ "X is equal 0")
  (T           (princ "X is greater than 0"))
)
```

cons

The function **cons** constructs a list by taking *first_element* and *list* and returns a new list.

(cons first_element list)

If the second argument is an atom, the function constructs a dotted pair list. Functions like **entmake** or **entmod** require this kind of list.

Examples:

```
(cons 8 "0")
(8 . "0")

(cons '1 '(2 3 4))
(1 2 3 4)

(cons '(1) '(2 3 4))
((1) 2 3 4)
```

cos

The function **cos** returns the cosine of an *angle* which is expressed in radians.

(cos angle)

Examples:

```
(cos (* 2 pi))
1.0000

(cos (/ pi 2))
0.0000
```

defun

This function defines function with the name *symbol*.

(defun symbol arguments term1 ...)

Parameters:

symbol name of the function
 Use the special prefix **C:** to **define** a new command. Commands can be entered from the command line without parentheses.

argument: list with arguments

Local variables are separated from arguments with a single slash character (/).

term1 ... one or more expressions to be evaluated

Return value:

The **defun** function returns the name of the function being defined. When the function is invoked, it returns the result of the last expression evaluated.

Examples:

(defun MYFUNC (a b c) (princ a)(terpri) (princ b)(terpri) (princ c)(terpri) (princ))	function with 3 arguments <u>Examples :</u> (myfunc 1 2 3) 1 2 3 (myfunc 1 2) 1 2 nil (myfunc 1 nil 3) 1 nil 3
(defun MYFUNC2 (/ a b) ...)	function without arguments 2 local variables
(defun MYFUNC3 (a / b c) ...)	function with 1 argument 2 local variables
(defun MYFUNC4 () ...)	function without arguments no local variables
(defun C:MYCMD () ...)	defines the command MYCMD

See also:

setq, load

delcmd

The function **delcmd** de-activates (deletes) a built-in CAD-Engine command. This disables the command for the user.

(delcmd string)

The de-activated command may be restored for the user with the function **actcmd**. If the given argument is not a valid name of a system command, it returns nil, otherwise T.

Parameters:

string Name of the system command

Example:

```
(delcmd "LINE")      ; The command LINE is no more available
T
>LINE
Unknown command
(actcmd "LINE")     ; LINE is restored now
T
```

See also:

actcmd

distance

The function **distance** calculates the distance between *point1* and *point2*. These points are assumed to be 3D points.

(distance *point1 point2*)

Examples:

```
(distance '(1 2) '(1 3))
1.0000
(distance '(1 2) '(1 2 3))
0.0000
(distance '(1 2 3) '(1 2 4))
1.0000
(distance '(1 2 3) '(4 5 6))
5.1962
```

distof

The function **distof** converts a given *string*, interpreted as real value, to a floating point number *value* depending on the argument *mode*.

(distof *string [mode]*)

Parameters:

string: string interpreted as a real number.
mode: specifies the format of the given string containing the unit mode for linear conversion.

The value given in *mode* is interpreted as follows:

Mode	Format
-1	The current value of the local system variable LUNITS (Linear Units) is used, which may contain one of the following settings (1 to 5).
1	Scientific representation
2	Decimal

3	English: Engineering (feet/inches)
4	English: Architectural (feet/fractional inches)
5	(1) English: Fractional

If the argument *mode* is omitted, the function uses the current value of the system variable LUNITS.

Return value:

If **distof** succeeds, it returns a real value, otherwise nil.

Examples:

(distof "1.234E02" 1) ; 1 = scientific representation
123.4000

(distof "123.4" 2) ; 2 = decimal
123.4000

(distof "10'-3.4\"" 3) ; 3 = engineering
123.4000

(distof "10'-3 6/16\"" 4) ; 4 = architectural
123.3750

(distof "123 6/16" 5) ; 5 = fractional
123.3750

dlg_***

The dialog functions are described in Chapter 11 of this manual.

entcheck

The function **entcheck** verifies, if the drawing entity specified by *ename* is (still) valid in the current drawing database.

(entcheck *ename*)

The function returns the entity name, if *ename* is valid for the current drawing database or is contained in a block definition of the current drawing; otherwise it returns nil.

Caution:

If a drawing object has been deleted, it is definitely removed from the drawing database. Note that an UNDO does not restore the previous entity reference (handle) and does not restore the *ename* of that entity. In this case a variable associated and containing the *ename* of the entity erased in the meantime is no more valid. This can lead to undefined conditions or states in later FDT or FLISP functions.

Examples:

```
(setq x (entlast))  
<Name: 432f5700>
```



```
(entcheck x)
<Ename: 432f5700>
(entdel x)
<Ename: 432f5700>
(entcheck x)
nil
```

entdel

The function **entdel** deletes the drawing object specified by *ename* in the current drawing database and erases that entity from the graphic window(s).

(entdel *ename*)

The function is performed drawing sensitive: the entity *ename* must be valid for the current drawing.

Return value:

If the function succeeds, it returns the entity name, otherwise it returns nil.

Example:

```
(entdel (entlast))
<Ename: 432f0ef4>
```

entget

The function **entget** returns the defining data of a drawing entity specified by its Entity Name *ename*.

(entget *ename* [*applications*])

Parameters:

ename: Entity Name of the drawing object, whose defining data are requested.

applications: The optional argument *applications* specifies the registered application name(s) in a list whose Application Entity Data should be retrieved by the function. The given names may contain wild cards. For valid wild card pattern match codes see the function **wcmatch** below in this section of the manual. The rules for the match codes are the same for both functions.

Return value:

If entget succeeds, it returns an association list containing the entity information, otherwise nil.

Only in the case that the group codes for Color or Linetype are BYLAYER this default value is not returned.

Examples:

```
(entget (entlast))
```

```
((-1 . <Name: 432f5c8e>) (0 . "LINE") (8 . "0") (5 . "00000005") (10 23.00  
23.00 0.00) (11 12.00 12.00 0.00))
```

```
; -1 = entity name  
; 0 = entity type  
; 8 = layer  
; 5 = handle  
; 10 = start point  
; 11 = end point
```

See also:

assoc, cdr, entmod

entlast

The function **entlast** retrieves the entity name of the drawing object most recently added to the current drawing database and returns the entity name.

(entlast)

Examples:

```
(entdel (entlast))
```

entmake

The function **entmake** allows to generate a new drawing entity (a basic entity or a complex object like a polyline or an insertion of type block) in the current drawing database and displays it in the current drawing.

(entmake data)

The argument *data* contains the defining data of the entity to be created.

The format of this list is similar to that returned by the **entget** function.

So, the function **entget** allows to retrieve the defining data of an existing drawing object and manipulate these entity data to create a new entity.

Also, in certain situations a technique may be applicable, where the entity whose defining data have been retrieved by the function **entget** is deleted and then restored as drawing object equivalent in entity type but modified in its geometric or property parameters by using the function **entmake**.

But note, in this case the entity reference (handles) and the database address (entity name) has changed although the visual representation of the entity in the graphic window(s) may be the same as before.

The function **entmake** tests the validity of the values supplied for Layer name, Color, Linetype and Text style before the entity is generated:

- If the layer name supplied is not found in the layer table of the current drawing, **entmake** creates that new layer using the default values of Layer "0".
- An invalid Color value leads to return nil.

- A linetype or text style not yet defined in the corresponding table of the current database leads to return nil.

The type of the entity must be provided in the first or second element of the list. If the type is supplied in the second buffer, the first buffer may contain only the entity name, which is ignored anyhow.

If an entity reference (Handle) is found in the list, it is ignored. The system automatically generates a new handle for the new created drawing object.

If an entity name (group code -1) is found in the list, it is ignored. The system generates a new Entity Name for the new object.

Creating Complex Objects

Complex objects (Polylines, Block-Definitions, Attributes of Block-Insertions) are created by multiple calls of **entmake**. The first call generates the base element (POLYLINE, BLOCK, or INSERT). The function recognizes that a complex object is effected and stores the current data contents and the following data functions temporary. The following calls of the function **entmake** will define the sub-elements (Vertices in POLYLINES, Attributes in BLOCK insertions, and entities defining a BLOCK). The sub-elements are added "temporary" up to the moment the following **entmake** function statement indicates the termination of the definition of a complex drawing object: This indication is done by specifying a ENDBLK "entity" (when defining a BLOCK) or a SEQEND "entity" (when defining a POLYLINE or an INSERT, which generates a BLOCK insertion).

The definition of a complex drawing object may be canceled by entering **entmake** with no arguments. The temporary stored object is then refused.

If an error occurs during the generation of a sub-entity, the sub-entity is refused, but the entities generated so far remain. This allows error handling by the developer.

Anonymous Part Definitions (BLOCK Definitions)

The BLOCK definition table in FCAD may contain anonymous blocks. A Cross-hatching and a dimension is an anonymous blocks. The user can not use anonymous blocks directly (for example in the command INSERT), because these block names stay "secret" to him. Only FDT and FLISP allow an access to anonymous blocks.

Anonymous blocks are created by the FCAD commands for dimensioning and cross-hatching. The application developer can generate anonymous block with the function **entmake**.

An anonymous block created with **entmake** has a block name like ***Uxxxxxx**, where the first two characters mark the block as created by **entmake**. The following characters contain a unique number to identify the block. This block number is only valid during the current FCAD session. If the drawing is re-opened, FCAD generates a new number for that anonymous block.

entmake recognizes automatically, that an anonymous block has to be created, if the block name (group 2) is **“*U“**. **entmake** adds a unique identifier to the internal block

name. Furthermore, **entmake** sets the flag "Block type" (group 70) in the block definition.

If the function succeeds, it returns the list with entity information's, otherwise nil.

Examples:

```
(entmake '((0 . "LINE")(10 0.0 0.0 0.0)(11 10.0 10.0 0.0))
; 10 - start point
; 11 - end point
```

entmod

The function **entmod** allows the modification of a basic entity or complex block definition existing in the current drawing database.

(entmod *data*)

The argument *data* contains the defining data of the entity to be modified. The entity to be modified must exist in the current drawing.

Before modifying the drawing element, **entmod** tests the validity of the supplied arguments for layer name, color, linetype and text style.

- If a not yet defined layer name is supplied, **entmod** creates that layer and initializes its color and linetype values corresponding to the default values of layer 0.
- An invalid color value leads to the cancellation of the function, which returns nil.
- A linetype or text style not yet defined leads to the cancellation of the function, which returns nil.

The name of the entity to be modified must be provided in the first element of the list (group = -2). Otherwise **entmod** cancels and returns nil.

The reference (handle) of the entity cannot be changed. The same is true for other internal fields like the entity name (group = -1).

Modifying a main (non-complex) entity causes automatically a redrawing of the changed object on the screen.

If the modified entity is a sub-entity (e.g. a vertex of a POLYLINE or an ATTRIB of an object of type INSERT), the display of the complex object needs to be refreshed (redrawn) on the screen with the function **entupd** which you find described below.

If the function succeeds, it returns the list with entity information's, otherwise nil.

Example:

```
(setq x (entget (car (entsel))))
(setq x (subst (cons 62 1) (assoc 62 x) x))
(entmod x)
```

See also:

entupd

entnext

The function **entnext** retrieves an entity in the current drawing database, which follows the entity specified by *ename* and returns its name.

(entnext [*ename*])

If *ename* contains the entity name of an object of type POLYLINE or INSERT (Block insertion), the function returns as next entity the first sub-element of that complex object (a Vertex or Attribute).

Entering entnext with no argument, it returns the first entity stored in the current drawing database.

Examples:

```
(setq ent1 (entnext))           ; First entity stored in database
(setq ent2 (entnext ent1))
```

entpos

The function **entpos** sets the database pointer in the entity section to an absolute or relative *position* and returns the name of the entity found at that position.

(entpos *position*)

The flag argument *position* sets the entity-name-pointer in the current drawing database entity section. Valid flags are:

Position	Description
0	First entity of the current drawing database
1	Last entity of the current drawing database
2	Succeeding (next) entity of the current drawing database
3	Preceding (previous) entity of the current drawing database
4	Current entity (entity at the current <i>ename</i> pointer position)

The function allows only to retrieve "main" (non-complex) entities, but not vertices, attributes etc.

See the function description of **entnext** to evaluate sub-elements of complex objects.

entsel

The function **entsel** allows the developer to let the user to select a single entity.

(entsel [*prompt*])

The user is prompted with a string contained in *prompt* to select or identify a single drawing object. Entering **entsel** with no argument the standard prompt of the FCAD Engine ("Select object: ") is displayed.

The function returns a list containing the entity name of the drawing object and the pick point coordinates (of the current UCS).
The function returns nil if the drawing is empty.

Examples:

```
(entsel)
Select object:
(<Ename: 44dfcf6c> (5.7398 6.3448 0.0000))
(entsel "Select an arc or a circle: ")
Select an arc or a circle:
(<Ename: 44dfcf6c> (5.2217 7.1699 0.0000))
```

entupd

The function **entupd** allows a display-update of a drawing object specified by *ename* in the graphic window(s). Points *ename* to the name of a sub-element of an complex object, the entire object is redrawn.

(entupd *ename*)

This function is especially used to update the display of a complex object when it has been modified.

If **distof** succeeds, it returns *ename*, otherwise nil.

See also:

entmod

eq

The function **eq** evaluates if two terms are identical.

(eq *term1 term2*)

It returns T if the first expression and the second expression are bound to the same object, otherwise it returns nil.

Examples:

```
(setq a '(1 2 3))
(setq b '(1 2 3))
(setq c b)
(eq a b)
nil
(eq b c)
T
```

See also:

equal, =

equal

The function **equal** evaluates if two terms have the same result. The optional third argument may specify a tolerance form comparison of numbers.

(equal term1 term2 [tolerance])

Examples:

```
(setq a (list 1 2 3))      (setq a 1.0)
(setq b (list 1 2 3))      (setq b 1.0)
(setq c a)                  (setq c a)
```

EQ	EQUAL	EQ	EQUAL
(eq a c) T	(equal a c) T	(eq a c) T	(equal a c) T
(eq a b) nil	(equal a b) T	(eq a b) T	(equal a b) T
(eq b c) nil	(equal b c) T	(eq b c) T	(equal b c) T

```
(setq a 0.1234)
(setq b 0.1235)
(eq a b)
nil
(equal a b)
nil
(equal a b 0.001)
T
```

See also:

eq, =

eval

The function **eval** returns the result of a LISP expression.

(eval *lisp_expression*)

Examples:

```
(setq a>(* 4 5))
(setq b 'a)
(eval a)
20
(eval b)
(* 4 5)
```

exit

The function **exit** terminates the current application.

(exit)

The following error message is displayed:

Warning: Program terminated by EXIT

See also:

quit

exp

The function **exp** returns e raised to power number, where e is the base of the natural logarithm.

(exp number)

Examples:

(exp 0)

1.0000

(exp 3)

20.0855

(exp -3)

0.0498

expt

The function **expt** returns *base_number* to the *power_number* .

(expt base_number power_number)

Examples:

(expt 2 3)

8

(expt 11 0)

1

(expt 13.0 1.0)

13.0000

fdt

The function **fdt** returns a list of strings containing the loaded external FDT C-language based applications. The list includes drive, path(s), and filename.

(fdt)

FDT = **FCAD Development Toolkit**

Example:

(fdt)

("C:\\FCAD\\APK\\SAMPLE.DLL")

(fdt)

()

See also**xload, xunload****findfile**

The function **findfile** searches the file *filename* and returns the complete name including drive and directory path.

(findfile filename [condition])

1. If condition = nil:

Search for a file in a certain order: Returns a complete file name (including drive and path) of the file found first in the (application) search path. It returns nil, if no file has been found.

2. If condition = T

Returns a list of file names in the specified or in the current drive/path

- the list of file names is not sorted
- a path name is not returned

The order to browse directories to find the specified file is as follows:

1. Current directory
2. Drawing directory as specified in the FCAD application INI file (FCADDWG)
3. Application search path: directories as specified in Application INI file (FCADSUP)
4. Path containing the FCAD OEM kernel system files as specified in Application INI file (FCAD)

Examples:

```
(findfile "J:\\FCAD\\SAMPLE\\*.flx" T)
("1.FLX" "APKPROTO.FLX" "NONAME_0.FLX")
(findfile "FELIX.EXE")
"J:\\FCAD\\BIN\\FELIX.EXE"
```

See also:**getfiled****fix**

The function **fix** converts a real number to an integer.

(fix number)

The fractional portion of the floating point value is truncated.

Examples:

```
(fix 2)
2
(fix 2.22)
2
```

```
(fix 2.77)
2
(fix 3000000000)
2147483647
```

float

The function **float** converts a given number into a real number.
(float *number*)

Examples:

```
(float 0.33)
0.3300
(float 1)
1.0000
```

flxnames

The function returns the open drawing files as dotted pair list.
(flxnames)

It returns nil if no drawing is open.

Examples:

```
(flxnames)
((0 . "J:\\FCAD\\SAMPLE\\TEST.FLX"))
```

foreach

These function steps through a given list assigning each element to the specified symbol and evaluating the specified expression for each element of the list.

(foreach *symbol list term1 ...*)

foreach returns the result of the last expression evaluated.

Examples:

```
(foreach el '(1 2 3) (princ el))
1233
(setq x 0)
0
(foreach el '(1 2 3) (setq x (+ x el)) (princ x)(terpri))
1
3
6
nil
```

gc

The function **gc** releases unused allocated memory.

(gc)

GC = **G**arbage **C**ollection

gcd

The function **gcd** returns the greatest common denominator of the given numbers.

(gcd *number1 number2*)

Examples:

(gcd 20 50)

10

(gcd 33 47)

1

getactvport

The function **getactvport** returns in a list the database number of the current drawing and the vport number of the current viewport.

(getactvport)

Example:

(getactvport)

(1 1)

See also:

setactvport

getangle

The function **getangle** prompts the user to specify an angle. The user can enter the angle as a numerical value at the keyboard or identify the angle in the drawing.

(getangle [*point*] [*prompt*])

Parameters:

point specifies the base point in the current UCS to compute the angle. If the argument is supplied, the user specifies the angle by entering or identifying a second point.

If *point* is not specified or nil, the user has to specify two points, where the first point entered is assumed to be the base point for the angle inquiry.

To visualize the angle specification, a rubberband is drawn from the base point to the current cursor position.

The user can keyboard angles in a format recognized by the function **angtof**.

prompt If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

The computed angle is returned in radians.

Zero radians (zero direction of angles) is defined by the local system variable ANGBASE (Angle Base).

Normally, an angle increases in a counter-clockwise direction.

The angle returned is based upon the current construction plane. The function interprets an angle entered via the keyboard under recognition of the local system variable AUNITS (Angle Units).

Examples:

ANGBASE

New value for ANGBASE <0.0000>: 0

ANGDIR

New value for ANGDIR <1>: 0

(getangle)

0,0

Next point: 0,1

1.5708

ANGDIR

New value for ANGDIR <0>: 1

(getangle)

0,0

Next point: 0,1

-1.5708

See also:

getorient, initget

getcorner

The function **getcorner** prompts the user to enter the opposite corner of a rectangle.

(getcorner point [prompt])

Parameters:

- point:** This argument specifies the base point of the rectangle. The base point is specified in coordinates of the current UCS. To visualize the corner specification, a box is drawn from the base point to the current cursor position.
- prompt:** If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

The function returns a valid point specified by the user in coordinates of the current user coordinate system (UCS).

The function attempts to interpret keyboard input as point specification.

Examples:

(getcorner '(10 10) "Second point:")

See also:

getpoint
initget

getdist

The function **getdist** pauses for user input to retrieve the distance between a given base point and a second point or between two points to be specified by the user. The distance computed by the function is returned in the variable *distance*.

(getdist [point] [prompt])*Parameters:*

- point:** This argument specifies the base point for finding the distance. The base point is specified in coordinates of the current UCS. If *point* is not specified or nil, the user has to specify two points, where the first point entered is assumed to be the base point for the distance inquiry. To visualize the distance specification, a rubberband is drawn from the base point to the current cursor position.
- prompt:** If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

The function attempts to interpret keyboard input as point specification.

The function returns the computed distance.

Examples:

```
(getdist)
(getdist "Please enter a distance:")
(getdist '(0 0) "Length:")
```

See also:

initget

getenv

The function **getenv** allows to retrieve the system directory path or file preferences set in the FCAD-OEM application INI file.

(getenv)

The function returns a dotted pair list containing the settings of the system paths and system files as defined in the application INI file.

Keyword	Description
---------	-------------

"FCADSYS"	FCAD Engine system directory (containing kernel executables and DLL's). Specified in the INI file as FCAD=
"FCADCFG"	Path to the user's configuration directory
"FCADDEV"	Directory for FCAD's device driver interface files (currently only tablet driver interface DLL; normally same as kernel directory - see "FCADSYS")
"FCADMNU"	Path for menu files (MNU, MNP, MNT)
"FCADSUP"	Application search path (one or multiple directory paths separated by semicolon)
"FCADTMP"	Directory for temporary user-specific files (Undo-List files and Display-List files)
"FCADHLP"	HLP filename (Help file)
"FCADDWG"	Default directory for drawings
"FCADCMD"	FCAD engine system directory (containing the command DLL's; normally same as kernel directory - see "FCADSYS")
"FCADDLG"	Path for dialog files (DLG files)
"FCADKEY"	KEY filename

Examples:

```
(setq SYSPATH (getenv))
(("FCADSYS" . "C:\\FCAD\\GDE")
 ("FCADCFG" . "C:\\FELIX\\JFCAD")
 ("FCADDEV" . "C:\\FCAD\\GDE")
 ("FCADMNU" . "C:\\FCAD\\APK")
 ("FCADSUP" . "C:\\FCAD\\APPLIC;C:\\FCAD\\FONTS")
 ("FCADTMP" . "C:\\FELIX\\JFCAD")
 ("FCADHLP" . "C:\\FCAD\\HELP\\FCAD.HLP")
 ("FCADDWG" . "C:\\FCAD\\SAMPLE")
 ("FCADCMD" . "C:\\FCAD\\GDE")
 ("FCADDLG" . "C:\\FCAD")
 ("FCADKEY" . "C:\\FCAD\\FCAD.KEY")
)
(setq fcfg (cdr (assoc "FCADCFG" SYSPATH)))
"C:\\FELIX\\JFCAD"
```

getfiled

The function **getfiled** displays a standard dialog box to let the user specify a filename, for example to open an existing or new file.

(getfiled title preset_file extension flags)

Parameters:

title: Caption of the dialog box.

preset_file: This string may contain a filename recommended as default value for filename specification.

extension: This string may contain one or more pre-defined file extension(s). To specify multiple file types, the file extensions must separate by comma. If "" is supplied as argument no file type restriction ("*.") is set.

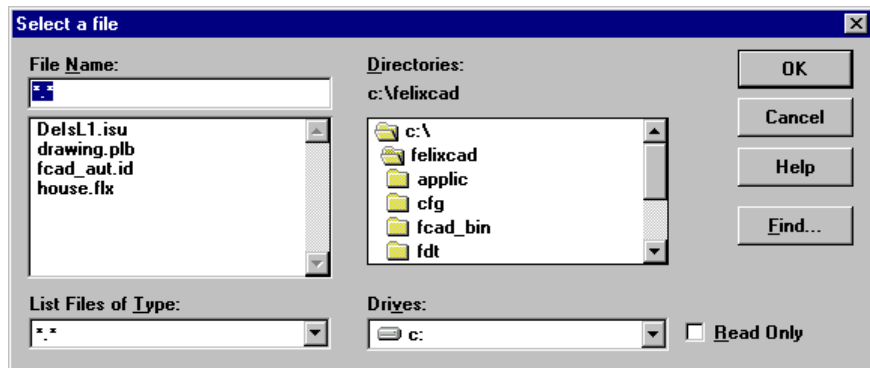
flags: In the argument *flags* bit-coded control flags are set, which have the following meaning:

Flag	Meaning
Bit 0	Set: A dialog to open a new file is displayed. Otherwise: A dialog to open an existing file is displayed.
Bit 1	Not used.
Bit 2	Set: Allows the user to specify any file extension. Otherwise: Allows to use only the file extension set as default
Bit 3	Not used.

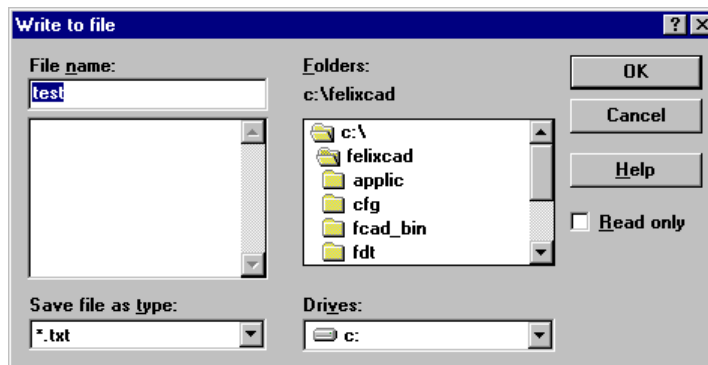
If a valid filename has been specified by the user, it is returned. Otherwise it returns nil.

Examples:

(getfiled "Select a file" "" "" 0)



(getfiled "Write to file" "test.txt" "txt" 1)



getint

The function **getint** prompts the user to enter an integer value and returns the value specified by the user.

(getint [*prompt*])

The integer value must lie in the range of -32768 to 32767. If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

Examples:

```
(getint)
```

```
(getint "Please enter the value:")
```

See also:

initget

getreal

getkeyword

The function **getkeyword** prompts the user to enter a keyword defined by **initget**.

(getkeyword [*prompt*])

Parameters:

prompt: If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

Keywords established by a prior call of the function **initget**.

Return value:

The function returns the keyword entered or picked by the user or nil.

Examples:

```
(initget 257 "Yes No")
```

```
(getkeyword "Erase entities ? (Y/N): ")
```

See also:

initget

getorient

The function **getorient** prompts the user to enter an angle.

(getorient [*point*] [*prompt*])

Parameters:

point: *point* specifies a 2D base point in the current UCS to compute the angle. If the argument is supplied, the user specifies the angle by entering or identifying a second point.

If *point* is not specified or nil, the user has to specify two points, where the first point entered is assumed to be the base point for the angle inquiry.

To visualize the angle specification, a rubberband is drawn from the base point to the current cursor position.

prompt: If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

The user can keyboard angles in a format recognized by the function `angtof`. The function works similar to `getangle`, but the angle returned is always measured as zero degrees being to the right (east). The function `getangle`, described above, recognizes the local system variable `ANGBASE` (Angle Base), `getorient` ignores it.

But, the function utilizes like `getangle` the system variables `ANGDIR` and `ANGBASE`. The returned value represents the setting of `ANGBASE` added to the result. For example:

If `ANGDIR` is set to 0 and `ANGBASE` is set to 90 degrees, the function returns an angle of 90 degrees as 3.14159 in radians (180 degrees).

If `ANGDIR` is set to 1 and `ANGBASE` is set to 0 degrees, the function returns an angle of 90 degrees as 4.7124 in radians.

The function interprets an angle entered via the keyboard under recognition of the local system variable `AUNITS` (Angle Units).

The function returns the computed orientation. The angle is returned in radians. The function returns the angle with respect to the current construction plane.

Examples:

```
(getorient)
```

```
0,0
```

```
Next point: 0,1
```

```
1.5708
```

```
(getorient)
```

```
90
```

```
1.5708
```

```
(getorient "Angle: ")
```

```
Angle: 45
```

```
0.7854
```

See also:

`initget`, `getangle`

getpoint

The function **getpoint** prompts the user to enter a point. The point identified (in the current user coordinate system) is returned.

(getpoint [*point*] [*prompt*])

Parameters:

point: The argument *point* may serve as base point in the current UCS for the point specification. To visualize point specification in relation to a given base point, a rubberband is drawn from the base point to the current cursor position.

prompt: If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

The function attempts to interpret keyboard input as point specification.

Return value:

The function returns the point specified by the user.

Examples:

```
(getpoint)
```

```
(getpoint '(0 0) "To point")
```

```
(getpoint "Point:")
```

See also:

initget

getreal

The function **getreal** waits for user input of a floating point value and returns the real number.

(getreal [*prompt*])

prompt: If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

Examples:

```
(getreal)
```

```
(getreal "Width:")
```

See also:

initget, getint

getstring

The function **getstring** prompts the user to enter a string.

(getstring [flag] [prompt])*Parameters:*

- flag:** Control flag. If *flag* is set to FALSE (0) the input of a space, of RETURN, or of TAB is interpreted as termination of the string input. Otherwise, the input is terminated by RETURN entered by the user. This means that the retrieved string may contain blanks.
- prompt:** If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise no prompt is displayed.

The function returns the string entered by the user.

Examples:

```
(getstring)
```

```
(getstring T "Please enter your name:")
```

```
(getstring "Title:")
```

See also:

getkeyword

getvar

The function **getvar** allows to retrieve global or local system variable settings.

(getvar variable)**Examples:**

```
(getvar "CMDECHO")
```

```
1
```

```
(getvar "FCVERSION")
```

```
"FCAD 1.0"
```

See also:

setvar

graphscr

The function **graphscr** closes the Lisp/History text window (if opened). The graphic desktop with the standard command line area is set *on top*.

(graphscr)

The function returns nil.

See also:

textscr

grclear

The function **grclear** clears the current viewport of all graphics, although the objects are not deleted in the drawing.

(grclear)

Use the **redraw()** function to restore the original display of the drawing window.

The function returns nil.

See also:

grdraw

grdraw

The function **grdraw** draws a vector between two points in the specified color to the current viewport.

(grdraw start end color [flag])

The vector drawn on the graphics screen is not written to the drawing database.

Parameters:

start: Startpoint of the vector expressed in coordinates of the current UCS.

end: Endpoint of the vector expressed in coordinates of the current UCS.

color: Number in the range of 0 through 255 to specify the color the vector is drawn.

flag: This parameter specifies if set non-zero to highlight the vector; otherwise it is drawn in normal mode.

If **grdraw** succeeds, it returns T, otherwise nil.

Example:

```
(grdraw '(0 0) '(10 10) 2)
```

See also:

grclear

grread

The function **grread** allows to read the next user input directly.

(grread [flag] [cursor])

Parameters:

flag: The parameter *flag* allows to control the input mode.

The way the function is applied and performed is specified by control bits, which has the following meaning:

Flag	Meaning
------	---------

1	Dragmode: As the cursor is moved, the coordinates are tracked permanently.
2	Any single input is terminated immediately and returned.
4	Allows to specify a cursor type in the argument <i>type</i> (see below)
8	Suppresses displaying ***Cancel*** in the command line if ESC has been pressed by the user.

cursor: This parameter sets the cursor type used by the function:

Value	Meaning
0	Standard cursor
1	No cursor
2	Select Box

Return value:

The function returns a list. The first element of the list specifies the kind of input device used. The second element is the content of the input dependent on the kind of input is stored as documented in the table below.

First element	Second element
2- Keyboard input	String containing the keyboard input
3- Point input	Point coordinates (picked point)
5- Drag point	Point coordinates (drag point)
101- Pull-down menu selection	String containing the MNU-file entry of the selected pull-down menu item
201...210 Palette button selection	String containing the MNP-file entry of the selected palette button
301 - Shortcut (if specified in the USER.KEY file section [SHORTCUTS])	String containing the KEY-file entry of the shortcut used
401 - System button selection (from the function bar or the tool panel)	String containing the command string of the selected button
501 - Tablet menu selection	String containing the index of the current MNT-file (tablet menu)

Examples:

```
(gread)
(3 (2687.00 2036.50 0.00)) ; picked point
(gread 2)
A(2 65) ; entered A
(gread 4 2) ; Cursor = Select Box
```

(2 "") ; entered RETURN

grtext

The function **grtext** displays a text in the message area of the status bar of the FCAD desktop.

(grtext string)

This message area of the status line is especially meant to display short context sensitive help strings as certain elements of the user interface are touched (but not yet clicked) by the cursor (a button of a palette, a system button, a tablet menu field, or a highlighted pull-down menu entry) or to display relative distances and angles as the cursor is moved when a point entry is requested.

The function **grtext** may be utilized by the developer to display cursor positions (absolute or polar coordinates) in the in the status bar.

As soon as the cursor is moved by the user, the string written to the status line disappears. By this, the function will be used by the developer only in actions combined with the function **grread**. It makes no sense to use it for display prompts or messages.

The length of the string, which can be displayed depends on the video resolution configured for the system. If the string contains more words than the message box in the status line can display, the text is truncated at the last word it can display.

Example:

```
(grtext "Status bar message")
```

handent

The function **handent** allows to retrieve the Entity Name via its unique reference supplied by the argument *handle*.

(handent handle)

The function returns nil, if in the argument *handle* not a valid reference ID has been supplied to the function. For example, nil is returned, if a handle of an entity that has been deleted in the meantime is passed to the function.

Examples:

```
(handent "C5")  
<Ename: 3ed70f6a>
```

help

The function **help** calls WinHelp with a specified help topic.

(help [topic] [helpfile] [command])

Parameters:

topic: Keyword in help file (optional)

helpfile: Name of the HLP file (optional)
 command: Help action to be performed ("1" ..."14")

Return value:

The function returns nil, if *help* does not exist, otherwise it returns the name of help file.

Examples:

```
(help "Line")
(help "Example" "myhelp.hlp")
```

if

This function performs an conditional evaluation.

(if test_expression then_expression [else_expression])

The function returns the result of the last expression evaluated.

Examples:

```
(if Test 1 2)
2
(if (= x 1) (setq ret "Yes")(setq ret "No"))
"No"
(if (/= x 3) (progn (princ "Not equal 3!")(terpri)(setq x 3)))
Not equal 3!
3
```

See also:

progn

initget

The function initializes keywords and/or input-filters (valid data ranges).

(initget [flags] [keywords])

It determines keywords for the next user input requested by an application's function of type **getxxx** or **xentselx**.

The initialization is only valid for the next function requesting the user and afterwards invalid.

Parameters:

flags Control flag for input evaluation. This argument controls the behavior of the single input functions. The control bits have the following meaning:

Control Bit	Integer Value	Meaning

0	1	A RETURN or null input is rejected
1	2	A numerical value of zero is rejected
2	4	A negative numerical value is rejected
3 / 4 / 5	---	Not used
6	64	The Z coordinate of a 3D point value is ignored
7	128	Allow arbitrary input whatever the user enters
8	256	Allows to display keyword(s) in the context bar (options bar) of the desktop. As the user clicks one of the option buttons the corresponding keyword is returned by the function of type getxxxx.

keywords The optional argument *keywords* may contain a list of pre-defined keywords valid for the next execution of a user input function. The keywords are supplied as a string separated by one (or more) spaces.

The following table shows which control bit is applicable for which function requesting user input:

Function	No null input	No zero	No negative	No 3D coord.	Arbitrary input	Key-word
<i>Control bit</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>6</i>	<i>7</i>	<i>8</i>
<i>Bit value</i>	<i>1</i>	<i>2</i>	<i>4</i>	<i>64</i>	<i>128</i>	<i>256</i>
getangle	x	x			x	x
getcorner	x				x	x
getdist	x	x	x	x	x	x
getint	x	x	x		x	x
getkeyword	x				x	x
getorient	x	x			x	x
getpoint	x				x	x
getreal	x	x	x		x	x
getstring						
entsel						x
nentsel						x
nentselp						x

inters

The function **inters** computes the intersection of two lines in the current drawing space.

(inters from1 to1 from2 to2 [TestON])

The two lines are defined by four points passed to the function. The first line is defined by the points *from_pt1* and *to_pt1*, the other line by the points *from_pt2* and *to_pt2*. The points are evaluated as 3D points; the intersection is calculated in 3D space.

The function accepts an optional argument *flag*. If set, the function assumes the lines to be infinite and returns the intersection even if the lines do not cross (virtual intersection). Otherwise the intersection must lie on both lines (real intersection).

The function returns nil if no intersection point has been found.

Examples:

```
(inters '(1 0) '(2 0) '(1 1)'(0 0) T)
nil
```

```
(inters '(1 0) '(2 0) '(1 1)'(0 0) nil)
(0.00 0.00 0.00)
```

itoa

The function **itoa** returns the given *integer* as string.

(itoa integer)

If **itoa** not succeeds, it returns nil.

Examples:

```
(itoa 2)
"2"
```

```
(itoa -2)
"-2"
```

lambda

This function allows the definition of an anonymous function.

(lambda argument_list term1 ...)

Parameters:

argument_list a list including the arguments of the anonymous function
term1 ... terms to be evaluated

The function returns the result of the last expression evaluated.

Example:

```
(mapcar '(lambda (s) (strcase s)) ('"a" "b" "c" "d" "e" "f" "g"))
("A" "B" "C" "D" "E" "F" "G")
```

See also:

mapcar, apply

last

The function **last** returns the last element of the given argument *list*.

(last list)

Examples:

```
(last '(1 2 3 4 5))
```

```
5
```

```
(last '(1 2 3 (4 5)))
```

```
(4 5)
```

length

The function **length** returns number of elements in the given argument list.

(length list)

Examples:

```
(length '(1 2 3))
```

```
3
```

```
(length '(1 2 3 (4 5)))
```

```
4
```

```
(length abc)
```

```
0
```

list

The function **list** creates a list out of expressions supplied as arguments to the function. The function accepts a variable number of expressions.

(list expression1 ...)

Examples:

```
(setq x 3)
```

```
(setq y (list 1 2 x))
```

```
(1 2 3)
```

```
(setq y '(1 2 x))
```

```
(1 2 X)
```

listp

The function **listp** verifies if the given argument *element* is a list.

(listp element)

It returns T if *element* is a list, otherwise nil.

Examples:

```
(listp '(1 2 3 ))
```

```
T
```

```
(listp 1)
```

```
nil
```

```
(listp "abc")  
nil
```

load

The function **load** loads an existing lisp file *filename* from disk, thus providing the functions defined in that file.

(load filename [error])

Expressions within the file which are outside of "defuns" are evaluated immediately.

The optional argument *error_expression* is an expression that is evaluated if loading fails. It might be an error message or a function.

If the given file cannot be loaded, the function returns nil; otherwise the function returns the result of the last expression evaluated.

Examples:

```
(load "test.lsp")  
(load "c:\\mylisp\\test.lsp")  
(load "test.lsp" "Cannot load TEST.LSP !")  
(load "test.lsp" (errorfunc))
```

See also

System variable LSPALOAD

log

This function returns the natural logarithm of a given number

(log number)

Examples:

```
(log 2)  
0.6931  
(log 1)  
0.0000
```

logand

The function **logand** returns the result of a logical bitwise AND on the integer values supplied as arguments to the function.

(logand number1 number2....)

Examples:

```
(logand 9 1)  
1  
(logand 15 4 1)  
0  
(logand 15 2 6)
```

2

logior

The function **logior** returns the result of a logical bitwise OR on the integer values supplied as arguments to the function.

(logior number1 number2....)

Examples:

```
(logior 1 2 4)
```

```
7
```

```
(logior 1 3 )
```

```
3
```

lsh

The function **lsh** performs a logical bitwise shift on *integer* of *n* bits.

(lsh integer n_Bits)

Examples:

```
(lsh 1 1)
```

```
2
```

```
(lsh 1 -1)
```

```
0
```

```
(lsh 15 -1)
```

```
7
```

mapcar

The function **mapcar** operates on successive elements of the lists *list1* ... *list_n*.

(mapcar function liste1...)

First the *function* is applied to the *car* of each list, then to the *cadr* of each list, and so on. **mapcar** returns a list containing the *n* results of the successive calls to the function.

Note: The elements of the lists are not calculated. A list is returned!

Examples:

```
(mapcar '(lambda (a b) (+ a b)) '(10 20 30 40) '(1 2 3 4) )
```

```
(11 22 33 44)
```

```
(mapcar '* '(1 2 3 4) '(1 2 3 4))
```

```
(1 4 9 16)
```

max

The function **max** returns the largest number of the numbers given as arguments to the function.

(max number1 number2 ...)

The function returns a real if any one of the given arguments is of type real. It returns an integer if all arguments are integer.

Examples:

```
(max 2 3 1)
```

```
3
```

```
(max 2 3.1 1)
```

```
3.1000
```

mem

This function displays the memory status of FLISP and returns nil.

(mem)**Examples:**

```
(mem)
```

```
WorkFLisp: iStatus: 0 iFreeNodes: 231 iParseEbene: 0
```

```
iNodes : 1028 iSegments: 0 iAllocate: 0
```

```
nil
```

member

The function **member** searches the given *list* for an *element* and returns the remaining portion of the *list*.

(member element list)

The function returns nil, if *element* is not a member of *list*.

Examples:

```
(member '1 '(1 2 3 4))
```

```
(1 2 3 4)
```

```
(member 'c '(a b c a b c))
```

```
(C A B C)
```

```
(member '1 '(a b c))
```

```
nil
```

min

The function **min** returns the smallest number of the numbers given as arguments to the function.

(min number1 number2 ...)

The function returns a real if any one of the given arguments is of type real. It returns an integer if all arguments are integer.

Examples:

```
(min 2 3 1)
```

```
1
```

```
(min 2 3.1 1)
```

```
1.0000
```

minusp

The function **minusp** tests if an real or an integer is negative.

(minusp *number*)

The function returns T if the argument supplied is negative or evaluates to a negative value.

Examples:

```
(minusp -1.33)
```

```
T
```

```
(minusp 5)
```

```
nil
```

```
(minusp 0)
```

```
nil
```

nentsel

The function **nentsel** prompts the user to select a drawing object and returns a list containing the entity name of the drawing object and the pick point coordinates (of the current UCS). In difference to the function **entsel**, it allows to retrieve the information of a complex object.

(nentsel [*prompt*])

prompt: If the optional argument *prompt* contains a valid string, it is displayed at the command line, otherwise the standard prompt of the FCAD Engine (“Select object:”) is displayed.

If the selected entity is contained in a complex object the function returns in addition:

- the resulting transformation matrix to allow transformation of points from the Entity Coordinate System (EKS) to the World Coordinate System (WCS); and
- a list containing the Entity Name of the superior object

Examples:

```
(nentsel)
```

```
(<Ename: 4047a928> (2918.00 2351.00 0.00))
```

```
(nentsel "Please pick a part:")
```

```
Please pick a part:
```

```
(<Ename: 00030000> (4303.50 2288.00 0.00) ((1.00 0.00 0.00 0.00) (0.00
```

```
1.00 0.00 0.00) (0.00 0.00 1.00 0.00) (4471.50 2309.00 0.00 1.00))
```

```
(<Ename: 404797ec>))
```

See also:

nentselp, entsel

nentselp

The function **nentselp** allows to select a drawing object and returns the defining data of the selected object. In difference to the function **entsel**, but identical to **nentsel**, it allows to retrieve the information of a complex object.

nentselp [*prompt*] [*point*]

In difference to **nentsel** it allows to pass a pick *point* to the function and prompting the user may be suppressed.

Examples:

```
(nentselp '(1 1))
(<Name: 4047a928> (2813.00 2393.00 0.00))
```

See also

nentsel, **entsel**

not

The function **not** returns T if *argument* is nil, otherwise returns nil.

(not arguments)**Examples:**

```
(not 12)
nil
(not "abc")
nil
(not nil)
T
```

nth

The function **nth** returns the *n*th element of *list*.

(nth n list)

Note: *n* counting starts with 0.

Examples:

```
(nth 1 '(0 1 2 3))
1
(nth 2 '(a b c))
C
(nth 5 '(a b c))
nil
```

null

The function **null** returns T if *argument* is (), otherwise returns nil.

(null argument)**Examples:**

```
(setq x 32)
32
(setq x nil)
```

```
nil
(null x)
nil
(null y)
T
```

numberp

The function **numberp** returns `T` if its *argument* is any kind of number, otherwise returns `nil`.

(numberp argument)

Examples:

```
(numberp pi)
T
(numberp 'pi)
nil
(numberp 5)
T
(numberp "0.1")
nil
```

open

This function opens a file given by *filename* to read or write.

(open filename io_mode)

The argument *io_mode* specifies :

IO_mode	Meaning
"r"	Opens for reading. If the file does not exist or cannot be found, it returns <code>nil</code> .
"w"	Opens an empty file for writing. If the given file exists, its contents are destroyed.
"a"	Opens for writing at the end of the file (appending); creates the file first if it doesn't exist.

Return value:

The function returns a file descriptor to be used by other I/O functions.

Examples:

```
(setq d (open "test.txt" "w"))
(setq d (open "test.txt" "r"))
(setq d (open "test.txt" "a"))
```

or

The function **or** returns the logical OR of multiple terms in a list.

(or value1...)

If at least one term is bound (not nil) it returns T, otherwise nil.

As soon as one of the terms evaluated is T the function returns nil.

Examples:

```
(setq a nil)
```

```
(or a 45 "abc")
```

```
T
```

```
(or a nil)
```

```
nil
```

osnap

The function **osnap** returns a 3D point as a result of applying the object snap mode(s) specified by the argument *mode* closest to the search point provided in the argument *point*.

(osnap point mode)

The string argument *mode* may contain one or multiple keywords. If multiple keywords are supplied these have to be separated by comma(s).

Supported values are:

Keyword	Snap mode
_cen	Center
_end	End point
_ins	Insert
_int	Intersection
_mid	Mid point
_nea	Nearest
_nod	Node
_per	Perpendicular
_qua	Quadrant

<code>_tan</code>	Tangent
-------------------	---------

Note: The system variable `SELECTBOX` defines the size of a box in pixels for object selection.

Examples:

```
(setq x (osnap '(1 1) "_endp"))
```

```
(setq x (osnap '(50 50) "_cen,_int"))
```

pi

This is the constant π approximated to 3.14159265.

polar

This function computes a point via polar coordinates.

(polar *point1* *distance* *angle*)

The function returns the polar point in the *distance* and in the *angle* from *point1*. The angle is returned in radians, seen counter-clockwise.

Examples:

```
(polar '(0 0 0) 0.7854 1.4142)
```

```
(1.0000 1.0000 0.0000)
```

prin1

The function **prin1** outputs the printed representation of *expression* at the command line or writes it to the open file specified by the optional argument *file_descriptor*.

(prin1 [*expression* [*file_descriptor*]])

Only the specified *expression* is printed; no newline is included.

Return value:

The function returns *expression*.

Examples:

```
(prin1 "Test")
```

```
"Test""Test"
```

```
(prin1 "\123")
```

```
"S""S"
```

```
(setq fd (open "test.txt" "w"))
```

```
<File: #3407:4624>
```

```
(prin1 "Text \n" fd)
```

```
"Text \n"
```

princ

The function **princ** prints *expression* on the command line or writes it to the open file specified by the optional argument *file_descriptor*.

(princ [*expression* [*file_descriptor*]])

Return value:

The function returns *expression*.

print

The function **print** prints *expression* on the command line or writes it to the open file specified by the optional argument *file_descriptor*.

(print [*expression* [*file_descriptor*])

This function is the same as `prin1`, but with a preceding Carriage Return and a Space following.

progn

The function **progn** evaluates one expression after the other grouped in **progn**.

(progn *term1* ...)

It returns the value of the last expression of the **progn** group.

Examples:

```
(if test
  (progn (princ test) (setq c (+ a b)) )
)
```

prompt

The function displays *message* at the command line.

(prompt *message*)

The function returns nil.

Examples:

```
(prompt "Please enter your name: ")
```

quit

The function **quit** terminates (cancels) the current application and returns to the command prompt.

(quit)

The following error message is displayed:

Warning: Program terminated by QUIT

See also:

exit

quote

The function simply returns *object*. (Term is not evaluated)

(quote *term*)

Examples:

```
(setq x (quote (1 2)))
```

```
(1 2)
```

```
(setq x '(1 2))
```

(1 2)

read

The function **read** retrieves the first atom or list from the given string and returns it according to its data type.

(read *string*)

The function returns the first letters preceding a space (in capital letters) if the argument is a text-string.

Examples:

```
(read "one two three")
```

```
ONE
```

```
(read "(1 2 3)")
```

```
(1 2 3)
```

```
(read "123 456")
```

```
123
```

read-char

The function **read-char** reads a single character from either the keyboard buffer or from an open file optionally specified by *file_descriptor*. It returns the ASCII code as integer.

(read-char [*file_descriptor*])

Examples:

```
(read-char)
```

```
(setq fd (open "test.txt" "r"))
```

```
(read-char fd)
```

read-line

The function **read-line** reads a string from the keyboard buffer or from an open file optionally specified by *file_descriptor*. The function returns a string.

(read-line [*file_descriptor*])

Examples:

```
(read-line)
```

```
(setq fd (open "test.txt" "r"))
```

```
(read-line fd)
```

redraw

The function redraws a specified drawing entity or refreshes the entire current viewports.

(redraw [*element* [*mode*])

If **redraw** is entered without arguments, a REDRAW is performed on the entire current viewport.

Parameters:

- element: Entity name of the drawing object to be redrawn.
- element: Redraw Mode. If *element* contains a valid entity name, the drawing object is re-displayed in one of the modes specified by the *mode* argument as follows:

Mode	Action executed
1	Redraw entity
2	Entity set invisible
3	Entity is highlighted
4	Removes highlighting of the entity

The function returns nil.

regapp

The function **regapp** registers the application name supplied in *applic_name*. A registered application name is required for manipulation of Application Entity Data (EED).

(regapp *applic_name*)

An application name may contain up to 31 characters.

If the function succeeds, it returns *applic_name*, otherwise nil.

rem

The function **rem** divides number1 by number2 and returns the remainder.

(rem *number1 number2*)

Examples:

```
(rem 10 3)
```

```
1
```

```
(rem 9 3)
```

```
0
```

repeat

The function **repeat** executes a given expression *n* times. *n* must be a positive number.

(repeat *n expression1 ...*)

The argument *n* must be a positive number.

Examples:

```
(repeat 5 (princ "Hello")(terpri))
```

```
Hello
```

```
Hello  
Hello  
Hello  
Hello  
nil
```

reverse

The function **reverse** reverses the given *list* and returns a new list.
(reverse list)

Examples:

```
(reverse '(1 2 3))  
(3 2 1)
```

rtos

The function **rtos** converts a floating point *value* to a *string*, depending on the specifications supplied in the arguments *format* for the unit conversion mode and *precision* for the number of decimal points.

(rtos value [format [precision]])

The value given in *format* is interpreted as follows:

Value	Format
-1	The current value of the local system variable LUNITS (Linear Units) is used, which may contain one of the following settings (1 to 5).
1	Scientific representation (1.235E2)
2	Decimal (123.50)
3	English: Engineering (feet/inches) (10'-3.50")
4	English: Architectural (feet/fractional inches) (10'-3 1/4")
5	English: Fractional (123 1/4")

The function **distof** corresponds to the function **rtos**.

A string returned by the function **rtos** can be re-converted error-free by the function **distof** (on the premise that the unit mode used for string formatting is the same) and vice versa.

set

The function **set** sets the value of a quoted symbol to an expression and returns that value. The first argument must be a symbol.

(set symbol expression)

Examples:

```
(set 'x 'y)  
Y  
(set x 5)  
5
```

!y
5**setactvport**

The function **setactvport** allows to set another drawing as current drawing or to set another viewport as active by determining the ID of a currently opened drawing and the ID of the drawing's viewport to be set.

(setactvport *nDbNo nVpNo*)

Parameters:

nDbNo Database ID number of a currently opened drawing.
(integer value between 0 and 3)

nVpNo ID of a opened viewport of the current drawing to be set as active.

Return value:

If the function succeeds, it returns the given arguments in a list, otherwise nil.

Example:

(setactvport 1 1)

See also:

getactvport

setfunhelp

The function **setfunhelp** sets a help topic belonging to a function.

(setfunhelp *function topic file*)

Parameters:

function a string containing the name of a defined function ("C:NAME")

topic a string containing the help topic

file a string containing the name of the help file (optional)

If the function succeeds, it returns the argument *function*, otherwise nil.

Examples:

(setfunhelp "C:MYFUNC" "MYFUNC_TOPIC" "TEST.HLP")
"MYFUNC"

setq

The function **setq** sets the value of a symbol to an expression and returns that value.

(setq *symbol1 value1 [symbol2 value2] ...*)

Multiple paired *symbol expression* arguments may be supplied. In this case the function returns the value of the last assignment.

Examples:

```
(setq a 35)
(setq b "abc")
(setq c '(1 2 3))
```

setvar

The function **setvar** sets a local or global system variable *sysvar* to the *value* supplied as argument.

(setvar *sysvar value*)

The function returns the new value of *sysvar*.

Examples:

```
(setvar "CMDECHO" 1)
```

See also:

getvar

sin

This function returns the sine of an angle.

(sin *angle*)

The argument *angle* must be expressed in radians.

Examples:

```
(sin pi)
```

```
0.0000
```

```
(sin 1.57)
```

```
1.0000
```

sqrt

This function returns the square root of a given positive *number*.

(sqrt *number*)

Examples:

```
(sqrt 9)
```

```
3.0000
```

```
(sqrt 5)
```

```
2.2361
```

ssadd

The function **ssadd** adds an valid entity specified by *ename* to an existing selection set specified by *sname* or creates a new selection set. The function returns a new selection set.

(ssadd [*ename* [*sname*]])

The entity specified by *ename* and the selection set *sname* must belong to the current drawing.

Entering **ssadd** with no argument returns a new, empty selection set.

Entering **ssadd** with a valid entity name, but without the argument *sname*, a new selection set is created, containing the single entity *ename*.

As in *ename* a valid entity name and in *sname* a valid selection set name is provided, the entity *ename* is added to selection set.

If the function fails, it returns nil.

Examples:

```
(setq as (ssadd))
<Selset: 3d671828:00000006>
(ssadd (entlast) as)
<Selset: 3d671828:00000006>
```

ssdbno

The function **ssdbno** returns the database ID (integer between 0 3) to verify for which drawing the specified Selection Set Name *sname* is valid.

(ssdbno *sname*)

The selection set *sname* must belong to the current drawing.

If the function fails, it returns nil.

Examples:

```
(setq as (ssadd (entlast)))
<Selset: 4237b048:00000004>
(ssdbno as)
0
```

ssdel

The function **ssdel** removes (deletes) an valid entity specified by *ename* from the selection set *sname*.

(ssdel *ename sname*)

The entity specified by *ename* and the selection set *sname* must belong to the current drawing.

If the function succeeds, it returns the name of the selection set, otherwise nil.

Examples:

```
(ssdel (entlast) as)
```

ssget

The function **ssget** creates a new selection set.

(ssget [*mode*] [*P1*] [*P2*] [*point_list*] [*filter*])

Parameters:

mode Selection mode. In the argument *mode* is specified the method to be applied for object selection. *mode* may have one of the values as follows:

Mode	Meaning
------	---------

-	The inter-active selection mode is started prompting the user to select objects.
"_C"	Crossing
"_CP"	Crossing Polygon
"_F"	Fence
"_L"	Last entity
"_P"	Previous selection set
"_W"	Window
"_WP"	Window Polygon
"_X"	Filtered Selection

P1/P2 *P1 /P2* may point either to a drawing point or to a list of drawing points.
 The variable *mode* controls the appropriate interpretation.
 In the case that the selection modes "_C" and "_W" are specified, *P1* and *P2* contain the coordinates required for the selection window.

If "_CP", "_WP", or "_F" are used, *P1* must point to a list of drawing points, which form the polygon to select the drawing objects; in this case *P2* is ignored.

filter Association list. Filter containing additional selection criteria. For example, you can specify certain entity types, entity properties, coordinate areas, etc. in the filter list. Geometric as well as non-geometric criteria allow flexible selection methods.

point_list List of points defining a polygon or fence.

Relational Tests

The special group code -4 allows numerical comparison. Any relational test must be invoked with this code. The value of the group code is a string specifying the operator.

Logical Operators

Logical operators are also specified via a group code -4. In this case the group code is used to start and to terminate an expression for logical comparison. Starting the operation is indicated by a '<' sign before the operator keyword (e.g. "<OR") as first expression in a list containing the contents of the logical comparison. Termination of the comparison expression is signaled by a '>' behind the operator keyword (e.g. "OR>"). The following possibilities, which also may be used nested, are at your disposal:

```
...((-4 . "<AND") [expression1] ...(-4 . "AND>"))
...((-4 . "<OR") [expression1] ...(-4 . "OR>"))
...((-4 . "<XOR") [expression1] ...(-4 . "XOR>"))
...((-4 . "<NOT") [expression] (-4 . "NOT>"))
```

If the function fails, it returns *nil*.

Examples:

```
(ssget)
(ssget '(10 10))
(ssget "_X")
(ssget "_P")
(ssget "_L")
(ssget "_W" '(0 0) '(100 100))
(ssget "_WP" '((0 0) (0 100) (100 0)))
(ssget "_F" '((0 0) (15 15) (15 0)))
(ssget "X" '((0 . "LINE"))) ; selects all lines
(ssget "X" '((0 . "CIRCLE") (8 "0")))
(ssget "_C" '(0 0) '(10 10) '((0 . "CIRCLE")))
; all circles crossing the window 0,0 10,10
(ssget "X" '(
  (-4 . "<OR")
  (-4 . "<AND")(0 . "TEXT")(40 . 10)(-4 . "AND>")
  (0 . "CIRCLE")
  (-4 . "OR>"))))
(ssget "X" '((0 . "TEXT") (-4 "=") (40 . 10)))
```

sslength

The function **sslength** retrieves the number of entities in a given selection set *sname*.

(sslength sname)

Example:

```
(setq as1 (ssadd (entlast)))
<Sset: 4b1f1360:00000030>
(sslength as1)
1
```

ssmemb

The function **ssmemb** allows to test, if a selection set *sname* contains a drawing object specified by *ename*.

(ssmemb *ename sname*)

The selection set specified by *sname* and the entity specified by *ename* must belong to the current drawing. The function is drawing-sensitive.

The function returns *ename*, if *ename* is member of *sname*; otherwise nil.

Examples:

```
(setq as1 (ssadd (entnext)))  
<Sset: 4b1f18cc:00000044>  
(ssmemb (entnext) as1)  
<Ename: 4b1f0f28>  
(ssmemb (entlast) as1)  
nil
```

ssname

The function **ssname** allows to retrieve the entity name contained in a given selection set of the current drawing via its index *n* in the selection set.

(ssname *sname n*)

The argument *n* may contain the range of 0 (first entity) to (sslength(*sname*)-1), which is the last entity in the selection set.

The function returns the *n*-th element of the selection set *sname* in *ename*.

For example, the function allows to evaluate a selection set created by **ssget** element-wise.

Examples:

```
(ssname as 0)  
<Ename: 4237b028>
```

strcase

The function **strcase** converts a given *string* to a new upper case (*mode* = nil (or not supplied) or lower case (*mode* /= nil) string.

(strcase *string [mode]*)

Examples:

```
(strcase "Abcde")  
"ABCDE"  
(strcase "Abcde" T)  
"abcde"
```

strcat

The function **strcat** returns a new string concatenating two or more strings.

(strcat *string1 [string2] ...*)

Example:

```
(strcat "Jim" "Meyer")
"JimMeyer"
```

stringsort

This function sorts a list of string-items alphabetically, ascending.

(stringsort *list*)

If at least one element of the list is not a string, nil is returned.

Examples:

```
(stringsort '("hello" "abc" "xyz" "Hello" "123"))
("123" "Hello" "abc" "hello" "xyz")
```

strlen

The function **strlen** returns the sum of the length (number of characters) of all strings given as argument to the function.

(strlen [*string1*] ...)**Examples:**

```
(strlen "123")
3
(strlen "123" "456" "789")
9
(strlen)
0
(strlen "")
0
```

subst

The function **subst** copies a list substituting every occurrence of *old_element* by *new_element*.

(subst *new_element old_element list*)

If *old_element* cannot be found in the given list, **subst** returns the original list unchanged.

Examples:

```
(subst '1 '4 '(4 2 3 4 2 3 4 2 3))
(1 2 3 1 2 3 1 2 3)
(subst '1 '4 '(1 2 3 1 2 3 1 2 3))
(1 2 3 1 2 3 1 2 3)
```

substr

The function **substr** returns a new - partial - string based on a given *string* supplied as argument starting at *start* and ending at *start+length+1*.

(substr *string start [length]*)

Starting number of the first character is 1.

Examples:

```
(substr "1234567890" 3 6)
"345678"
```

```
(substr "1234567890" 3)
"34567890"
```

symbtos

The function **symbtos** returns any lisp expression in a string.

(symbtos *symbol*)

Examples:

```
(strcat "Last entity name: " (symbtos (entlast)))
" Last entity name: <Ename: 484f1410>"
```

tbldel

The function **tbldel** deletes in a specified table a record specified by the name of the item, but only if the table entry is not referenced in the drawing.

(tbldel *table_name record*)

Valid names of table types are:

BLOCK, DIMSTYLE, LAYER, LTYPE, STYLE, UCS and View.

If the function succeeds, it returns the name of the deleted record, otherwise nil.

Example:

```
(tbldel "LAYER" "TEST")
```

See also:

tblpurge

tblmake

The function **tblmake** generates a new table-entry described in an association list *table_list*.

(tblmake *table_list*)

Before creating a new table record, the function **tblmake** tests the validity of the contents of the supplied group codes for the corresponding table type.

Valid names of table types are:

BLOCK, DIMSTYLE, LAYER, LTYPE, STYLE, UCS, VIEW, and VPORT.

The function **tblmake** returns nil and creates no table entry, if one or more group codes required for a complete table entry is missing.

- The type of the table to be enhanced with a new record (Layer, Linetype, UCS, etc.) must be specified with the group code 0 in the association list.
- The name of the table entry to be created must be supplied with the group code 2 in the association list.

Examples:

```
(setq x (list
  (cons 0 "VPORT")      ; name of the table
  (cons 2 "*ACTIVE")
  (cons 12 (list 1 1 1)) ; center point
  (cons 40 10)         ; height
))
(tblmake x)

(setq y (list
  (cons 0 "LAYER")      ; name of the table
  (cons 2 "ABC")        ; name of the new layer
))
(tblmake y)
```

Return value:

If the function succeeds, it returns the association list, otherwise nil.

tblmod

The function **tblmod** modifies an table-entry by a new *table_list*.

(tblmod table_list)*Parameters:*

table_list Association list containing the information of the table type and table item to be modified. The association list has the same structure as an association list returned by the functions **tblnext** and **tblsearch**.

Valid names of table types are:

BLOCK, DIMSTYLE, LAYER, LTYPE, STYLE, CS, VIEW, and VPORT.

Before modifying an existing table entry, the function **tblmod** tests the validity of the contents of the supplied group codes for the corresponding table type and table item name.

The function **tblmod** returns nil and modifies no table entry, if one or more group codes required for a complete table record modification is missing.

- The type of the table to be enhanced with a new record (Layer, Linetype, UCS, etc.) must be specified with the group code 0 in the result buffer.
- The name of the table entry to be created must be supplied with the group code 2 of the result buffer list.

If the modification of an table entry succeeds, the drawing is redrawn automatically if necessary.

Example:

```
; change color of layer 0 (Code 62)
(tblmod (list (cons 0 "LAYER")(cons 2 "0")(cons 62 1)))
```

or :

```
(setq x (tblnext "LAYER" T))  
(setq x (subst (cons 62 1)(assoc 62 x) x))  
(tblmod x)
```

tblnext

The function returns the first or the next record of a table specified by *table_name*.

(tblnext *table_name* [*first*])

first: If *first* is 0 the first entry of the table is returned. If *first* is 1 the next entry (following the current pointer position) is returned.

If no (or no more) table-entry is found, the function returns nil.

Example:

```
(tblnext "LAYER" T)  
((0 . "LAYER") (2 . "0") (70 . 64) (62 . 7) (6 . "CONTINUOUS"))
```

tblpurge

The function purges an entire table of the specified type and deletes all not referenced entries.

(tblpurge *table_name* *flag*)

Valid names of table types are:

BLOCK, DIMSTYLE, LAYER, LTYPE, STYLE, UCS and View.

If the argument *flag* is not nil, a list of purged table entries is displayed in the command line area.

Example:

```
(tblpurge "LAYER" T)
```

See also:

tbldel

tblrename

The function allows the renaming of a table entry (table item). In *table* the entry *old_name* is replaced by the entry *new_name*.

(tblrename *table* *old_name* *new_name*)

Parameters:

table Table type

Valid names of table types are:

BLOCK, DIMSTYLE, LAYER, LTYPE, STYLE, UCS and VIEW.

old_name: Name of the table item to be renamed (current entry).

new_name New name for the table item.

Renaming entries in the tables BLOCK, DIMSTYLE, or STYLE cause that the entire drawing database is updated and that the drawing is redrawn. Records of tables of type APPID and VPORT can not be renamed.

Example:

(tblrename "LAYER" "ABC" "XYZ")

tblsearch

The function **tblsearch** searches the entry specified *item_name* in the table specified by *table_type* and returns the table item information.

(tblsearch table_type item_name)

Parameters:

table_type Table type

Valid names of table types are:

APPID, BLOCK, DIMSTYLE, LAYER, LTYPE, STYLE, UCS, VIEW and VPORT.

item_name Name of the table entry to be searched.

The table pointer is positioned to the table record found. This position is at the next call of **tblnext** the base position of the table pointer to the next position.

If the function succeeds, it returns the table-entry described in an association list, otherwise it returns nil.

Example:

(tblsearch "LAYER" "0")

((0 . "LAYER") (2 . "0") (70 . 0) (62 . 1) (6 . "CONTINUOUS"))

tblset

The function **tblset** sets a table entry as current entry. In the table specified by *table_type* the entry *item_name* is set as current (default / active). A associated system variable is updated.

(tblset table_type item_name)

Parameters:

table_type Table type

Valid names of table types are:

DIMSTYLE, LAYER, LTYPE, STYLE and UCS

item_name Name of the table entry to be set as current.

Table entries of tables of type APPID, BLOCK, VIEW, and VPORT can not be set current by this function.

Return value:

If the function succeeds, it returns *item_name*, otherwise it returns nil.

terpri

The function **terpri** outputs a newline at the command line and always returns nil.

(terpri)

textbox

The function **textbox** allows the computation of the bounding box of a virtual text or an existing text object in the current drawing.

(textbox association_list)

The argument to the function is an entity list describing a text.

The function returns the bounding box corners of the text.

Examples:

```
(textbox '((0 . "TEXT")(1 . "abc")(7 . "STANDARD")(40 . 0.2)))  
((0.00 0.00 0.00) (0.48 0.13 0.00))
```

textscr

The function **textscr** switches the input focus to the Lisp/History text-window.
(textscr)

The *History Command / Lisp Interpreter Window* (also called text window) is meant especially for the input of multi-line LISP expressions, which - as evaluated successfully - may be transferred via the Windows Clipboard to text files to write Lisp routines.

Also, in this window allows to survey the commands history (History / Commands-Review). Using standard shortcuts allow to copy commands or LISP expressions to the Clipboard (Clipboard-Copy) and paste these strings or expressions at the command line (Clipboard-Paste).

System commands of FelixCAD do not use this window: No built-in command switches to the text window. We recommend for application development to maintain this style of user interface. The dialog functions allow a much more comfortable possibilities to present the user listed information.

See also:

graphscr

trans

The function **trans** allows the transformation of a point between different coordinate systems.

(trans point origin_cs target_cs [disp_mode])

The function transforms the coordinates supplied with the argument *point* and returns the resulting coordinates. The origin coordinate system and the target coordinate system are specified by the arguments *origin_cs* and *target_cs*.

The parameter *disp_mode* is used to control the contemplation of point (point or direction).

The arguments *origin_cs* and *target_cs* are evaluated as follows:

Argument is...	Description
0	Specifies the coordinate system to be applied. World Coordinate System (WCS)
1	Current User Coordinate System (UCS)
2	Current View Coordinate System (VCS)
Entity name	Entity coordinate system (ECS) of the drawing entity
3D point	Extrusion vector

Return value:

The function returns the result of the transformation.

Examples:

```
(trans '(0 0 0) 0 1)
```

```
(trans '(0 0 0) 1 (entlast) 1)
```

type

The function **type** returns the data type of *object* as symbol

(type object)

The function returns one of the symbols as follows:

ENAME	Entity Name
EXSUBR	External GDE-Lisp Function
FILE	File Descriptor
INT	Integer
LIST	List (or function)
PICKSET	Selection Set
REAL	Floating point number
STR	String
SUBR	Internal GDE-Lisp Function
SYM	Symbol

Examples:

```
(type 1.234)
```

REAL
(type a)
nil
(type 'a)
SYM
(type setq)
SUBR

ver

The function **ver** returns a string containing the information on the current version of the FLISP interpreter.

(ver)

Example:

(ver)
"FLISP Version 1.00"

wcmatch

The function **wcmatch** allows a wild card pattern match search on a *string* supplied to the function.

(wcmatch string pattern)

Parameters:

pattern Wild card pattern match to be performed on *string*. The pattern may contain the wild card characters as documented in the following table:

Placeholder	Meaning
#	Single numeric digit
@	Single alphabetical character
.	Single non-alphabetical character
*	Series of characters, including ""
?	Any single character
~	If ~ (tilde) is the first character in the pattern, the function matches anything except the pattern
[...]	The function matches any one of the characters enclosed
[~...]	The function matches any single character not enclosed
-	Used inside brackets to specify a range for a single character

'	Separator for two patterns
'	Escapes special characters, reads next character literally.

If a match is found, the function returns T; otherwise it returns nil.

Examples:

```
(wcmatch "a1" "a*")
T
(wcmatch "a1" "a@")
nil
(wcmatch "a1" "a.")
T
(wcmatch "a1" "a[1-12]")
T
(wcmatch "a1" "a[-1]")
nil
(wcmatch "a1" "a3,a2,a1")
T
(wcmatch "a1" "a?")
T
```

while

The while construct allows iteration to continue until the specified expression evaluates to nil.

(while test_expression expression1 ...)

The function returns the result of the last expression evaluated.

Example:

```
(setq x 0)
(while (< x 5) (princ "X=")(princ x)(terpri) (setq x (+ x 1)))
X=0
X=1
X=2
X=3
X=4
5
```

write-char

The function **write-char** writes a single character to a file specified by *file_descriptor* or to the command line.

(write-char integer [file_descriptor])

The single character is specified by an ASCII code (given as integer).

The function returns an *integer* value.

Examples:

```
(write-char 65)
A65                ; 65 is the return value
(write-char (ascii "A"))
A65
(setq fd (open "abc.txt" "w"))
<File: #4717:4624>
(write-char 65 fd)
65
```

write-line

The function **write-line** writes a string to file specified by *file_descriptor* or to the command line.

(write-line string [file_descriptor])

The function returns *string*.

Examples:

```
(write-line "ABC")
"ABC"
(setq fd (open "abc.txt" "w"))
<File: #4717:4624>
(write-line "ABC" fd)
"ABC"
```

xload

The function **xload** loads an "external" FDT application's DLL specified by *dll_filename*.

(xload dll_filename [error])

FDT applications are a special DLL's. The functions contained in the DLL are automatically registered and then available for the user.

The optional argument *error* is an expression that is evaluated if loading fails.

It might be an error message or a function.

If successful the function returns the complete DLL filename including drive and path, otherwise nil.

Examples:

```
(xload "test")
(xload "c:\mydll\test.dll")
(xload "test" "Cannot load test.dll")
(xload "test" (errorfunc))
```

xunload

The function **xunload** de-activates an "external" FDT application.

(xunload dll_filename)

The function searches for the DLL filename *dll_filename* and de-activates (unloads) all functions of that application DLL if the file search has been successful.

Example:

```
(xunload "test")
```

zerop

The function **zerop** returns T if *number* is zero (integer zero or floating-point zero).

(zerop *number*)

Examples:

```
(setq x 0)
```

```
0
```

```
(zerop x)
```

```
T
```

```
(zerop 0.000)
```

```
T
```

```
(zerop 1)
```

```
nil
```

CHAPTER 11

Programming Dialog Boxes

Often the first step to creating an application based on the FelixCAD is to create the interface: the menu, the palettes (tool boxes), and the dialog boxes the user will see and use. Then you write the program code to make the interface active.

The Dialog and Menu Editor (DME) and a set of FLISP functions to control dialogs provide advanced tools to enhance your CAD application with Windows typical communication via dialogs. The dialogs, you supply with your application commands will have the Windows look and feel and typical dialog control elements, the user is already familiar with.

You use dialogs to get user input and to display output. Dialog boxes are in most cases the best way to request complex user input, provide the user several choices, or present extensive facts.

The first part of this chapter introduces you step by step, how to load, display, fill and evaluate your dialogs and the contained controls. Examples are provided to demonstrate program code for dialog control.

The reference of all dialog functions of the FLISP interface, in alphabetical order, is found in the second part of this chapter.

Design of a Dialog

Using FelixCAD's Dialog- and Menu Editor (DME) you lay out size and properties of **dialogs** and arrange the **control elements** like buttons, list boxes, check boxes in the dialog window and specify the **properties** of each control element.

In a dialog file created with DME multiple dialogs may be stored.

An important point to remember when creating a dialog is, that you specify for each dialog and each dialog element a unique **Control Name** (also called key). Control names are identifiers allowing the application programs, either Lisp or C, to access the to the dialog and to its control elements.

Maintaining unique keys makes sure that the dialog functions provided by the programming interfaces are able to perform and evaluate dialog operations properly.

Caution:

Note, that the specification of control name is case sensitive.

[Additional Information on Dialog Creation and Design](#)

Further detailed information on creating and designing dialog boxes are provided in Chapter 9 *Creating and Editing Dialog Boxes* of the Programmer's Manual.

Loading and Displaying a Dialog

After having created dialogs with the Dialog Editor and saved the dialog files, the programs can be written which load and display the dialog boxes.

Principally a program's scheme to load and display a dialog is as follows:

File	Meaning
Dlg_DialogLoad	Loading a dialog file
Dlg_DialogNew	Loading a dialog
Dlg_DialogStart	Initializing and starting a dialog
Dlg_DialogDone	Terminate dialog
Dlg_DialogUnload	Unload a dialog

Example:

For the examples it is assumed that a dialog with the name "Hello" has been saved in the file *test.dlg*. The following program code demonstrates how to load and display this dialog box:

```
(setq dlg_id (Dlg_DialogLoad "test"))  
(Dlg_DialogNew "Hello" dlg_id)  
(Dlg_DialogStart)
```

The dialog box "Hello" will be displayed if all functions could have been executed successfully.

Please note, that the function **Dlg_DialogLoad** returns an dialog ID, which must be supplied to the function **Dlg_DialogNew** together with the dialog name. The dialog ID is also required as argument for the function **Dlg_DialogUnload** to unload the dialog.

Initializing Dialog Controls

The previous example allowed to display a dialog box as it has been created with the aid of the Dialog Editor. However, neither dialog control elements of the dialog have been filled with values nor could the program react to user input.

The following description shows how to initialize a dialog box and how to fill dialog controls with values.

To fill dialog controls or to set modes of dialog controls, to react to events caused by user input in the dialog, and to evaluate the dialog control settings when the dialog has been terminated by the user, the developer of a program that utilizes a dialog must pass as argument to a function **Dlg_DialogStart** the name of a function that contains all these settings and actions.

Example:

Our modified example (compare to the one above) now looks like this:

```
(setq dlg_id (Dlg_DialogLoad "test"))  
(Dlg_DialogNew "Hello" dlg_id)  
(Dlg_DialogStart "(init_function)")
```

(Dlg_DialogUnload dlg_id)

Basic Functions to initialize a Dialog

There are two basic functions to assign a new value to dialog controls and to set the display mode of controls:

The function: **(Dlg_TileSet *key value*)**

sets a new *value* for the dialog control specified by the Control Name *key*.

The function: **(Dlg_TileMode *key mode*)**

sets a new display *mode* for the dialog control specified by the Control Name *key*.

A detailed description of the meaning of the display modes is found in the functions reference (see below).

Special Functions to initialize a Dialog

A set of specific functions is provided for the following control elements used in dialogs:

- List Boxes and Combo Boxes
- Image Boxes
- Sliders

For these control elements the function **Dlg_TileSet** to set a new value for a control is not sufficient. How to initialize these dialog controls is discussed next.

List and Combo Boxes

Operations to fill a list box control must be introduced with the function **(Dlg_ListStart *key operation index*)**.

Using multiple calls of the function **(Dlg_ListAdd *string*)** allows to fill the list with strings.

The function **Dlg_ListEnd** must be called to terminate the operations on the list.

Example sequence:

```
(Dlg_ListStart "list1")
(Dlg_ListAdd "My line no. 1")
(Dlg_ListAdd "My line no. 2")
(Dlg_ListEnd)
```

In this example a list box is filled with two entries, each a separate line in the list. The function **Dlg_ListStart** accepts additional parameters to execute further operations and manipulations on list boxes.

Please consult the description to the function **Dlg_ListStart** in the functions reference (see below) for detailed information on the parameters which, for example, allow to modify or to delete a specific line in a list.

Other functions in conjunction with list boxes are.

- Setting Tab Stops in a list box:
(Dlg_ListSetTabstops *key values*)

Example:

(Dlg_ListSetTabstops "list1" "20 40 60")

- Setting the column width in a multi-column list box:
(Dlg_ListSetColumnWidth *key width*)

Example:

(Dlg_ListSetColumnWidth "list1" "50")

Image Boxes

To fill image boxes with bitmap picture or vector data the following functions are provided:

- Displaying Bitmap files:
(Dlg_ImageBmp *x1 y1 x2 y2 filename*)
- Displaying Vector files of type WMF (Windows Meta File Format)
(Dlg_ImageWmf *x1 y1 x2 y2 filename*)
- Displaying a filled colored rectangle
(Dlg_ImageFill *x1 y1 x2 y2 color_no*)

Here in mean: *x1, y1* = Coordinates in the image box for the upper left corner of the rectangle
 x2, y2 = Coordinates in the image box for the lower right corner of the rectangle

- Drawing colored vectors in an image box:
(Dlg_ImageVector *x1 y1 x2 y2 color_no*)

Here in mean: *x1, y1* = Start point of the vector
 x2, y2 = End point of the vector

To get an useful display in the image box, the function **Dlg_TileDimensions()** is provided. The function allows to retrieve the height and the width of an image control.

Likewise to the functions for operations on list box items, you find both a function to start the operations on an image control element and one to terminate these operations:

(Dlg_ImageStart *key*)

...

(Dlg_ImageEnd)

Example:

```

; x = Width of the image control element
  (setq x (Dlg_TileDimX "picture1"))
; y = Height of the image control element
  (setq y (Dlg_TileDimY "picture1"))
; Start operations on "picture1"
  (Dlg_ImageStart "picture1")
; Fill the entire image field with the blue color (color no. 5) ...
  (Dlg_ImageFill 0 0 x y 5)
; Display the bitmap 'sample.bmp' in the entire image field ...
  (Dlg_ImageBmp 0 0 x y "sample.bmp")
; Operations on "picture1" finished
  (Dlg_ImageEnd)

```

Slider

The function **Dlg_SliderSet** allows to set the properties of a Slider (scroll bar) control.

(Dlg_SliderSet *key pos min max smallStep bigStep*).

The arguments of the function are as follows:

<i>Parameter</i>	<i>Meaning</i>
<i>key</i>	Control Name
<i>pos</i>	Initial position of the scroll box on the scroll bar
<i>min</i>	Minimum value for the slider (slider on scroll bar outermost left)
<i>max</i>	Maximum value for the slider (slider on scroll bar outermost right)
<i>smallStep</i>	Small steps: Increment value when a scroll arrow is picked
<i>bigStep</i>	Big steps: Increment value when the scroll bar is picked

Example:

The following example shows the operation of the function. With the aid of a slider a value between 0 and 100 should be retrieved. A default start value of 4 is assumed. When picking to the scroll arrow the interim value should be incremented by 2. When the scroll bar itself is clicked the value should change by 10. The corresponding function statement is as follows:
(Dlg_SliderSet "slider1" 4 0 100 2 10)

Retrieving User Input and Reacting to it

Until now it has been described, how to initialize and display a dialog box. In this section you find information how to retrieve an user-input-event and how react to it.

The Action Function for Dialog Controls

Generally to any dialog control a function can be associated which is called as soon as the control is activated by the user. The function to do so is: (**Dlg_TileAction** *key* **FLISP-String**).

Example:

```
(Dlg_TileAction "button1" "(button1_func)")
```

The function expects as first argument the identifier of the control element (key name), as second argument a string containing an FLISP expression. The following local variables can be evaluated:

\$key	Name of the control element (key name)
\$value	Current value of the control
\$data	Current client-data
\$reason	Return reason code
\$x,\$y	Coordinates of an image control

Functions to Evaluate Dialog Control Elements

The function (**Dlg_TileGet** *key*) allows to retrieve the value a control currently contains or is set to.

The function returns the value in a string for the control element specified by its unique name *key*.

For List boxes and Sliders (scroll bars) you find the additional functions **Dlg_ListGet** and **Dlg_SliderGet**, which make it ease to read the items of a list or to get the current position of the scroll box.

Example:

The following program code shows how to retrieve all selected items of a list box:

```
(Dlg_ListStart "list1" 12)
;;; Operation 12 starts reading selected lines from a list box
(while (setq I1 (Dlg_ListGet))
  (setq I2 (append I2 I1))
)
(Dlg_ListEnd)
```

The current position of the scroll box in a scroll bar can be evaluated with the function (**Dlg_SliderGet** *key*).

The function returns values in a list as follows:

Parameter	Meaning
-----------	---------

nPos:	Current value (the position of the scroll box)
-------	--

nMin: Minimum value for the slider (slider on scroll bar outermost left)
 nMax: Maximum value for the slider (slider on scroll bar outermost right)
 nSmallStep: Small steps: Increment value when a scroll arrow is picked
 nBigStep: Big steps: Increment value when the scroll bar is picked

Functions to Terminate a Dialog

The standard buttons *OK* and *Cancel* lead to an exiting of a dialog. Beneath that you can assign an action to terminate the dialog to other control elements with the function (**Dlg_DialogDone status**).

If the dialog has been terminated successfully this function returns the value contained in the argument status to the function **Dlg_DialogStart**. This allows to evaluate the exit status of the dialog.

Calling the function (**Dlg_DialogTerm**) allows to remove all opened dialog boxes with one program statement.

Overview

Loading and Unloading of Dialog Files

(Dlg_DialogLoad <i>dlg_file</i>)	Loads a dialog file
(Dlg_DialogUnload <i>dlg_id</i>)	Unloads a dialog file

Opening and Closing Dialog Boxes

(Dlg_DialogNew <i>dlg_name dlg_id [def_action [position]]</i>)	Provides a new dialog (loaded to memory)
(Dlg_DialogStart [<i>Control_function</i>])	Initializes and displays a dialog
(Dlg_DialogDone [<i>Status</i>])	Closes a dialog
(Dlg_DialogTerm)	Terminates all open dialogs

General Operations for Control Elements

(Dlg_TileAction <i>key action</i>)	Assigns an action to a control
(Dlg_TileSet <i>key value</i>)	Sets a control to a value
(Dlg_TileGet <i>key</i>)	Reads the current value of a control
(Dlg_TileClientData <i>key data</i>)	Assigns data to a control
(Dlg_TileMode <i>key mode</i>)	Determines the display mode of a control
(Dlg_TileDimX <i>key</i>)	Returns the width of a control
(Dlg_TileDimY <i>key</i>)	Returns the height of a control
(Dlg_TileSetFont <i>key int</i>)	Sets a font for a control element

List Boxes and Combo Boxes

(Dlg_ListStart <i>key [operation [index]]</i>)	Start function for operations in a list box
(Dlg_ListEnd)	Terminates operations in a list box
(Dlg_ListAdd <i>value</i>)	Adds or modifies a list box item
(Dlg_ListGet)	Reads a list box item
(Dlg_ListSetTabstops <i>key values</i>)	Sets Tab-stops in a list box
(Dlg_ListSetColumnWidth <i>key width</i>)	Sets the width of columns in a list box

Slider Control

(Dlg_SliderGet <i>key</i>)	Reads the values and properties of a slider control
(Dlg_SliderSet <i>key pos min max smallStep bigStep</i>)	Sets the values and properties of a slider control

Image Controls

(Dlg_ImageStart <i>key</i>)	Start function to display bitmaps or vector graphic in an image box
(Dlg_ImageEnd)	Terminates operations on an image box
(Dlg_ImageVector <i>x1 y1 x2 y2 color</i>)	Draws a vector in an image box
(Dlg_ImageFill <i>x1 y1 x2 y2 color</i>)	Draws a filled rectangle in an image box
(Dlg_ImageBmp <i>x1 y1 x2 y2 bmp_file</i>)	Displays a bitmap file in an image box
(Dlg_ImageWmf <i>x1 y1 x2 y2 wmf_file</i>)	Displays an vector graphic of an WMF file in an image box

Dialog Functions: Reference

Dlg_DialogDone

The function **Dlg_DialogDone** terminates a specified dialog.
(Dlg_DialogDone [status])

Parameter:

status: (Optional) Return value for **Dlg_DialogStart**
 Number > 1

Returns:

The function returns the XY screen coordinates of the upper left corner of the dialog box when it was closed. The retrieved screen coordinates can be used in a later call of the function **Dlg_DialogNew**.

Examples:

(Dlg_DialogDone)
 (Dlg_DialogDone 2)

Dlg_DialogLoad

The function **Dlg_DialogLoad** loads a dialog file containing dialog box descriptions.

(Dlg_DialogLoad dlg_file)

Parameters:

dlg_file String containing the file name of the dialog file. The file type **.dlg** does not need to be supplied. Drive and path name may be specified optional.

Returns:

dlg_id If the function succeeds, it returns a positive integer required as Handle for later execution of the functions **Dlg_DialogNew** and **Dlg_DialogUnload**.

The function returns -1, if the specified dialog file could not be loaded.

Examples:

(Dlg_DialogLoad "test")
 (Dlg_DialogLoad "c:\\dlg\\test")
 (Dlg_DialogLoad "c:/dlg/test")

See also:

findfile

Dlg_DialogUnload

Dlg_DialogNew

Dlg_DialogNew

The function **Dlg_DialogNew** provides the specified new dialog for further usage by the application. The function is required for later displaying the dialog with the function **Dlg_DialogStart**.

(Dlg_DialogNew **dlg_name** **dlg_id** [**default_action** [**position**]])

Parameters:

- dlg_name:** String supplying the unique name of the dialog.
- dlg_id:** Handle of the dialog file which contains the dialog.
- def_action:** Optional string containing a FLISP expression which is evaluated, if a dialog control is clicked for which no action function has been specified.
- position:** Optional 2D point which identify the screen coordinates of the upper left corner of the dialog. Specifying -1,-1 forces the dialog to be displayed centered on the screen.

Returns:

If the function succeeds, it returns T, otherwise nil.

Examples:

```
(Dlg_DialogNew "dialog_1" dlg_id)
(Dlg_DialogNew "dialog_1" dlg_id "(default_func)")
(Dlg_DialogNew "dialog_1" dlg_id "(default_func)" '(0 0))
(Dlg_DialogNew "dialog_1" dlg_id "" '(-1 -1))
```

Dlg_DialogStart

The function **Dlg_DialogStart** starts the dialog box.

(Dlg_DialogStart [**init_func**])

Parameter:

- init_func:** Function to initialize the controls of the dialog. Used to set controls to specified values and by assigning actions to control with the function **Dlg_TileAction()**.

Returns:

- Status:** Status of the dialog
- | | |
|----|--|
| 1 | User pressed OK |
| 0 | User pressed Cancel or equivalent |
| -1 | All dialogs terminated with Dlg_DialogTerm |
| >1 | (Dlg_DialogDone n) |

Examples:

```
(Dlg_DialogStart "(dlg_init)")
(Dlg_DialogStart)
```

Dlg_DialogTerm

The function **Dlg_DialogTerm** terminates (closes) **all** open dialog boxes.
(Dlg_DialogTerm)

Returns:

The function always returns nil.

See also:

Dlg_DialogDone

Dlg_DialogUnload

The function **Dlg_DialogUnload** removes a dialog from memory.
(Dlg_DialogUnload dialog_ID)

Parameter:

dialog_ID: Dialog Handle. Return value of **Dlg_DialogNew()**.

Returns:

The function always returns nil.

Examples:

```
(setq dlg_id (Dlg_DialogLoad "test"))
(Dlg_DialogUnload dlg_id)
```

See also:

Dlg_DialogLoad

Dlg_ImageBmp

The function **Dlg_ImageBmp** fills an Image Control with a bitmap file.
(Dlg_ImageBmp x1 y1 x2 y2 BmpPath)

Parameters:

It is required to pass to the function the coordinates which determines a rectangle in the Image Control to fill with the bitmap. **Note:** If all coordinates are set to 0, the entire window is filled.

x1: X coordinate of the upper left corner

y1: Y coordinate of the upper left corner

x2: X coordinate of the lower right corner

y2: Y coordinate of the lower right corner

file: Filename of the BMP file (probably including drive and path)

Returns:

If the function succeeds, it returns T, otherwise nil.

Example:

```
(setq x (Dlg_TileDimX "picture1"))
(setq y (Dlg_TileDimY " picture 1"))
(Dlg_ImageStart " picture 1")
(Dlg_ImageBmp 0 0 x y "testpic.bmp")
(Dlg_ImageEnd)
```

See also:

Dlg_TileDimX
Dlg_TileDimY
Dlg_ImageWmf
Dlg_ImageStart
Dlg_ImageEnd
Dlg_ImageFill

Dlg_ImageEnd

The function **Dlg_ImageEnd** ends the operation(s) on an Image Control, which have been started with the function **Dlg_ImageStart**.

(Dlg_ImageEnd)

Returns:

If the function succeeds, it returns T, otherwise nil.

Example:

```
(Dlg_ImageStart "picture1")
...
(Dlg_ImageEnd)
```

See also:

Dlg_ImageStart
Dlg_ImageVector
Dlg_ImageFill
Dlg_ImageBmp
Dlg_ImageWmf

Dlg_ImageFill

The function **Dlg_ImageFill** draws a filled rectangle in an Image Control in the color and at the coordinates specified.

(Dlg_ImageFill x1 y1 x2 y2 color)

Parameters:

x1: X coordinate of the upper left corner
y1: Y coordinate of the upper left corner
x2: X coordinate of the lower right corner

y2: Y coordinate of the lower right corner
 color: Color number. Valid values are

Value	Meaning
> 0	FelixCAD color number
-2	Current background color of the graphic window
-15	Background color of the current dialog box
-16	Foreground color of the dialog (color of text items)
-18	Previously used Vector or Fill color (current color)

Returns:

If the function succeeds, it returns T, otherwise nil.

Example:

```
(Dlg_ImageStart "picture1")
(Dlg_ImageFill 0 0 100 100 2)
(Dlg_ImageEnd)
```

See also:

Dlg_ImageStart
 Dlg_ImageVector
 Dlg_ImageFill
 Dlg_ImageBmp
 Dlg_ImageWmf

Dlg_ImageStart

The function **Dlg_ImageStart** starts operations on an Image Control.

(Dlg_ImageStart key)

Parameter:

key: Name of the dialog control of type Image.

Returns:

If the function succeeds, it returns T, otherwise nil.

Example:

```
(Dlg_ImageStart "picture1")
...
(Dlg_ImageEnd)
```

See also:

Dlg_ImageEnd
 Dlg_ImageVector
 Dlg_ImageFill
 Dlg_ImageBmp
 Dlg_ImageWmf

Dlg_ImageVector

The function **Dlg_ImageVector** draws a vector in an Image Control.
(Dlg_ImageVector x1 y1 x2 y2 color)

Parameters:

x1 X coordinate of the start point
y1 Y coordinate of the start point
x2 X coordinate of the end point
y2 Y coordinate of the end point
color: Color number

No.	Meaning
> 0	FCAD color number
-2	Current background color of the graphic window
-15	Background color of the current dialog box
-16	Foreground color of the dialog (color of text items)
-18	Previously used Vector or Fill color (current color)

Returns:

If the function succeeds, it returns T, otherwise nil.

Example:

```
(Dlg_ImageStart "picture1")  
(Dlg_ImageVector 0 0 50 50 2)  
(Dlg_ImageEnd)
```

See also:

Dlg_ImageStart
Dlg_ImageVector
Dlg_ImageFill
Dlg_ImageBmp
Dlg_ImageWmf

Dlg_ImageWmf

The function **Dlg_ImageWmf** allows to display a Windows Metafile (WMF) in an Image Control. WMF is a vector format.
(Dlg_ImageWmf x1 y1 x2 y2 WmfPath)

Parameter:

It is required to pass to the function the coordinates which determines a rectangle in the image control to fill with the WMF image. Note: If all coordinates are set to 0, the entire image control is filled.

x1: X coordinate of the upper left corner
y1: Y coordinate of the upper left corner

x2: X coordinate of the lower right corner
 y2: Y coordinate of the lower right corner
 file: Filename of the WMF file (probably including drive and path)

Returns:

If the function succeeds, it returns T, otherwise nil.

Example:

```
(setq x (Dlg_TileDimX "picture1"))
(setq y (Dlg_TileDimY "picture1"))
(Dlg_ImageStart "picture1")
(Dlg_ImageBmp 0 0 x y "testpic.wmf")
(dlg_imagend)
```

See also:

Dlg_TileDimX
 Dlg_TileDimY
 Dlg_ImageBmp
 Dlg_ImageStart
 Dlg_ImageEnd
 Dlg_ImageFill

Dlg_ListAdd

The function **Dlg_ListAdd** allows to add or replace an entry in a List Box.
(Dlg_ListAdd *item*)

Parameter:

item String to be added to the list respectively to replace an item in the list.

Examples:

```
(Dlg_ListStart "liste1" 1 0) ; change the first line
(Dlg_ListAdd "Fist line of list box")
(Dlg_ListEnd)
```

See also:

Dlg_ListStart
 Dlg_ListGet
 Dlg_ListEnd
 Dlg_ListSetTabstops
 Dlg_ListSetColumnWidth

Dlg_ListEnd

The function **Dlg_ListEnd** terminates operations on a List Box.

(Dlg_ListEnd)

Returns: Nil

See also:

Dlg_ListStart

Dlg_ListAdd

Dlg_ListGet

Dlg_ListSetTabstops

Dlg_ListSetColumnWidth

Dlg_ListGet

The function **Dlg_ListGet** allows to read list box entries.

(Dlg_ListGet)

Returns:

The function returns a list with the scheme (*position item*):

position List Box index.

item Content of the line

Example:

```
(Dlg_ListStart "list" 13)
```

```
(while (setq l1 (Dlg_ListGet)) (setq l2 (append l2 l1)) )
```

```
(Dlg_ListEnd)
```

See also:

Dlg_ListStart

Dlg_ListSetTabstops

Dlg_ListSetColumnWidth

Dlg_ListAdd

Dlg_ListEnd

Dlg_ListSetColumnWidth

The function **Dlg_ListSetColumnWidth** serves to set the column width in a multiple column list box.

(Dlg_ListSetColumnWidth *key width*)

Parameter:

key Name of the dialog control of type LISTBOX.

width String specifying the value for the column width in Pixel.

Returns: Nil.

Example:

(Dlg_ListSetColumnWidth "list1" "50")

See also:

Dlg_ListSetTabstops

Dlg_ListSetTabstops

The function **Dlg_ListSetTabStops** allows to set TAB stops in a list box.

(Dlg_ListSetTabstops *key tabstops*)

Parameters:

key Name of the dialog control of type LIST BOX.
tabstops String containing the tab stop positions. The numbers specifying the tab stops are separated by spaces. A space or a zero sets a tabulator of two dialog units. One or more ascending values define in dialog units the tabulator positions.

Returns:

Nil

Example:

(Dlg_ListSetTabstops "list1" "10 20 30 40")

See also:

Dlg_ListSetColumnWidth

Dlg_ListStart

The function **Dlg_ListStart** starts the operations performed on a list box.

(Dlg_ListStart *key [operation [index]]*)

Parameter:

key Name of the dialog control of type LIST BOX.
operation Operation to be perform in the list box as documented in the following table:

Operation	Meaning	Function
1	Change selected list item	Dlg_ListAdd
2	Append a new list item	Dlg_ListAdd
3	Delete the current content of the list and create a new one	Dlg_ListAdd
10	Insert a list item at a specified position	Dlg_ListAdd
11	Delete the list item at a specified position	
12	Returns all selected list items	Dlg_ListGet
13	Returns all list items	Dlg_ListGet
14	Mark list item at specified position	

	-1 = all (only for list boxes allowing multiple selection)	
15	Un-mark list item at specified position -1 = all (only for list boxes allowing multiple selection)	
16	Set cursor to list item	

index Number specifying a list item (respectively its position) the operation should take effect on. Note: The index of the first list item is 0.

- 0 is standard,
- 1 DLG_L_INSERT only: insert starting at cursor position

Returns:

If the function succeeds, it returns T, otherwise nil.

Examples:

```

; Change the fifth item of the list
(Dlg_ListStart "list" 1 4)
(Dlg_ListAdd "new value")
(Dlg_ListEnd)

; Get all items of the list
(Dlg_ListStart "list" 12)
(while (setq l1 (Dlg_ListGet))
      (setq l2 (append l2 l1)))
(Dlg_ListEnd)

```

See also:

- Dlg_ListAdd
- Dlg_ListGet
- Dlg_ListEnd
- Dlg_ListSetTabstops
- Dlg_ListSetColumnWidth

Dlg_SliderGet

The function **Dlg_SliderGet** returns the current settings of the specified Slider control element.

(Dlg_SliderGet *key*)

Parameter:

key Name of the dialog control of type SLIDER.

Returns:

If the function succeeds, it returns the following list:

- nCurPos* position current of the scroll box
- nMinPos* minimum value at the extreme left or top position of the scroll bar
- nMaxPos* maximum value at the extreme right or bottom position of the scroll bar
- nSmallStep* small increment value which is used when an arrowhead of the slider (scroll arrow) is picked
- nBigStep* big increment value which is used when the scroll bar is picked

See also:

Dlg_SliderSet

Dlg_SliderSet

The function **Dlg_SliderSet** allows to set initial or current values for a Slider control element.

**(Dlg_SliderSet key nCurPos nMinPos nMaxPos
nSmallStep nBigStep)**

Parameter:

- key: Name of the Dialog element
- nCurPos: Value for the current the position of the scroll box
- nMinPos: Minimum value for the extreme left or top position in the scroll bar
- nMaxPos: Maximum value for the extreme right or bottom position in the scroll bar
- nSmallStep: Small increment value which is used when an arrowhead of the slider (scroll arrow) is picked
- nBigStep: Big increment value which is used when the scroll bar is picked

Returns:

Nil

Example:

(Dlg_SliderSet "slider1" 0 0 100 2 10)

See also:

Dlg_SliderGet

Dlg_TileAction

The function **Dlg_TileAction** binds an action expression to a dialog control element. This Lisp expression provided by the application developer is called

when the a control element has been clicked (e.g. a button) or edited (e.g. an input field).

(Dlg_TileAction key action)

Parameters:

- key: Identifier name of the dialog control element.
- action: String containing a FLISP expression which is evaluated, if a dialog control is clicked.

The following local variables can be evaluated:

- \$key Name of the control element (key name)
- \$value Current value of the control
- \$data Current client-data
- \$reason Return reason code
- \$x,\$y Coordinates of an image control

Returns:

If the function succeeds, it returns T, otherwise nil.

\$reason	Meaning
1	A dialog element has been selected (e.g. by picking)
2	For <i>Edit</i> controls: This value indicates that the user has left the field (e.g. by using the TAB key)
3	For <i>Slider</i> controls: This value indicates that the user has modified the scroll box position
4	For <i>List</i> items and <i>Image</i> controls: the control element has been selected by <i>double click</i>

Examples:

```
(Dlg_TileAction "item1" "(function_el1)")
(Dlg_TileAction "picture1" "(princ $x)")
```

See also:

Dlg_TileClientData

Dlg_TileClientData

The function **Dlg_TileClientData()** allows to allocate data to a dialog element, which may later be recalled via "dlg_callback_packet. *cpkt->client_data*" in a **Dlg_TileAction()** call.

(Dlg_TileClientData key data)

Parameters:

- key Identifier name of the dialog control element.
- data Data for the dialog control element.

Returns:

If the function succeeds, it returns T, otherwise nil.

Example:

```
(Dlg_TileClientData "list1" "red yellow black")
```

Dlg_TileDimX

The function **Dlg_TileDimX** allows to retrieve the width of a dialog control element.

(Dlg_TileDimX key)*Parameter:*

key Name of the dialog control.

Returns:

If the function succeeds, it returns the width of the dialog control element, otherwise nil.

Examples:

```
(setq x (Dlg_TileDimX "picture1"))
```

See also:

Dlg_TileDimY
Dlg_ImageVector
Dlg_ImageFill
Dlg_ImageBmp
Dlg_ImageWmf

Dlg_TileDimY

The function **Dlg_TileDimY** allows to retrieve the height of a dialog control element.

(Dlg_TileDimY key)*Parameter:*

key Name of the dialog control.

Returns:

If the function succeeds, it returns the width of the dialog control element, otherwise nil.

Examples:

```
(setq y (Dlg_TileDimY "picture1"))
```

See also:

Dlg_TileDimX
Dlg_ImageVector
Dlg_ImageFill
Dlg_ImageBmp
Dlg_ImageWmf

Dlg_TileGet

The function **Dlg_TileGet** allows to retrieve the current value of a dialog control.

(Dlg_TileGet key)

Parameter:

key Name of the dialog control

Returns:

The function returns the current content of the dialog control element.

Name of the key	Content of value
DIALOG_CAPTION	Dialog Title
BUTTON	Text of the button
BMPBUTTON	--- (No value !)
RADIOBUTTON	0 = Radio button not marked 1 = Radio button is marked
CHECKBOX	0 = Check box not marked 1 = Check box is marked 2 = Check box is undefined (x3State-Mode)
LISTBOX	"" = Null string, if no item of the list box is selected "n" = Index of the selected items "n1 n2 n3 ..." = Indices of the selected items (occurs only in multiple select list boxes)
COMBOBOX	Content of the Edit field of the Combo Box respectively of the selected Combo Box item
WINDOW	Current Mode of the Image (Window Control) 0 = Normal 1 = Inverted
SLIDER	Position of the Scroll Box
GROUPBOX	Identifier (Group box title)
LABEL	Content of the Static element (Label Control)
EDIT	Content of the Edit Control
INPUT	Content of the Input Control

Example:

(setq value1 (Dlg_TileGet "edit1"))

Dlg_TileMode

The function **Dlg_TileMode** sets a certain mode for a dialog control element.

(Dlg_TileMode key mode)

Parameters:

key Unique name of the dialog control element.

mode Positive integer specifying the mode for the dialog control as follows:

Mode	Meaning
0	Make dialog control selectable
1	Make dialog control not available
2	Set focus to the dialog control
3	Select entry of edit control
4	Toggle Image Control: Highlighted Normal
5	Set dialog control to visible
6	Hide dialog control

Returns:

If the function succeeds, it returns T, otherwise nil.

Examples:

```
(Dlg_TileMode "edit1" 0)
(Dlg_TileMode "edit1" 3)
(Dlg_TileMode "edit1" 2)
```

Dlg_TileSet

The function **Dlg_TileSet** sets the specified control element to a specified value.

(Dlg_TileSet key value)

Parameter:

key Name of the dialog control.

value New content for the dialog element.

Mode	Meaning
0	Make dialog control selectable
1	Make dialog control not available
2	Set focus to the dialog control
3	Select entry of edit control
4	Toggle Image Control: Highlighted Normal
5	Set dialog control to visible
6	Hide dialog control

Returns:

If the function succeeds, it returns T, otherwise nil.

Examples:

```
(Dlg_TileSet "text1" "new value")
(Dlg_TileSet "checkbox1" "1")
(Dlg_TileSet "DIALOG_CAPTION" "Dialog Title")
```

See also:

Dlg_TileGet

Dlg_TileSetFont

The function **Dlg_TileSetFont** sets a certain type of font for a dialog control element.

(Dlg_TileSetFont *key font*)

Parameters:

key Name of the dialog control.

font A short integer in the range of 0 through 3 specifies the font used in the dialog as follows:

Value	Meaning
0	System font
1	System fixed font
2	ANSI system font
3	ANSI system fixed font

Returns:

Nil

Example:

(Dlg_TileSetFont "button1" 2)

Overview: Alphabetically Ordered

Dlg_DialogDone	Closes a dialog
Dlg_DialogDonePositioned	Closes a dialog and returns the last screen position of that dialog
Dlg_DialogLoad	Loads a dialog file
Dlg_DialogNew	Provides a new dialog (loaded to memory)
Dlg_DialogNewPositioned	Provides a new dialog at a specified screen position (loaded to memory)
Dlg_DialogStart	Initializes and displays a dialog
Dlg_DialogTerm	Terminates all open dialogs
Dlg_DialogUnload	Unloads a dialog file
Dlg_ImageBmp	Displays a bitmap file in an image box
Dlg_ImageEnd	Terminates operations on an image box
Dlg_ImageFill	Draws a filled rectangle in an image box
Dlg_ImageStart	Start function to display bitmaps or vector graphic in an image box
Dlg_ImageVector	Draws a vector in an image box
Dlg_ImageWmf	Displays an vector graphic of an WMF file in an image box
Dlg_ListAdd	Adds or modifies a list box item
Dlg_ListEnd	Terminates operations in a list box
Dlg_ListGet	Reads a list box item
Dlg_ListSetColumnWidth	Sets the width of columns in a list box
Dlg_ListSetTabStops	Sets Tab-stops in a list box
Dlg_ListStart	Start function for operations in a list box
Dlg_SliderGet	Reads the values and properties of a slider control
Dlg_SliderSet	Sets the values and properties of a slider control
Dlg_TileAction	Assigns an action to a control
Dlg_TileClientData	Assigns data to a control
Dlg_TileDimensions	Returns the size (height and width) of a control
Dlg_TileGet	Reads the current value of a control
Dlg_TileMode	Determines the display mode of a control
Dlg_TileSet	Sets a control to a value
Dlg_TileSetFont	Sets a font for a control element

CHAPTER 12

System and Drawing File Variables

When working with FelixCAD in many cases various parameters like default numerical values, factors, or base coordinates are set or changed as result of using the settings dialogs or other command execution. These parameters are stored in system variables and can be retrieved with the next call of a corresponding function or command. Evaluating and setting system variables also provides advanced possibilities of customization and programming. This chapter of the manual is divided into three section describing in detail the global system variables, the local drawing variables and the local dimensioning variables. Each section is ordered alphabetically.

Global and Local Variables

Global variables are common variables valid for the entire program and in any of the drawings currently open. For example, these variable allow to set default values for a template drawing, for default linear and angular units (when no template drawing is used). Also, the current file settings of the items of the user interface (pull-down menu, palettes, tablet-menu) can be retrieved.

Local variables are valid only in a certain drawing. These variables may differ from drawing to drawing currently opened on the desktop. Most of these variables are stored with the drawing. This means, that you will find the same drawing environment when the drawing is reopened. The major part of the local variables stores current settings for drawing and editing commands, e.g. hatching and dimensioning parameters. Others allow to retrieve current user coordinate system and view settings. A number of variables store **Read-Only** values. You can return their values, but you cannot edit them. Variables notified as **Read-Write** can be changed in their values.

Retrieving and Setting the Variables

Often, the goal of programming is to alter the default settings for the program environment or for a specific drawing. The **LISP functions (getvar ...)** respectively **(setvar ...)** allow you to retrieve or modify the current setting of a system variable. You can read the current value of a variable by using the **(getvar ...)** function.

You can alter the value of a variable with the **(setvar ...)** function with exception of the Read-Only variables. When you supply a value to the function whose data type or data range is invalid an error message is returned. More information and examples on using the functions (getvar ...)

and (setvar ...) to manipulate variable settings is found in the chapter *Lisp Programming* in this manual.

Scheme of the System Variables Reference

This chapter describes each of the variables. In addition to the short description of the meaning you find structured information on the properties of the variable and a cross-reference to commands which set or modify the variable, as follows:

Properties

- **Status**

With the item Status are two kinds of variables are described:

Variables notified as **Read/Write** can be set and allow to manipulate settings for the program and the drawing environment. A number of global and local variables can be retrieved but cannot be modified. These variables are notified as **Read-Only**. They contain either fixed values (like the information on the program platform) or contain updated information on a current status (e.g. user coordinate and view settings). Some variables contain the result of a previously used inquiry command (e.g. the last computed area).

- **Type**

This item specifies the valid data type of the variable, which may be on of the following:

<i>Integer</i>	contains integer values
<i>Real</i>	contains floating point values
<i>String</i>	contains a string
<i>2D-Point</i>	contains coordinates of a two dimensional point
<i>3D-Point</i>	contains coordinates of a two dimensional point
<i>Bit-Code</i>	contains a Bit-Code
<i>Pixel-Coord.</i>	contains the screen coordinates in pixels

- **Default / Initial Value**

A number of variables contain a certain default or initial value. This item of a variable description documents the standard settings of the program. If a local variable is described, you find the default value documented for the case that no template drawing of your application is used.

But note, that many of the Read/Write variables may contain different values as they could have been set in a previous session with the program or a previous working on a re-opened drawing.

- **Storage Location**

The variables can be divided into three groups corresponding to their location of storage:

1. Global variables, whose values are stored in the initialization file *applic.INI* or in the configuration file *applic.CFG*. Please note, that you should not modify an applications INI file. The CFG file is binary and cannot be altered with a text editor.
2. Local variables, whose values are stored **in the drawing** itself.
3. Global or local variables, whose values are valid during the working session with the program or while working on a drawing. These variables are not maintained when the drawing is closed or the program is terminated.

Commands to set or get variables

In many cases the variable reference contains a paragraph cross-referencing command names. This is the case, if the variable is influenced in any way by a built-in command.

Note: The commands SETTINGS, PRECPAR, EDITPAR and VIEWPAR serve explicitly to set system variables. All of these commands can be call transparently, which means while another command is executed.

Global Variables: Reference

ACTDB

The variable ACTDB (Active Database) returns the ID number of the currently active drawing database which is currently active. Because up to four drawings may be opened simultaneously, this variable may contain one of the values as follows:

- 1 = No drawing database opened
- 0 = First drawing database active
- 1 = Second drawing database active
- 2 = Third drawing database active
- 3 = Fourth drawing database active

Properties

Status Read-Only
Type Integer (-1 ... 3)
Initial value -1, because normally no drawing is opened when the program starts up.

Commands

The variable is reset if one of the commands NEW, OPEN, or CLOSE has been executed successfully.

See also

Lisp functions (**flxnames ...**) and (**getactvport ...**);
Local system variable CVPORT (Current viewpoint).

AREA

This variable returns the area computed by the most recently executed command AREA in the current session with the program. Note: The variable is not drawing sensitive.

Properties

Status Read-Only
Type Real
Initial Value 0.0

Commands

The variable is reset by the usage of the command AREA.

See also

Global system variables PERIMETER and DISTANCE.

CDATE

The variables CDATE (Current Date) return the current date and time. The calendar date returned is as follows:

YYYYMMDD.HHMMSSmm

== (Year Month Day . Hour Minute Second Millisecond).

Properties

<i>Status</i>	Read-Only
<i>Type</i>	Real
<i>Value</i>	Current date and time of the computer system
<i>Example</i>	19950907.12055239

CIRCLERAD

This variable allows to sets the default value for the radius requested by the command CIRCLE (Circle Radius).

Properties

<i>Status</i>	Read / Write (not stored)
<i>Type</i>	Real
<i>Initial Value</i>	0.0

Command

If the variable is set to 0.0, the command CIRCLE does not display a default value when the user is prompted for a radius or a diameter. Any other value is displayed as default, which might be accepted by the user by pressing RETURN. The variable is influenced by the command CIRCLE itself. As soon as the user has specified a radius or a diameter the variable is set to that value.

CIRCLERES

Determines the number of line segments used to display circles and arcs in the drawings (Circle Resolution). Valid integer values lie in the range between 8 and 1024.

Note:

- The higher the value, the more ideal the displaying of arcs and circles.
- The lower the value, the better the performance when dynamically dragging arched objects during entity generation or selection and when redrawing or regenerating a drawing viewport.

Properties

<i>Status</i>	Read / Write (saved in the CFG file)
<i>Type</i>	Integer (8 ... 1024)
<i>Default Value</i>	48

Command

The variable may be modified by the user via the dialog command DRAWMODE in the option box *Circle Segments*.

CMDACTIVE

This variable (Command Active) allows the developer to evaluate the type of a currently active command. A bit code is returned with a meaning as follows:

Bit 1 = Standard command
Bit 2 = Standard and transparent command
Bit 4 = Macro script

Properties

Status Read-Only
Type Integer (Bit-Code)

CMDECHO

This variable (Command Echo On/Off) allows the developer to set a preference for the mode a statement using the LISP function (**command ...**) is performed as follows:

0 = OFF The command sequences are not visible for the user
1 = ON The command sequences are visible for the user

Properties

Status Read / Write (not saved)
Type Integer
Default Value 1

Command

Not influenced by a built-in command.
Programming: See function (**command ...**).

CMDNAMES

This variable (Command Names) allows the developer to retrieve in his routines the name(s) of the currently active command(s).

Properties

Status Read-Only
Type String
Examples "LINE", "LINE 'ZOOM"

Example

```
> LINE
From point: 'ZOOM
Zoom scale factor: (progn (setq x (getvar "cmdnames"))(princ x)(princ))
LINE 'ZOOM
```

DATE

This variable returns the current date and the current time in the *Julian Date* format.

Properties

Status Read-Only
Type Real
Return Value Current date / time of the computer system

Example 2449968.64956018

See also

Global system variable CDATE.

DEFANGBASE

If no template drawing is used when the user creates a new drawing, this variable (Default Angle Base) allows to set a default value for the zero degree angle direction. As reference point for the direction serves the positive X axis of the current user coordinate system.

Properties

Status Read / Write (saved in the system's CFG file)

Type Real

Default Value 0.0

DEFANGDIR

If no template drawing is used when the user creates a new drawing, this variable (Default Angle Direction) sets the default direction when angle are specified by the user as follows:

0 = Counter-clockwise

1 = Clockwise

Properties

Status Read / Write (saved in the system's CFG file)

Type Integer (0 | 1)

Default Value 0

DEFAUNITS

If no template drawing is used when the user creates a new drawing, this variable (Default Angular Units) allows the programmer to determine the unit format to enter or to measure angles as follows:

0 = Decimal Degrees

1 = Degrees/Minutes/Seconds

2 = Grads

3 = Radians

4 = Surveyor's Units

Properties

Status Read / Write (saved in the system's CFG file)

Type Integer (0 ... 4)

Default Value 0

DEFAUPREC

If no template drawing is used when the user creates a new drawing, this variable (Default Angular Units Precision) determines the standard setting for

the number of decimal places to be displayed when prompting the user for angle specifications. A setting of up to eight decimal places is valid.

Properties

Status Read / Write (saved in the system's CFG file)

Type Integer (0 ... 8)

Default Value 0

DEFDIMZIN

If no template drawing is used when the user creates a new drawing, this variable (Default Dimension Zero Inches) determines the display mode of dimensions.

- **Note: The meaning of the integer values (in the range between 0 and 16) is documented in the description of the local system variable DIMZIN (see below).**

Properties

Status Read / Write (saved in the system's CFG file)

Type Integer (0 ... 16)

Default Value 0

DEFLUNITS

If no template drawing is used when the user creates a new drawing, this variable (Default Linear Units) allows the programmer to determine the standard setting for linear unit format used in the drawing as follows:

- 1 = Scientific
- 2 = Decimal
- 3 = English: Engineering
- 4 = English: Architecture
- 5 = Fraction

Properties

Status Read / Write (saved in the system's CFG file)

Type Integer (1 ... 5)

Default Value 2

DEFLUPREC

If no template drawing is used when the user creates a new drawing, this variable (Default Linear Units Precision) allows the programmer to determine the standard setting for the number of decimal places of linear units. A value up to 8 decimal places is accepted.

Properties

Status Read / Write (saved in the system's CFG file)

Type Integer (0 ... 8)

Default Value 2

DIASTAT

This variable (Dialog Status) allows the programmer to retrieve the status, how the most recently dialog box of a built-in GDE command called by the user has been exited. Allows to retrieve the exit status of a dialog when inside a LISP program a built-in dialog-command has been called.

0 = Dialog has been terminated by OK

1 = Dialog has been terminated by Cancel

Properties

Status Read-Only

Type Integer

Initial Value 0

See also

Global system variables CMDNAMES, CMDACTIVE, LASTVAR.

DISTANCE

This variable (Distance) returns the value of a length respectively distance measurement most recently performed by a call of the command 'DIST.

Properties

Status Read-Only

Type Real

Initial Value 0.0

Commands

The variable is reset if the user has performed the command 'DIST.

Note: This variable is drawing-sensitive.

ERRNO

This variable (Error Number) may contain an error number set by the system if a built-in API function could not be executed error-free.

Properties

Status Read-Only

Type Integer

Initial Value 0

EXPERT

This variable allows the developer to set on or off a so called Expert Mode for his file operations.

0 = Requests may display warnings or security prompts in some cases

1 = Requests do not display warnings or security prompts

Properties

Status Read / Write (not saved)

Type Integer
Default Value 0

FCTEMPLATE

This variable allows the programmer to set the default name (including drive and path) for the template drawing used when the user executes the command NEW. No template drawing is pre-set, if a null string ("") is specified.

Properties

Status Read / Write (saved in the system's CFG file)
Type String
Example "c:\applic\template.flx"

Command

Note: The user has the possibility to specify another template drawing or to disable the usage of a template drawing with the command NEW. The user's specification is written to the system variable FCTEMPLATE.

FCVERSION

This variable returns the name and the version number of the FelixCAD Graphic Developer's Engine.

Properties

Status Read-Only
Type String
Example "FCAD 2.0"

FILEDIA

This variable (File Dialog) may be set or evaluated by the developer to determine, if for certain commands a dialog box or a command line request sequence is used.

0 = Dialog Box version of the command is used

1 = Command prompts parameters at the command line

Properties

Status Read / Write (not saved)
Type Integer (0 | 1)
Default Value 1

HIGHLIGHT

This variable is used to determine, if selected drawing objects are marked by highlighting them. The variable may be set to:

0 = OFF

1 = ON

Properties

Status Read / Write (not saved)
Type Integer (0 | 1)
Default Value 1

LANGUAGE

This variable allow to determine the language used by the FelixCAD.
 Currently two values are accepted:
 1 = German
 2 = English

Properties

Status Read / Write (not saved)
Type Integer
Default Value 1 (read from INI file)

LASTVAR

This variable (Last Variable) allows the developer to retrieve the most recently called system variable.

Properties

Status Read-Only
Type String
Example "PDMENUNAME"

OFFSETDIST

This variable (Offset Distance) sets the default value for the command OFFSET when prompting the user for the distance between the original objects (contour) and the copy of the objects.

Properties

Status Read / Write (not saved)
Type Real
Initial Value 0.0

Command

The variable is probably re-written when the user executes the command OFFSET.

PALETTE1 ... PALETTE n

This variable returns the filename of a palette number n currently loaded and displayed on the desktop. The variable returns in a string the filename including drive and path.
 If no palette has been loaded indexed as n , the variable returns a null string ("").

Properties

Status Read-Only
Type String
Example "C:\APX\AP_DRAW.MNP"

Commands

PALETTE, PALMAN, MENÜ

PANSCALE

This variable (Pan Scale) specifies the factor for the PAN commands 'PANRIGHT, 'PANLEFT, 'PANUP, and 'PANDOWN, which move the view of the current viewport up or down or to the left or right.

Properties

Status Read / Write (saved in the system's CFG file)
Type Real
Default Value 0.5

Commands

The variable may be set by the user with the DISPPAR command.

PDMENUNAME

This variable (Pull-down Menu Name) returns the filename including drive and path of the currently used pull-down menu.

Properties

Status Read-Only
Type String
Example "c:\applic\applic.mnu"

Command

MENU

PERIMETER

This variable returns the value of the perimeter computed by the most recently usage of the command AREA.

Properties

Status Read-Only
Type Real
Initial Value 0.0

Command

AREA

See also

Global system variable AREA.

PLATFORM

This variable returns a string to identify the operating system platform.

Properties

<i>Status</i>	Read-Only
<i>Type</i>	String
<i>Value</i>	"Win32" (32bit version)

POLYSIDES

This variable (Polygon Sides) allows to set the default value for the command NGON when prompting the user to enter the number of sides of the regular polygon to be drawn. The value of type integer can lie in the range between 3 and 1024.

Properties

<i>Status</i>	Read / Write
<i>Type</i>	Integer (3 ... 1024)
<i>Default Value</i>	4

Command

The setting is updated when the command NGON is used.

RINGDIA1

This variable (Ring Diameter 1) specifies for the built-in command RING the preference value for the inner diameter .

Properties

<i>Status</i>	Read / Write (not saved)
<i>Type</i>	Real
<i>Default Value</i>	0.5

Command

The variable is probably reset if the command RING has been used.

RINGDIA2

This variable (Ring Diameter 2) specifies for the built-in command RING the preference value for the outer diameter .

Properties

<i>Status</i>	Read / Write (not saved)
<i>Type</i>	Real
<i>Default Value</i>	1.0

Command

The variable is probably reset if the command RING has been used.

SCREENMODE

This variable returns a flag to evaluate, if the *Lisp/History* window is active.

0 = Lisp/History text window is active

1 = Lisp/History text window is closed

Properties

<i>Status</i>	Read-Only
<i>Type</i>	Integer
<i>Initial Value</i>	0 (the text window is not displayed "on top" when a new session with the program is started)

Command

The command 'TSCREEN re-sets this variable. The variable is only of interest for LISP programming, which allows to set *the Command History / Lisp Interpreter* command line window on top with the function (**textscr**) and to remove the command line window with the functions (**graphscr**) to not overlap the drawing windows.

SCREENSIZE

This variable (Screen Size) returns the size of the current viewport of the current drawing in pixels (X, Y). The value 0,0 indicates, that no drawing is opened.

Properties

<i>Status</i>	Read-Only
<i>Type</i>	Screen coordinates (in pixels)
<i>Example</i>	550,283

Note

The commands 'WTILE and 'WCASCADE as well as the Windows operations Full Screen, Maximize, Minimize, etc. lead to a reset of the system variable.

SELECTBOX

This variable allows to retrieve and set the size (in pixels) of the Select Box cursor which is used when prompting the user to select objects.

Properties

<i>Status</i>	Read / Write (saved in the system's CFG file)
<i>Type</i>	Integer
<i>Default Value</i>	5

Commands

The transparent command 'PRECPAR allows the user to specify an appropriate value for his current needs.

SERNUMBER

This variable returns the serial number of the user's program license. The serial number is also displayed in the dialog displayed by the 'INFO command.

Properties

<i>Status</i>	Read-Only
---------------	-----------

Type String

SNAPBOX

This variable (Snap Cursor Box) sets the size of the snap area in pixels for object snap functions.

Properties

Status Read / Write (saved in the CFG file)

Type Integer

Default Value 5

Commands

The transparent command 'PRECPAR allows the user to specify an appropriate value for his current needs.

TABMENUAME

This variable (Tablet Menu Name) returns the filename including drive and path of the currently used tablet menu. It returns a null string ("") if no tablet menu is used.

Properties

Status Read-Only

Type String

Example "c:\felixcad\applic\tablet.mnt"

Commands

MENU, TABLET

UNDOCTL

This variable sets the status for the function to undo steps performed by a function or command. Also, it returns the current default value. The setting is bit-coded as follows:

Bit 1: Undo disabled

Bit 2: Undo all elements

Bit 3: Undo the last element or the last group

Bit 4: Undo Messages On/Off

Properties

Status Read / Write

Type Integer (bit coded)

Default Value 2

ZINSCALE

This variable (Zoom In Scale) specifies the zoom factor for the transparent command ZOOMIN.

Properties

Status Read / Write (saved in the system's CFG file)

Type Real
Default Value 0.5

Commands

The user may set a new value by using the dialog command VIEWPAR.

ZOUTSCALE

This variable (Zoom Out Scale) specifies the zoom factor for the transparent command ZOOMOUT.

Properties

Status Read / Write (saved in the system's CFG file)
Type Real
Default Value 2.0

Commands

The user may set a new value by using the dialog command VIEWPAR.

Local Variables: Reference

Local variables are drawing sensitive. Local variables of type **Read/Write** normally are stored in the drawing. Otherwise they are indicated as **Read-Only** variables. Local variables of type **Read-Only** are not stored in the drawing.

ANGBASE

Determines a default value for the zero degree angle direction (Default Angle Base). As reference point for the direction serves the positive X axis of the current user coordinate system.

Properties

Status Read / Write
Type Real
Default Value 0.0

ANGDIR

Determines the default direction when angles are specified by the user as follows:

- 0 = Counter-clockwise
- 1 = Clockwise

Properties

Status Read / Write
Type Integer
Default Value 0

ATTDIA

Controls the mode for attribute requests when inserting parts.

0 = Attribute request at command line

1 = Attribute request in a dialog box

Properties

Status Read / Write

Type Integer

Default Value 1

Command

The variable influences the command QINSERT which is used by LISP programmers in functions to insert parts or by advanced users to integrate part insertions into menus and palettes.

ATTMODE

Controls the visibility of attributes in the drawing (Attribute Mode). Valid values are:

0 = Hide all attributes (set to invisible)

1 = Visibility as determined within the attribute definition (visible | invisible)

2 = Display all attributes (visible)

Properties

Status Read / Write

Type Integer (0 | 1 | 2)

Default Value 1

Command

DRAWMODE. See also command ATTDEF for variable value 1.

ATTREQ

Determines the mode of requests of attribute values when parts (which contain attribute definitions) are inserted. Valid values are:

0 = Attribute value requests are suppressed

1 = Attribute value requests performed normally

Properties

Status Read / Write

Type Integer

Default Value 1

Command

INSERT

AUNITS

Determines the unit format for input or measurement of angles as follows:

- 0 = Decimal Degrees
- 1 = Degrees/Minutes/Seconds
- 2 = Grads
- 3 = Radians
- 4 = Surveyor's Units

Properties

Status Read / Write

Type Integer

Default Value 0

Command

AUPREC

Determines the number of decimal places for angles. The angular unit precision is specified by an integer value between 0 (no decimal places) and 8 (eight decimal places, e.g. 0.00000001)

Properties

Status Read / Write

Type Integer (0 ... 8)

Default Value 0

CECOLOR

This variable determines the current color when entities are drawn.

Properties

Status Read / Write

Type Integer

Default Value "BYLAYER" (= 256)

Command

'SETCOLOR

CELTYPE

This variable specifies the current linetype when drawing entities. The argument of the variable contains the name of the linetype.

Properties

Status Read / Write

Type String

Default Value "BYLAYER"

Command

'SETLINETYPE. See also command LINETYPE.

CHAMFERA

Determines the default value of the trim distance, in world units, associated with the first entity the user clicks on when using the edit command CHAMFER. The variable is updated when the user specifies another distance when being prompted.

Properties

Status Read / Write

Type Real

Default Value 0.0

Command

CHAMFER

CHAMFERB

Determines the default value of the trim distance, in world units, associated with the second entity the user clicks on when using the edit command CHAMFER. The variable is updated when the user specifies another distance when being prompted.

Properties

Status Read / Write

Type Real

Default Value 0.0

Command

CHAMFER

CLAYER

Determines the Current Layer. The argument of the variable contains the name of the layer.

Properties

Status Read / Write

Type String

Default Value "0"

Command

'SETLAYER, LAYER

CVPORT

This variable allows the LISP programmer to retrieve the identification number of the Current Viewport.

Properties

Status Read-Only
Type Integer
Default Value 1

See also

Lisp functions (**getactvport**) and (**setactvport**) in chapter 5.

DBMOD

This variable allows the programmer to returns a bit-coded modification status of the drawing database. Valid bit codes values are:
1, 2, 4, 8, 16

Properties

Status Read-Only
Type Bit-Code

DIMxxx

The dimensioning variables are documented below in the separate section of this manual.

DWGNAME

Returns the complete name of the current drawing as string.

Properties

Status Read-Only
Type String
Default Value "c:\applic\noname_0"

DWGPREFIX

Returns the drive and the path of the current drawing.

Properties

Status Read-Only
Type String
Default Value "c:\applic\"

DWGWRITE

Specifies the access mode to drawings of type DWG and allows protection of DWG files when these should only be displayed. Valid values are:

0 = Read-Only
1 = Read/Write

Properties

Status Read / Write
Type Integer

ELEVATION

Specifies the value for an elevation (Z axis) in the current user coordinate system. This might be used as temporary construction plane.

Properties

Status Read / Write
Type Real
Default Value 0.0

FILLETRAD

Determines the default value for the Fillet Radius when the command FILLET is used.

Properties

Status Read / Write
Type Real
Default Value 0.0

Command

FILLET

FILLMODE

Controls the fill mode for entities of type Polyline and 2D-Face. Valid values are:

0 = Display entities not filled
 1 = Display entities filled

Properties

Status Read / Write
Type Integer
Default Value 1

Commands

2DFACE, POLYLINE, CHAIN, NGON, RING

GRIDMODE

Turns displaying a grid on or off. The dots of the grid are displayed in the XY distances as specified by the system variable GRIDUNIT and in the area specified by the system variables LIMMIN and LIMMAX.

0 = Off
 1 = On

Properties

Status Read / Write
Type Integer
Default Value 0

Command

'PRECPAR

GRIDUNIT

Specifies the horizontal and vertical distances (x, y) between the grid dots when grid display is turned on (see system variable GRIDMODE).

Properties

Status Read / Write
Type 2D-Punkt
Default Value 0.0,0.0

Command

'PRECPAR

HPANG

Determines the hatch angle of a pattern file based hatch pattern. See the variables HPFILE, HPNAME, and HPSCALE.

Properties

Status Read / Write
Type Real
Default Value 0

Command

HATCH

HPDOUBLE

Determines if the built-in quick hatch pattern is drawn with single lines or with crossing lines. Valid values for this flag a:

- 0 = Single line hatch pattern
- 1 = Double line hatch pattern (cross)

Properties

Status Read / Write
Type Integer
Default Value 0

Command

HATCH

HPFILE

Determines the current complete filename for the hatch pattern file. If the variable contains a null string ("") no hatch pattern file is set to allow pattern selection in the Hatch dialog box.

Properties

Status Read / Write
Type String
Example "c:\applic\ansi.pat"

Command

HATCH

HPNAME

Determines the default name of a hatch pattern based on a hatch pattern file. If the variable contains a null string ("") the hatching commands uses the built-in quick pattern for hatching (see HPSPACE, HPANGLE, and HPDOUBLE).

Properties

Status Read / Write
Type String
Default Value ""

Command

HATCH

HPSCALE

Determines the scale factor for a hatch pattern based on a hatch pattern description file.

Properties

Status Read / Write
Type Real
Default Value 1.0

Command

HATCH

HPSPACE

Determines the distance between the lines of the standard quick pattern. See also HPNAME, HPDOUBLE, HPUSRANG.

Properties

Status Read / Write
Type Real
Default Value 1.0

Command

HATCH

HPUSRANG

Determines the hatch angle of a built-in quick hatch pattern (single line or cross pattern; see also the system variable HPDOUBLE).

Properties

Status Read / Write
Type Real
Default Value 0

Command

HATCH

INSBASE

Determines the base point for parts when writing selected entities to file to define a part. Note: The command PARTEXP does not request a base point when creating a drawing file representing a part when the entity selection method is used.

Properties

Status Read / Write
Type 3D point
Default Value 0.00,0.00,0.00

Command

PARTEXP. See also DEFPART.

INSNAME

Determines a default part name for the INSERT command. The variable is updated when the user specifies another part in the INSERT dialog. If the variable contains a null string (""), no default part name is set.

Properties

Status Read / Write
Type String
Default Value ""

Command

INSERT

LASTPOINT

Returns the coordinates of the point specified by the last point input within a drawing or editing command function. May also be set.

Properties

Status Read / Write
Type 3D Point
Default Value x, y, z

LIMMAX

Determines the upper right XY coordinates for the display of a grid in the drawing. May also, in some cases, be interpreted as drawing sheet corner.

Properties

Status Read / Write
Type 2D Point
Default Value 12.0, 9.0

Command

- - -

LIMMIN

Determines the lower left XY coordinates for the display of a grid in the drawing. May also, in some cases, be interpreted as drawing sheet corner.

Properties

Status Read / Write
Type 2D Point
Default Value 0.0, 0.0

Command

-

LSPALOAD

Determines the file name of a LISP file which is loaded automatically when the drawing is opened. Recommended for use especially in template drawings to perform certain application specific tasks (e.g. evaluation of application specific configuration files).

Properties

Status Read / Write
Type String
Default Value ""

Command

- - -

LTSCALE

Determines a common scale factor for linetypes. This allows to adjust the display of dashed, dash-dotted, or dotted lines to the drawing and plotting scale.

The dash lengths specified in the lincype definition (in drawing units) are multiplied with the factor specified in the variable LTSCALE. The variable accepts floating point values which must be greater than zero.

Properties

Status Read / Write
Type Real
Default Value 1.0

Command

'DRAWMODE. See also: LAYER, LINETYPE, 'SETLINETYPE.

LUNITS

Determines the mode for the linear unit format used in the drawing. Valid values are:

- 1 = Scientific
- 2 = Decimal
- 3 = English: Engineering
- 4 = English: Architecture
- 5 = Fraction

Properties

Status Read / Write
Type Integer (1 ... 5)
Default Value 2

Command

LUPREC

This variable (Linear Units Precision) determines the number of decimal places of linear units. A value up to 8 decimal places is accepted.

Properties

Status Read / Write
Type Integer
Default Value 2

Command

MIRRTEXT

This variable (Mirror Text) determines, if text is reflected or retains direction when selected by the MIRROR or FLIP command.

0 = Mirror text

1 = Retain text direction when mirrored

Properties

Status Read / Write

Type Integer

Default Value 1

Command

MIRROR, FLIP

ORTHOMODE

This variable turns the orthogonal drawing mode on or off.

0 = Off

1 = On

Properties

Status Read / Write

Type Integer

Default Value 0

Commands

PRECPAR, 'TORTHO

OSMODE

This variable (Object Snap Mode) sets an explicit object snap mode for drawing and editing. To set more than one object snap mode explicitly, the sum of the following valid value must be specified. Valid values for the bit-coded argument of the system variable are:

Value	Meaning
0	No explicit object snap
1	End point
2	Mid point
4	Center point of arcs and circles
8	Insertion point of text, parts, part and attribute definition and anonymous blocks like dimensioning and hatching
16	Quadrant of arcs and circles
32	Intersection point
64	Insertion point
128	Perpendicular to
256	Tangent to
512	Next point on nearest entity

Properties

Status Read / Write
Type Integer
Default Value 0

Command

'PRECPAR

PDMODE

Determines the symbol used to display point entities in the drawing.

Properties

Status Read / Write
Type Integer (0 ... 20)
Default Value 0

Command

'DRAWMODE

PDSIZE

Determines the size of point entities in the drawing.

Properties

Status Read / Write
Type Real
Default Value 0.0

Command

'DRAWMODE

PLINEWID

Determines the default value for the width when polylines are drawn with one of the commands POLYLINE, CHAIN, NGON, or RECTANGLE.

Properties

Status Read / Write
Type Real
Default Value 0.00

Commands

POLYLINE, CHAIN, NGON, RECTANGLE

PREVCMD

Returns the most recently called or executed command in a string even if the command has been canceled. The variable returns both a built-in command of the graphics engine and a command provided by a so-called "c:" function defined within the user interface. If the most recently called command has been a lisp expression, like `(setq a 4.7)`, or an unknown command, the variable contains the string "PREVCMD". The variable is of interest only for programming.

Properties

Status Read-Only
Type String
Example "_line"

SNAPBASE

Determines the origin (x, y) for the snap grid.

Properties

Status Read / Write
Type 2D Point
Default Value 0.00,0.00

Command

'PRECPAR

SNAPMODE

Determines the snap mode. If snap mode is turned on, the cursor movements and the coordinates identified are locked to the nearest point on the snap grid. If turned on, this mechanism is disabled.

0 = Off
 1 = On

Properties

Status Read / Write
Type Integer
Default Value 0

Commands

'PRECPAR, 'TSNAP

SNAPUNIT

Determines the horizontal and vertical distance of the points on the snap grid.

Properties

Status Read / Write

Type 2D Point

Default Value 1.00, 1.00

Command

'PRECPAR

SPLFRAME

Determines how to display the edges of entities of type 3DFACE which have been set to invisible are currently displayed. The value may be as follows:

0 = Edges of 3D Faces are displayed as defined (visible or invisible)

1 = All edges of 3D Faces are displayed visible

Properties

Status Read / Write

Type Integer

Default Value 0

Commands

'VIEWPAR. See also: 3DFACE.

TEXTSIZE

Determines the default value of the text size proposed by the command TEXT or ATTDEF for entity creation.

Properties

Status Read / Write

Type Real

Default Value 0.2

Commands

TEXT, ATTDEF

TEXTSTYLE

Determines the default value for the text style proposed by the command TEXT or ATTDEF for entity creation. The string must contain a valid text style name already stored in the STYLE table of the drawing database. A text style can be loaded, defined, or re-defined with the command FONT.

Properties

Status Read / Write
Type String
Default Value "STANDARD"

Commands

TEXT, ATTDET. See also: FONT.

TRIMMODE

Determines the trim mode used when executing the command FILLET or CHAMFER as follows

0 = OFF: The selected lines to be filleted or chamfered are not modified

1 = ON: The selected lines are trimmed when filleted or chamfered

Properties

Status Read / Write
Type Integer (0 | 1)
Default Value 1

Command

EDITPAR, FILLET, CHAMFER

UCSFOLLOW

Determines, if the alteration of the user coordinates system (UCS) causes an automatic change to the plan view of the new UCS.

0 = Off: A new UCS does not change the view

1 = On: Plan view follows automatically to an UCS alteration

Properties

Status Read / Write
Type Integer (0 | 1)
Default Value 0

Command

'VIEWPAR

UCSNAME

Returns the name of the current user coordinate system (UCS). If the variable is a null string (""), the UCS is unnamed.

Properties

Status Read-Only
Type String
Default Value ""

Command

UCS Control

UCSORG

Returns the coordinates of the origin of the current user coordinate system (UCS).

Properties

Status Read-Only
Type 3D Point
Default Value 0.00,0.00,0.00

Command

UCS Origin

UCSXDIR

Returns the coordinates for the direction of the X axis of the current user coordinate system.

Properties

Status Read-Only
Type 3D Point
Default Value 1.00,0.00,0.00

Command

UCS

UCSYDIR

Returns the coordinates for the direction of the Y axis of the current user coordinate system.

Properties

Status Read-Only
Type 3D Point
Default Value 0.00,1.00,0.00

Command

UCS

USERI1 ... USERI5

Stores a user-defined integer in the one of the variables USERI1 through USERI5.

Properties

Status Read / Write
Type Integer
Default Value 0

Command

USERR1 ... USERR5

Stores a user-defined floating point number in the one of the variables USERR1 through USERR5.

Properties

Status Read / Write
Type Real
Default Value 0.00

Command

USERS1 ... USERS5

Stores a user-defined string number in the one of the variables USERS1 through USERS5.

Properties

Status Read / Write
Type String
Default Value ""

VIEWCTR

Returns the coordinates (of the current user coordinate system) of the center point of the current viewport.

Properties

Status Read-Only
Type 2D Point
Default Value x, y

VIEWDIR

Returns the coordinates (of the current user coordinate system) containing the view direction of the current viewport.

Properties

Status Read-Only
Type 3D Point
Default Value 0.00,0.00,1.00

VIEWSIZE

Returns the height in drawing units of the visible drawing portion (view) in the current viewport.

Properties

Status Read-Only
Type Real
Default Value 9.00

VIEWTWIST

Returns the view angle to the current viewport.

Properties

Status Read-Only
Type Real
Default Value 0.0

WORLDUCS

Indicates, if the current user coordinate system (UCS) is identical with the world coordinate system (WCS). In this case the variable contains the integer value 1; otherwise 2.

- 1 = UCS is identical to the WCS
- 2 = UCS does not correspond to WCS

Properties

Status Read-Only
Type Integer (1 | 2)
Default Value 1

Dimension Variables: Reference

The display of dimensions are controlled by a set of dimension variables affecting the different elements of dimensions like dimension text, dimension line, extension lines, arrowheads, units of measurement etc.

The current settings of dimension variables can be stored in a dimension style within the dialog box called by the command DIMTYPE. This command also serves to restore a previously named dimension style. When programming the functions DIMSAVE and DIMREST serve for the same purposes. Named dimension styles are stored in the drawing database in the DIMSTYLE table. Also, the current setting of the individual dimension variables are stored in the drawing.

Properties

The *status* of all dimension variables is **Read/Write**.

The *storage location* of the dimension variables is in any case the drawing database.

The *data type* of the variable and the *default value* (in the case no template drawing is used) is documented with each dimension variable.

Command

The command DIMTYPE allows the user to set all of the variables described in this section.

DIMALT

0 = OFF (default) | 1 = ON

Disables or enables alternate unit dimensioning.

DIMALTD

0 .. 8 (Default: 2)

Determines the number of decimal places for alternate unit dimensioning if the variable DIMALT is set to 1 (see above) .

DIMALTF

Real (Default: 1.0)

Determines the scale factor for alternate unit linear dimensioning if the variable DIMALT is set to 1 (see above).

DIMAPOST

String (Default: "")

Determines a string appended to the measurement of an alternate unit dimension if the variable DIMALT is set to 1 (see above). Valid for all dimensions with exception of angular dimensions.

DIMASO

0 = OFF | 1 = ON (default)

Disables or enables associative dimensioning.

If associative dimensioning is turned on, all dimensions are created as anonymous block and can be modified and updated as a unique object. Also, the dimension is associated to its defining points in the geometry.

If associative dimensioning is turned off, all entities of the dimension are created as single lines, arrows, text, etc. An association to the geometry is not maintained.

DIMASZ

Real (Default: ...)

Determines the size of arrows at the end of dimension lines.

DIMBLK

String (Default: "")

Determines the name of an arrow symbol block replacing the normal arrow at the ends of dimension lines.

DIMBLK1

String (Default: "")

Determines the name of an arrow symbol block replacing the normal arrow at the starting point of a dimension line, if DIMSAH is set to 1 (see below).

DIMBLK2

String (Default: "")

Determines the name of an arrow symbol block replacing the normal arrow at the ending point of a dimension line, if DIMSAH is set to 1 (see below).

DIMCEN

Real (Default: ...)

Determines the size (in drawing units) of center marks when circles or arcs are dimensioned with one of the commands DIMCENT, DIMRAD, or DIMDIA. Positive values determine that only a center mark is drawn.

Negative values determine that center lines are drawn.

DIMCLRDR

0 .. 256

Determines the color number assigned to dimension lines, arrowheads, and dimension line leaders.

DIMCLRRE

0 .. 256

Determines the color number assigned to dimension extension lines.

DIMCLRRT

0 .. 256

Determines the color number assigned to dimension text.

DIMDLE

Real

Determines the length of the extension of the dimension line past the extension line(s) when ticks are replacing the normal arrowheads.

DIMDLI

Real

Determines the offset distance of dimension lines when base line dimensioning is applied.

DIMEXO

Real

Determines an offset of extension line starting point from the measured entity.

DIMEXE

Real

Determines a value for the length the end of extension lines exceed the dimension line.

DIMGAP

Real

Determines and offset distances around the dimension text to be maintained when the dimension label is located in between the dimension line.

DIMLAYER

String (Default: "")

Determines a layer dimensions are forced to be placed on, if the variable contains a valid string of an existing layer. A null string ("") specification determines to use the current layer for dimension objects.

DIMLFAC

Real (Default: 0.00 = not used)

Forces all linear dimensioning measurements (including diameter, radius, and ordinate dimensioning) to be multiplied with a factor specified by this dimension variable and generates dimension text containing the result of that multiplication. DIMLFAC has no effect on angular dimensioning. The factor is used as a multiplier only in the case that DIMLFAC is non-zero. The DIMLFAC variable is especially applicable when dimensioning scaled details.

DIMLIM

0 = OFF (default) | 1 = ON

Determines that the string values of the variables DIMTP and DIMTM are used as dimension label to create limit dimensioning. When DIMLIM is set to 1 (on), DIMTOL is set to 0 (off).

DIMLINE

0 = OFF (default) | 1 = ON

Determines whether the inner dimension line of linear and circle/arc dimensioning is forced to be drawn as single line (on) or not (off).

The DIMLINE variable is especially applicable when dimensioning diameters and radii.

DIMPOST

String (Default: "")

Determines a text prefix, suffix or both, for dimension label created with following dimension commands. These strings surround the dimension measurement value computed. The string <> used in the variable setting serves to separate prefix and suffix specification; otherwise the string supplied is used as suffix.

Example: A variable value of "*Approx. <> m*" might result in the dimension text *Approx. 12 m*

DIMRND

Real (Default: 0.00)

Determines to round all dimension measurements to the setting of this variable. For example, a DIMRND value set to 0.5 results to round all dimensions represented in the dimension text to the next 0.5 value.

DIMSAH

0 = OFF (default) | 1 = ON

Determines whether separate arrowhead symbol blocks are allowed (on) or not (off). The arrow blocks used are specified by the variable settings of DIMBLK1 and DIMBLK2 (see above).

DIMSCALE

Real (Default: 1.0)

Determines a general scale factor applied on all scalar dimension variables (e.g. text size, arrow size, tick size, extension line offset, ...).

Note: This variable does *not* take effect on the measured values (lengths, angles, or coordinates) by the linear, radial, angular, and ordinate dimensioning commands.

DIMSE1

0 = OFF | 1 = ON (default)

Determines whether the first extension line is displayed (1) or not (0).

DIMSE2

0 = OFF | 1 = ON (default)

Determines whether the second extension line is displayed (1) or not (0).

DIMSHO

0 = OFF | 1 = ON (default)

Determines whether the dynamic dragging mode is used when creating or modifying dimensions. Note: This variable is not stored in a dimension style.

DIMSOXD

0 = OFF(default) | 1 = ON

Suppresses dimension lines drawn outside the extension lines.

DIMSTYLE

Contains the name of the current dimension style which can be set with the DIMTYPE command.

DIMTAD

0 = OFF (default) | 1 = ON

Determines, if set to 1 (on), to place dimension text above the dimension line. Then a single solid dimension line is drawn beneath the dimension text. If activated, the variable takes effect,

- if the dimension text is drawn between the extension line (see DIMTIX) and at the same angle as the dimension line; or
- if the dimension text is placed outside of extension lines.

If DIMTAD is set to zero (off) the vertical location of dimension text in relation to the dimension line is controlled by the DIMTVP variable (see below).

DIMTFAC

Real

Determines a scale factor for the text height of tolerance values relative to the general dimension text height.

DIMTIH

0 = OFF | 1 = ON (default)

Forces dimension text to be drawn horizontally in linear, diameter, or radius dimensioning, when the dimension text is placed between the extension lines. When set to 0 (off), the angle of the dimension text takes the angle of the dimension line.

DIMTIX

0 = OFF (default) | 1 = ON

Forces dimension text to be drawn in between the extension lines.

DIMTP

String

Determines a value for the positive value of a plus/minus tolerance to be drawn. The tolerance value is only drawn if either DIMTOL or DIMLIM is set to 1(on).

DIMTM

String

Determines a value for the negative value of a plus/minus tolerance to be drawn. The tolerance value is only drawn if either DIMTOL or DIMLIM is set to 1(on).

DIMTMSTR

String

Determines an additional string for the negative value of a plus/minus tolerance to be displayed behind the dimension text. See also: DIMTPSTR.

DIMTOFL

0 = OFF | 1 = ON

Forces, if set to 1 (on), a dimension line to be drawn between the extension lines even when the dimension text is placed outside the extension line.

For dimension and radius dimensioning the variable has an additional meaning: If at the same time the variable DIMTIX is turned off (0) and DIMTOFL is turned on (1), the dimension line and the arrowheads are drawn inside the circle or arc, but the leader line and the dimension text are placed outside the circle or arc.

DIMTOH

0 = OFF | 1 = ON (default)

Forces dimension text to be drawn horizontally in linear, diameter, or radius dimensioning, when the dimension text is located outside the extension lines. When set to 0 (off), the angle of the text outside of extension lines takes the angle of the dimension line.

DIMTPSTR

String

Determines an additional string for the positive value of a plus/minus tolerance to be displayed behind the dimension text. See also: DIMTMSTR.

DIMTSTYLE

Determines the text style used when dimension text is created or modified. The string must contain a valid text style name already stored in the STYLE table of the drawing database. A text style can be loaded, defined, or re-defined with the command FONT. If the variable is set to a null string (""), the current text style specified in the local variable TEXTSTYLE (see above) is used.

DIMTSZ

Real (Default: 0.0)

Determines the length of ticks, if tick symbols are replacing arrowheads. If the variable setting of DIMTSZ is 0, arrows are drawn.

If the variable value is > 0 ticks are drawn with the size resulting from the product of $BEMSLG * BEMFKTR$.

DIMTVP

Real

Allows to adjust the vertical offset of the dimension text in relation to the dimension line. The dimension text can be placed above or below the dimension line. The magnitude of the vertical offset of the dimension text is the product of text height and the current variable setting:
 $DIMTXT * DIMTVP$.

The DIMTVP value is only used, if DIMTAD (see above) is set to off (0).

DIMTXT

Real

Determines the height of dimension text in dimension labels by a floating point number greater than zero, unless the current dimension text style has a fixed height.

DIMZIN

0, 1, 2, 3, 4, 8, 12

Determines the display mode of dimension text to suppress leading and/or trailing zeros in dimensions. The integer value affects dimension text as follows:

- 0 = Suppress zero feet and suppress precisely zero inches
- 1 = Display zero feet and display precisely zero inches
- 2 = Display zero feet, but suppress zero inches
- 3 = Suppress zero feet, but display zero inches
- 4 = Suppress leading zeros in all decimal dimensions
- 8 = Suppress trailing zeros in all decimal dimensions
- 12 = Suppress leading and trailing zeros in all decimal dimensions

Note: The values 0 through 3 affect feet/inch dimensioning only. The values 4, 8, and 12 are applied on decimal dimensions.

APPENDIX A

Entity and Table Group Codes

In this appendix you find an overview on the DXF group codes separated into the sections ENTITY, BLOCK and TABLE sections.

The programming of customized routines or of applications using the FelixCAD programming interfaces requires knowledge of so called group codes.

This knowledge is required especially, if you want to implement functions, which access to the entities or tables of the drawing database or which modify existing entities or table records (e.g.. entmake, entmod, tblmake, tblmod, etc.).

Entity Group Codes

The group code 0 describes the entity type (e.g. "LINE").

This group code may not be changed by an **entmod** function.

Within the column **Required for entmake** you find remarked, if the corresponding group code is required to create an entity, when using the **entmake** function.

Common Entity Codes

The following table documents group codes valid for all entities of the ENTITY section of the drawing database.

Note, that the values for the Entity Name (group code -1) and Handles (group code 5) are set by the system and can not be influenced by an application program.

Line types (group code 6) and Colors (group code 62) of an entity are only referenced if they differentiate from the default value BYLAYER.

Code	Type	Standard Value	Meaning
-1	Entity Name	-	Entity Name
0	String	-	Entity Type
8	String	-	Layer name
5	String	-	Handle
6	String	BYLAYER	Line type name
62	Integer	BYLAYER	Color number
210	Real	-	Extrusion direction
-2			

-3			
-4			

3DFACE

Code	Type	Required for entmake	Meaning
10	3D point	x	First corner
11	3D point	x	Second Corner
12	3D point	x	Third Corner
13	3D point	x	Fourth Corner
70	Integer	---	Invisible edge Flag 0 No invisible edges (default) 1 First edge invisible 2 Second edge invisible 4 Third edge invisible 8 Fourth edge invisible

ARC

Code	Type	Required for entmake	Meaning
10	3D point	x	Center
40	Real	x	Radius
50	Real	x	Start angle
51	Real	x	End angle

ATTDEF

Code	Type	Required for entmake	Meaning
10	3D point	x	Text start point
40	Real	x	Text height
1	String	x	Default value
3	String	x	Prompt
2	String	x	Tag string
70	Integer	-	Attribute Flag : 0 Normal (default) 1 Attribute is invisible

			2 Constant attribute 8 Preset attribute (no prompt during insertion)
50	Real	-	Rotation angle (default 0)
41	Real	-	X-scale factor (default 0)
51	Real	-	Oblique angle (default 0)
7	String	-	Text style name (default "STANDARD")
71	Integer	-	Text-generation Flag (default 0) 2 Text is mirrored in X (backward) 4 Text is mirrored in Y(upside)
72	Integer	-	Horizontal Text justification (default 0) 0 Left 1 Center 2 Right 3 Aligned 4 Middle 5 Fit
74	Integer	-	Vertical alignment (default 0) 0 Baseline 1 Bottom 2 Middle 3 Top
11	3D point	-	Alignment point (for group 72 or 74)

ATTRIB

Code	Type	Required for entmake	Meaning
10	3D point	x	Text start point
40	Real	x	Text height
1	String	x	Value
2	String	x	Attribute tag
70	Integer	-	Attribute Flag : 0 normal (default) 1 Attribute is invisible 2 Attribute is constant 8 Attribute is preset (no prompt during insertion)
50	Real	-	Rotation angle (default 0)
41	Real	-	X-scale factor (default 0)
51	Real	-	Oblique angle (default 0)

7	String	-	Text style name (default "STANDARD")
71	Integer	-	Text-generation flags (default 0) 2 Text is mirrored in X (backward) 4 Text is mirrored in Y(upside)
72	Integer	-	Horizontal Text justification (default 0) 0 Left 1 Center 2 Right 3 Aligned 4 Middle 5 Fit
74	Integer	-	Vertical Text alignment (default 0) 0 Baseline 1 Bottom 2 Middle 3 Top
11	3D point	-	Alignment point (for group 72 or 74)

CIRCLE

Code	Type	Required for entmake	Meaning
10	3D point	x	Center point
40	Real	x	Radius

DIMENSION

Code	Type	Required for entmake	Meaning
2	String	x	Name
3	String	-	Dimension style name
10	3D point	x	Definition point
11	3D point	x	Middle point of dimension text
12	3D point	x	Insertion point
70	Integer	x	Dimension type 0 Linear 1 Aligned 2 Angular 3 Diameter 4 Radius 5 Angular 3 point 6 Ordinate
1	String	-	Dimension text

13	3D point	-	Definition point (linear- or angular dimensions)
14	3D point	-	Definition point (linear- or angular dimensions)
15	3D point	-	Definition point (diameter, radius and angular dimensions)
16	3D point	-	Definition point (angular dimensions)
40	Real	-	Leader length
50	Real	-	Angle of rotated, horizontal, or vertical linear dimensions
51	Real	-	Horizontal direction
52	Real	-	Extension line angle for oblique linear dimensions
53	Real	-	Rotation angle of dimension text

INSERT

Code	Type	Required for entmake	Meaning
2	String	x	Block name
10	3D point	x	Insertion point
66	Integer	-	Attributes follow flag (default 0)
41	Real	-	X scale factor (default 1)
42	Real	-	Y scale factor (default 1)
43	Real	-	Z scale factor (default 1)
50	Real	-	Rotation angle (default 0)

The following group codes are ignored:

- 70 (column count)
- 71 (row count)
- 44 (column spacing)
- 45 (row spacing)

LINE

Code	Type	Required for entmake	Meaning
10	3D point	x	Start point
11	3D point	x	End point

POINT

Code	Type	Required for entmake	Meaning
10	3D point	x	Coordinates of the point

The group code 50 (angle of X axis for the UCS) is ignored.

POLYLINE

Code	Type	Required for entmake	Meaning
66	Integer	x	Vertices follow flag - always 1
70	Integer	(x)	Polyline flag (default 0) 1 A closed Polyline (or a polygon mesh closed in the M direction) 2 Curve-fit vertices have been added 4 B-spline-curve 8 3D-Polyline 16 Polygon mesh 32 The polygon mesh is closed in the M direction 64 Polyface mesh 128 The line type pattern is generated continuously around the vertices of this Polyline
40	Real	-	Starting width (default 0)
41	Real	-	Ending width(default 0)
71	Integer	-	Polygon mesh M vertex count (default =0)
72	Integer	-	Polygon mesh N vertex count (default =0)
73	Integer	-	Smooth surface M density (default =0)
74	Integer	-	Smooth surface N density (default =0)
75	Integer	-	Curves and smooth surface type: 0 (default) 5 Quadratic B-spline surface 6 Cubic B-spline surface 8 Bezier-surface

The group code 10 (base point) is ignored.

SEQUEND

There are no special group codes for SEQUEND.

SOLID

Code	Type	Required for entmake	Meaning
10	3D point	x	First corner
11	3D point	x	Second corner
12	3D point	x	Third corner
13	3D point	x	Fourth corner

TEXT

Code	Type	Required for entmake	Meaning
10	3D point	x	Text start point
40	Real	x	Text height
1	String	x	Text
50	Real	-	Rotation angel (default 0)
41	Real	-	X-scale factor (default 0)
51	Real	-	Oblique angle (default 0)
7	String	-	Text style name(default "STANDARD")
71	Integer	-	Text generation Flag (default 0) 2 Text is mirrored in X (backward) 4 Text is mirrored in Y (upside)
72	Integer	-	Horizontal text justification (default 0) 0 Left 1 Center 2 Right 3 Aligned 4 Middle 5 Fit
73	Integer	-	Vertical alignment (default 0) 0 Baseline 1 Bottom 2 Middle 3 Top
11	3D point	-	Alignment point (for group 72 or 73)

VERTEX

Code	Type	Required for entmake	Meaning
10	3D point	x	Definition point
40	Real	-	Starting width (default 0)
41	Real	-	Ending width(default 0)
42	Real	-	Bulge (default 0)
70	Integer	-	Vertex flags (default 0) 1 Extra vertex created by curve-fitting 2 Curve-fit tangent defined 8 Spline vertex created by spline-fitting 16 Spline frame control point 32 3D Polyline vertex 64 3D Polyline mesh vertex 128 Polyface mesh vertex
50	Real	-	Curve fit tangent direction (default 0)

Table Group Codes

The following tables are documenting the group code definitions of symbol tables.

A table contains symbols of the same nature like layer or line types.

The entries (or records) of a table define and describe the properties of a table item.

The table type is identified by the group code 0.

Customized or application defined function have access to the following tables:

APPID
BLOCK
DIMSTYLE
LAYER
LTYPE
STYLE
UCS
VIEW
VPORT

Within the column **Required for tblmake** you find remarked, if the corresponding group code is required to create a table record, when using the **tblmake** function.

APPID

In this table all used Application Names are registered. The registration of an application is necessary, if it wants to append application specific extended entity data (AED) to an entity.

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "APPID"
2	String	x	Application name
70	Integer	-	Flag value: 0 or 64 (default = 0) 64 Application name is used

BLOCK

All definitions of parts are stored in this table.

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "BLOCK"
2	String	x	Part name
70	Integer	-	Type flag 1 Anonymous Part

			2 Part has Attributes 64 Definition is referenced in the drawing Hint: Other flags are ignored.
10	3D point	x	Base point
3	String	-	Name of the drawing file (external parts)

DIMSTYLE

In this table all defined dimension styles are stored.

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "DIMSTYLE"
2	String	x	Dimension style name
3	String	x	DIMPOST
4	String	x	DIMAPOST
5	String	x	DIMBLK
6	String	x	DIMBLK1
7	String	x	DIMBLK2
40	Real	x	DIMSCALE
41	Real	x	DIMASZ
42	Real	x	DIMEXO
43	Real	x	DIMDLI
44	Real	x	DIMEXE
45	Real	x	DIMRND
46	Real	x	DIMDLE
47	Real	x	DIMTP
48	Real	x	DIMTM
140	Real	x	DIMTXT
141	Real	x	DIMCEN
142	Real	x	DIMTSZ
143	Real	x	DIMALTF
144	Real	x	DIMLFAC
145	Real	x	DIMTVP
146	Real	x	DIMTFAC
147	Real	x	DIMGAP
71	Integer	x	DIMTOL
72	Integer	x	DIMLIM

73	Integer	x	DIMTIH
74	Integer	x	DIMTOH
75	Integer	x	DIMSE1
76	Integer	x	DIMSE2
77	Integer	x	DIMTAD
78	Integer	x	DIMZIN
170	Integer	x	DIMALT
171	Integer	x	DIMALTD
172	Integer	x	DIMTOFL
173	Integer	x	DIMSAH
174	Integer	x	DIMTIX
175	Integer	x	DIMSOXD
176	Integer	x	DIMCLRD
177	Integer	x	DIMCLRE
178	Integer	x	DIMCLRT

Additional Codes:

903	String	x	DIMTPSTR
904	String	x	DIMTMSTR
907	String	x	DIMTSTYLE
908	String	x	DIMLAYER
972	Integer	x	DIMLINE

The group code 70 is ignored (always 0).

LAYER

This table manages all layers and their characteristics.

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = " LAYER "
2	String	x	Layer name
70	Integer	-	Flag (default = 0) 1 Layer is frozen 4 Layer is locked 64 Layer is referenced in the drawing Note: Other flags are ignored.

62	Integer	-	Color (default = 7)
6	String	-	Line type (default = "CONTINUOUS")

LTYPE

In this table all line type patterns are stored.

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "LTYPE"
2	String	x	Line type name
3	String	x	Description
70	Integer	-	Flag: 0 (default) 64 Line type is referenced in the drawing
73	Integer	x	Number of dash length items
40	Real	x	Total pattern length
49	Real	-	Dash length
...			

The group code 72 is ignored (always 65).

STYLE

In this table all text style definitions are stored.

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "STYLE"
2	String	x	Text style name
3	String	x	Font filename
40	Real	x	Fixed text height
41	Real	x	Width factor
42	Real	x	Last height used
50	Real	x	Oblique angle
70	Integer	-	Text style properties (default = 0) 4 Vertically 64 Text style is referenced in the drawing
71	Integer	x	Text-generation flags (default 0) 2 Text is mirrored in X (backward)

			4 Text is mirrored in Y(upside)
--	--	--	---------------------------------

UCS

This table stores all named User Coordinate Systems (UCS).

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "UCS "
2	String	x	UCS name
10	3D point	x	Origin
11	3D point	x	X axis direction
12	3D point	x	Y axis direction

The group code 70 is ignored.

VIEW

This table stores all named view.

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "VIEW "
2	String	x	View name
40	Real	x	Height
41	Real	x	Width
10	3D point	x	Center point
11	3D point	x	View direction
12	3D point	x	Target point
50	Real	-	Twist angle
71	Integer	-	View mode

The following group codes are ignored:

42	Real	Lens length
43	Real	Front clipping plane
44	Real	Back clipping plane
70	Integer	Flag

VPORT

This table contains information about the opened windows of the current drawing. In the difference to other table groups all active entries have the

name *ACTIVE (Code 2) The entries are differentiated by group code 69 (VPORT ID).

Code	Type	Required for tblmake	Meaning
0	String	x	Name of the table = "VPORT"
2	String	x	"*ACTIVE" or ""
12	2D point	x	Center point of the window
16	2D point	x	View direction
17	2D point	-	Target point
40	Real	x	Height
41	Real	x	Aspect ratio
51	Real		Twist angle
68	Integer		Status field
69	Integer	x	VPORT-ID
74	Integer	-	USCICON setting

The following group codes are ignored:

10	2D point	Lower-left corner
11	2D point	Upper-right corner
13	2D point	Snap base point
14	2D point	Snap spacing
15	2D point	Grid spacing
42	Real	Lens length
43	Real	Front clipping plane
44	Real	Back clipping plane
50	Real	Snap rotation angle
70	Integer	Flag
71	Integer	View mode
72	Integer	Circle zoom setting
73	Integer	Not evaluated at the moment
75	Integer	Snap
76	Integer	Grid
77	Integer	Not evaluated at the moment
78	Integer	Not evaluated at the moment

APPENDIX B

Command line versions for Menus, Macros, Lisp-(command ...)'s and FDT-Functions

Command Name	Description
<i>File Commands</i>	
QOPEN	Open an existing drawing
QPRINT	Print or plot current view, drawing extensions or specified area
<i>Data Exchange</i>	
QBMPOUT	Write specified window or current viewport to a BMP File
QWMFOUT	Write current viewport contents to WMF File
DXFOUT	Write current drawing to DXF File
DXFIN	Open new drawing from DXF File
<i>Menu Management ...</i>	
'PALOPEN	Open new Palette / Replace existing palette (filename is requested at command line)
'PALCLOSE	Close a Palette
PULLDOWN	Set pull-down menu (MNU)
TABLET	Open Tablet Menu (*.MNT)
TABSECTION	Replace tablet menu section
<i>Part and Hatch Insertion</i>	
QINSERT	Part Insertion
QHATCH	Crosshatching (command line)
<i>Commands for Database Table Settings</i>	
SETVIEWDIR	Set individual 3D viewpoint
DIMSAVE	Save Dimension Type (Naming the current Dimension variable settings)

DIMREST	Restore Dim Type
LOADLTYPE	Load line type pattern(s)
	<i>Misc. Commands designed for Developers</i>
MACRO	Execute macro script (MCR)
UNDO	Undo Control
	<i>Font Conversion Dialog</i>
FCOMPILE	Converts a font source file (SHP or FNT) to a binary FelixCAD Font File (FSH). Required for DXF Exchange !

Note:

- All commands listed are performed sequentially (except FCOMPILE).
- The commands PALOPEN and PALCLOSE can be used as transparent commands.
- The command sequence "QINSERT ?" displays dialog for quick part name selection. The command is influenced by settings of the system variables ATTDIA and ATTREQ.
- The Exchange commands are using no dialog for filename-request !

